ADSP-CM40x Mixed-Signal Control Processor

with ARM Cortex-M4 and 16-bit ADCs Hardware Reference

Revision 0.3, November 2015
Part Number
82-100120-01

Analog Devices, Inc.
One Technology Way
Norwood, MA 02062-9106

ANALOG DEVICES

Notices

Copyright Information

Printed in the USA.

Disclaimer

Analog Devices, Inc. reserves the right to change this product without prior notice. Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use; nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under the patent rights of Analog Devices, Inc.

Trademark and Service Mark Notice

The Analog Devices logo, Blackfin, CrossCore, EngineerZone, EZ-Board, EZ-KIT Lite, EZ-Extender, SHARC, and VisualDSP++ are registered trademarks of Analog Devices, Inc.

Blackfin+, SHARC+, and EZ-KIT Mini are trademarks of Analog Devices, Inc.

All other brand and product names are trademarks or service marks of their respective owners.

Contents

Preface

Purpose of This Manual 1–1

Intended Audience 1–1

What's New in This Manual 1–1

Technical or Customer Support 1–1

Product Information 1–2

Analog Devices Web Site 1–2

EngineerZone 1–3

Notation Conventions 1–3

Register Documentation Conventions 1–4

ARM Cortex-M4 Core Memory Sub-System

Cortex-M4 Memory Features 2–1

Cortex-M4 Memory Functional Description 2–2

ADSP-CM40x M4P Register List 2–2

ADSP-CM40x M4P Interrupt List 2–3

Cortex-M4 Memory Internal Buses Block Diagram 2–3

Cortex-M4 Memory Map 2–4

Cortex-M4 Memory 2–5

Cortex-M4 Memory Map - Code and Data Regions 2–5

Cortex-M4 Memory Accessibility - Cortex Core Perspective 2–6

Cortex-M4 Memory Accessibility - User/Application Perspective (Read Access) 2–7

Cortex-M4 Memory Accessibility - User/Application Perspective (Write Access) 2–8

Cortex-M4 Memory - Bit Banding 2–9

Cortex-M4 Memory - Translation Memory Blocks (MEMX and MEMY) 2–10

Cortex-M4 Memory - Synchronization Sequence 2–11

Cortex-M4 Cache 2–12
Cache Controller Features 2–12
Cache Structural Organization 2–12
Clearing the Cache 2–14
Bypassing the Cache 2–14
Using the Cache Counters 2–14
Using Cache Parity Control 2–15
Cortex-M4 Code and Data SRAM 2–16
SRAM Features 2–17
SRAM Block Diagram 2–17
SRAM Bank Organization 2–17
SRAM Partitioning using ConfigBanks 2–18
SRAM Interface Coherence Specification 2–20
Using Synchronization to Achieve SRAM Coherency 2–21
SRAM Write Buffers 2–21
SRAM Write Collisions and Write Priority 2–22
SRAM Access Collisions, Priority, and Stalling 2–22
SRAM Exclusive Accesses, Global Exclusive Monitor 2–23
SRAM Parity Protection 2–23
SRAM Posted System Writes (NormSysWrite versus PostSysWrite) 2–24
ADSP-CM40x M4P Register Descriptions 2–25
Bus Fault Error Information Register 2–26
Code Cache Configuration and Status Register 2–27
Cache Counter Control Register 2–33
Cache DCODE Line Fill Counter Register 2–36
Cache DCODE Miss Counter Register 2–37
Cache DCODE Reference Counter Register 2–38
Cache ICODE Line Fill Counter Register 2–39
Cache ICODE Miss Counter Register 2–40

Cache ICODE Reference Counter Register 2–41
MEMX Space Configuration Register 2–42
MEMY Space Configuration Register 2–44
Code Cache Parity Error Address Register 2–46
SRAM Configuration Register 2–47
SRAM Parity Error Address (Core) Register 2–49
SRAM Parity Error Address (DMA) Register 2–51
SysTick Calibration Register 2–53

System Crossbars (SCB)

SCB Features 3–1
SCB Functional Description 3–1
Architectural Concepts 3–2
SCB Block Diagrams 3–2
ADSP-CM40x SCB Bus Master IDs 3–6
System Crossbars 3–7
ADSP-CM40x SCB Arbitration 3–8
Clock Domain Synchronization 3–10
ADSP-CM40x SCB Register List 3–10
ADSP-CM40x SCB Programming Model 3–12
ADSP-CM40x SCB Programming Concepts 3–13
ADSP-CM40x SCB Register Descriptions 3–13
Master 0 Read Quality of Service 3–16
Master 0 IB Sync Mode 3–17
Master 0 Write Quality of Service 3–18
Master 1 Read Quality of Service 3–19
Master 1 IB Sync Mode 3–20
Master 1 Write Quality of Service 3–21
Master 2 Read Quality of Service 3–22

Master 2 Write Quality of Service 3–23
Master 3 Read Quality of Service 3–24
Master 3 Write Quality of Service 3–25
Master 4 Read Quality of Service 3–26
Master 4 Write Quality of Service 3–27
Master 5 Read Quality of Service 3–28
Master 5 Write Quality of Service 3–29
Master 6 Read Quality of Service 3–30
Master 6 Write Quality of Service 3–31
Master 7 Read Quality of Service 3–32
Master 7 Write Quality of Service 3–33
Master 8 Read Quality of Service 3–34
Master 8 Write Quality of Service 3–35
Master 9 Read Quality of Service 3–36
Master 9 Write Quality of Service 3–37
Master 10 Read Quality of Service 3–38
Master 10 Write Quality of Service 3–39
Master 11 Read Quality of Service 3–40
Master 11 Write Quality of Service 3–41
Master 12 Read Quality of Service 3–42
Master 12 Write Quality of Service 3–43
Master 13 Read Quality of Service 3–44
Master 13 Write Quality of Service 3–45
Master 14 Read Quality of Service 3–46
Master 14 Write Quality of Service 3–47
Master 15 Read Quality of Service 3–48
Master 15 Write Quality of Service 3–49
Master 16 Read Quality of Service 3–50
Master 16 Write Quality of Service 3–51
Master 17 Read Quality of Service 3–52

Master 17 Write Quality of Service 3–53
Master 18 Read Quality of Service 3–54
Master 18 Write Quality of Service 3–55
Master 19 Read Quality of Service 3–56
Master 19 Write Quality of Service 3–57
Master20 Read Quality of Service 3–58
Master 20 Write Quality of Service 3–59
Master 21 Read Quality of Service 3–60
Master 21 Write Quality of Service 3–61
Master 22 Read Quality of Service 3–62
Master 22 Write Quality of Service 3–63
Master 23 Read Quality of Service 3–64
Master 23 Write Quality of Service 3–65
Master 24 Read Quality of Service 3–66
Master 24 Write Quality of Service 3–67
Master 25 Read Quality of Service 3–68
Master 25 Write Quality of Service 3–69
Master 26 Read Quality of Service 3–70
Master 26 Write Quality of Service 3–71

Clock Generation Unit (CGU)

CGU Features 4–1
CGU Functional Description 4–1
ADSP-CM40x CGU Register List 4–2
ADSP-CM40x CGU Interrupt List 4–2
ADSP-CM40x CGU Trigger List 4–3
CGU Definitions 4–3
CGU PLL Block Diagram 4–4
CGU Operating Modes 4–5

CGU Power-up Sequence 4–5
CGU Event Control 4–5
Oscillator Watchdog 4–6
CGU Programming Model 4–8
Configuring CGU Modes 4–8
Changing Clock Frequencies 4–8
Changing the PLL Clock Frequency 4–8
Changing the CCLKn or SYSCLK Frequency Without Modifying the PLLCLK Frequency 4–9
Changing the USBCLK Clock Frequency 4–10
Changing the OUTCLK Frequency 4–10
Aligning All Clocks 4–11
Valid Clock Multiplier Settings 4–11
ADSP-CM40x CGU Register Descriptions 4–12
CLKOUT Select Register 4–13
Control Register 4–15
Clocks Divisor Register 4–17
Oscillator Watchdog Register 4–20
Status Register 4–22
Time Stamp Counter 32 LSB Register 4–26
Time Stamp Counter 32 MSB Register 4–27
Timestamp Control Register 4–28
Timestamp Counter Initial 32 LSB Value Register 4–29
Time Stamp Counter Initial MSB Value Register 4–30

System Protection Unit (SPU)

SPU Features 5–1
SPU Functional Description 5–1
ADSP-CM40x SPU Register List 5–1
Peripheral Register Write Protection 5–2
SPU Block Diagram 5–4

SPU Architectural Concepts 5–4
SPU Event Control 5–4
SPU Programming Model 5–4
SPU Mode Configuration 5–5
Locking Write-Protect Registers 5–5
Protecting a Peripheral 5–6
ADSP-CM40x SPU Register Descriptions 5–6
Control Register 5–7
Status Register 5–8
Write Protect Register n 5–9
ADSP-CM40x Write-Protect Registers 5–9

Dynamic Power Management (DPM)

DPM Features 6–1
DPM Functional Description 6–1
ADSP-CM40x DPM Interrupt List 6–1
ADSP-CM40x DPM Register List 6–1
DPM Definitions 6–2
DPM Operating Modes 6–3
Reset State 6–3
Full-on Mode 6–4
Active Mode 6–4
ACTIVE with PLL Disabled 6–4
Deep Sleep Mode 6–4
DPM Event Control 6–5
DPM Programming Model 6–5
Configuring Deep Sleep Mode 6–5
ADSP-CM40x Wake-Up Sources 6–6

ADSP-CM40x Clock Buffer Disable Bit Assignments 6–6
ADSP-CM40x DPM Register Descriptions 6–7
Core Clock Buffer Disable Register 6–8
Core Clock Buffer Enable Register 6–9
Core Clock Buffer Status Register 6–10
Core Clock Buffer Status Sticky Register 6–11
Control Register 6–12
System Clock Buffer Disable Register 6–14
Status Register 6–15
Wakeup Enable Register 6–17
Wakeup Polarity Register 6–18
Wakeup Status Register 6–19

System Event Controller (SEC)

SEC Features 7–1
SEC Functional Description 7–1
ADSP-CM40x SEC Register List 7–1
ADSP-CM40x Interrupt List 7–2
ADSP-CM40x SEC Trigger List 7–8
SEC Definitions 7–8
SEC Block Diagram 7–8
SFI Block Diagram 7–9
SEC Architectural Concepts 7–10
System Interrupt Acknowledge 7–10
Nested Vectored Interrupt Controller (NVIC) 7–10
NVIC Registers with ADSP-CM40x Specifications 7–12
System Fault Interface (SFI) and NVIC 7–12
SEC Error 7–14
SEC Programming Model 7–14
Programming Concepts 7–14

Programming Examples 7–14
Configuring a System Interrupt with NVIC 7–15
Configuring FMU as Fault Pin 7–15
Managing Faults Inside a Triggered ISR 7–15
Configuring and Managing Faults (that are also interrupts) 7–15
SEC Programming Restrictions 7–16
ADSP-CM40x SEC Register Descriptions 7–16
Fault COP Period Register 7–18
Fault COP Period Current Register 7–19
Fault Control Register 7–20
Fault Delay Register 7–23
Fault Delay Current Register 7–24
Fault End Register 7–25
Fault Source ID Register 7–26
Fault System Reset Delay Register 7–27
Fault System Reset Delay Current Register 7–28
Fault Status Register 7–29
Global Control Register 7–31
Global Status Register 7–32
Global Raise Register 7–34
Source Control Register n 7–35
Source Status Register n 7–37

Trigger Routing Unit (TRU)

TRU Features 8–1
TRU Functional Description 8–1
ADSP-CM40x TRU Register List 8–1
ADSP-CM40x TRU Interrupt List 8–2
ADSP-CM40x Trigger List 8–2
TRU Definitions 8–7

TRU Block Diagram 8–7
TRU Architectural Concepts 8–8
TRU Programming Model 8–8
Programming Concepts 8–8
Programming Example 8–9
TRU Event Control 8–9
TRU Status and Error Signals 8–9
ADSP-CM40x TRU Register Descriptions 8–9
Error Address Register 8–10
Global Control Register 8–11
Master Trigger Register 8–13
Slave Select Register 8–14
Status Information Register 8–15

Static Memory Controller (SMC)

SMC Features 9–1
SMC Definitions 9–1
SMC Functional Description 9–3
SMC Architectural Concepts 9–3
Avoiding Bus Contention 9–4
ARDY Input Control 9–5
SMC Operating Modes 9–5
Asynchronous Flash Mode 9–6
Asynchronous Page Mode 9–6
SMC Event Control 9–6
SMC Programmable Timing Characteristics 9–6
Asynchronous SRAM Reads and Writes 9–6
Asynchronous SRAM Reads with IDLE Transition Cycles Inserted 9–8
High-Speed Asynchronous SRAM Read Burst 9–8

High-Speed Asynchronous SRAM Writes 9–9
Asynchronous SRAM Reads with ARDY 9–10
Asynchronous Flash Reads 9–11
Asynchronous Flash Writes 9–13
Asynchronous Flash Page Mode Reads 9–14
Asynchronous FIFO Reads and Writes 9–15
SMC Programming Model 9–16
ADSP-CM40x SMC Register Descriptions 9–17
Bank 0 Control Register 9–18
Bank 0 Extended Timing Register 9–20
Bank 0 Timing Register 9–22
Bank 1 Control Register 9–24
Bank 1 Extended Timing Register 9–26
Bank 1 Timing Register 9–28
Bank 2 Control Register 9–30
Bank 2 Extended Timing Register 9–32
Bank 2 Timing Register 9–34
Bank 3 Control Register 9–36
Bank 3 Extended Timing Register 9–38
Bank 3 Timing Register 9–40

SPI (Internal) Flash

Features 10–1
Functional Description 10–1
Definitions 10–2
Signal Descriptions 10–4
Architectural Concepts 10–5
Memory Organization 10–5
Flash Device Protection 10–5

Operating Modes 10–8
Interface Modes 10–8
Operating Protocols 10–8
Command Skip Mode 10–10
Dummy Cycle Configuration 10–10
Programming Model 10–12
Command Definitions 10–12
Identification Operations 10–13
READ ID 10–13
Read Operations 10–14
READ 10–14
FAST READ 10–14
DUAL OUTPUT FAST READ 10–15
DUAL I/O FAST READ 10–15
QUAD OUTPUT FAST READ 10–16
QUAD I/O FAST READ 10–17
SET BURST WITH WRAP 10–18
Write Operations 10–19
WRITE ENABLE 10–19
WRITE ENABLE FOR VOLATILE STATUS REGISTER 10–20
WRITE_DISABLE 10–20
Register Operations 10–20
READ STATUS1 REGISTER 10–20
READ STATUS2 REGISTER 10–21
READ STATUS3 REGISTER 10–21
WRITE STATUS REGISTER 10–21
Program Operations 10–22
PAGE PROGRAM 10–22
Erase Operations 10–23
SECTOR ERASE 10–23
BLOCK ERASE 10–23

BULK ERASE 10–24
OTP Operations 10–24
READ OTP ARRAY 10–24
PROGRAM OTP ARRAY 10–25
ERASE OTP ARRAY 10–25
Power Loss Recovery Sequence 10–26
SPI Flash Registers 10–27
STATUS1 Register 10–27
STATUS2 Register 10–29
STATUS3 Register 10–30

Cyclic Redundancy Check (CRC)

CRC Features 11–1
CRC Functional Description 11–2
ADSP-CM40x CRC Register List 11–2
ADSP-CM40x CRC Interrupt List 11–3
CRC Definitions 11–3
CRC Block Diagram 11–4
Peripheral DMA Bus 11–5
MMR Access Bus 11–5
Mirror Block 11–6
Data FIFO 11–6
DMA Request Generator 11–6
CRC Engine 11–6
Compare Logic 11–6
CRC Architectural Concepts 11–6
Lookup Table 11–7
Data Mirroring 11–7
FIFO Status and Data Requests 11–8

CRC Operating Modes 11–9
Data Transfer Modes 11–9
Memory Scan Compute and Compare 11–10
Memory Scan Data Verify 11–10
Memory Transfer Compute and Compare 11–10
Memory Transfer Data Fill Mode 11–11
CRC Event Control 11–11
Interrupt Signals 11–11
CRC Programming Model 11–12
CRC Mode Configuration 11–12
Look-up Table Generation 11–12
Core Driven Memory Scan Compute and Compare Mode 11–13
DMA Driven Memory Scan Compute and Compare Mode 11–14
Core Driven Memory Scan Data Verify Mode 11–16
DMA Driven Memory Scan Data Verify Mode 11–17
Core Driven Memory Transfer Compute and Compare Mode 11–19
DMA Driven Memory Transfer Compute and Compare Mode 11–20
DMA Driven Memory Transfer Data Fill Mode 11–22
ADSP-CM40x CRC Peripheral and DMA Channel List 11–23
ADSP-CM40x CRC Register Descriptions 11–23
Data Compare Register 11–25
Control Register 11–26
Data Word Count Register 11–29
Data Count Capture Register 11–30
Data Word Count Reload Register 11–31
Data FIFO Register 11–32
Fill Value Register 11–33
Interrupt Enable Register 11–34
Interrupt Enable Clear Register 11–35
Interrupt Enable Set Register 11–36

Polynomial Register 11–37
CRC Current Result Register 11–38
CRC Final Result Register 11–39
Status Register 11–40

Direct Memory Access (DMA)

DMA Channel Features 12–1
DMA Channel Functional Description 12–3
ADSP-CM40x DMA Register List 12–3
ADSP-CM40x DMA Channel List 12–3
DMA Definitions 12–4
Block Diagram 12–6
Architectural Concepts 12–7
DMA Channel SCB Interface 12–7
DMA Channel Peripheral DMA Bus 12–9
Memory DMA and Triggering 12–13
DMA Channel MMR Access Bus 12–15
DMA Channel Operation Flow 12–15
DMA Channel Errors 12–21
DMA Operating Modes 12–24
Register Based Flow Modes 12–24
Stop Mode 12–25
Autobuffer Mode 12–25
Descriptor Based Flow Modes 12–25
Descriptor-Array Mode 12–26
Descriptor-List Mode 12–26
Descriptor-On-Demand Modes 12–27
Data Transfer Modes 12–28
Two-Dimensional DMA 12–28
DMA Channel Event Control 12–29

Event Signals 12–30
Work Unit State Events 12–30
Peripheral Interrupt-Request Events 12–31
Peripheral Data-Request Events 12–31
DMA Channel Triggers 12–32
Issuing Triggers 12–32
Waiting For Triggers 12–32
DMA Channel Programming Model 12–33
Mode Configuration 12–34
Register-Based Linear-Buffer Stop Flow Mode 12–34
Register-Based Autobuffer Flow Mode 12–35
Descriptor-Array Flow Mode 12–36
Descriptor-List Flow Mode 12–37
Register-Based Memory-to-Memory Transfer-in-Stop Flow Mode 12–37
Programming Concepts 12–39
Synchronization of Software and DMA 12–39
Descriptor Queues 12–40
ADSP-CM40x DMA Register Descriptions 12–43
Start Address of Current Buffer Register 12–44
Current Address Register 12–45
Bandwidth Limit Count Register 12–46
Bandwidth Limit Count Current Register 12–47
Bandwidth Monitor Count Register 12–48
Bandwidth Monitor Count Current Register 12–49
Configuration Register 12–50
Current Descriptor Pointer Register 12–58
Pointer to Next Initial Descriptor Register 12–59
Previous Initial Descriptor Pointer Register 12–60
Status Register 12–61
Inner Loop Count Start Value Register 12–64

Current Count(1D) or intra-row XCNT (2D) Register 12–65
Inner Loop Address Increment Register 12–66
Outer Loop Count Start Value (2D only) Register 12–67
Current Row Count (2D only) Register 12–68
Outer Loop Address Increment (2D only) Register 12–69

General-Purpose Ports (PORT)

PORT Features 13–1
PORT Functional Description 13–2
ADSP-CM40x PORT Register List 13–2
ADSP-CM40x PADS Register List 13–3
ADSP-CM40x PINT Register List 13–3
ADSP-CM40x PINT Interrupt List 13–4
ADSP-CM40x PINT Trigger List 13–4
PORT Definitions 13–5
PORT Architectural Concepts 13–5
Internal Interfaces 13–5
External Interfaces 13–5
GPIO Functionality 13–5
Port Multiplexing Control 13–6
PORT Event Control 13–7
PORT Interrupt Signals 13–7
PORT Programming Model 13–9
ADSP-CM40x PORT Register Descriptions 13–12
Port x GPIO Data Register 13–14
Port x GPIO Data Clear Register 13–17
Port x GPIO Data Set Register 13–20
Port x GPIO Output Toggle Register 13–23
Port x GPIO Direction Register 13–26
Port x GPIO Direction Clear Register 13–30

Port x GPIO Direction Set Register 13–33
Port x Function Enable Register 13–36
Port x Function Enable Clear Register 13–39
Port x Function Enable Set Register 13–42
Port x GPIO Input Enable Register 13–45
Port x GPIO Input Enable Clear Register 13–48
Port x GPIO Input Enable Set Register 13–51
Port x GPIO Lock Register 13–54
Port x Multiplexer Control Register 13–56
Port x GPIO Polarity Invert Register 13–58
Port x GPIO Polarity Invert Clear Register 13–61
Port x GPIO Polarity Invert Set Register 13–64
ADSP-CM40x PINT Register Descriptions 13–66
Pint Assign Register 13–67
Pint Edge Clear Register 13–68
Pint Edge Set Register 13–71
Pint Invert Clear Register 13–74
Pint Invert Set Register 13–77
Pint Latch Register 13–80
Pint Mask Clear Register 13–84
Pint Mask Set Register 13–87
Pint Pinstate Register 13–90
Pint Request Register 13–94
ADSP-CM40x PADS Register Descriptions 13–98
Peripheral Configuration0 Register 13–99

General-Purpose Timer (TIMER)

GP Timer Features 14–1
ADSP-CM40x TIMER Register List 14–2
ADSP-CM40x TIMER Interrupt List 14–3

ADSP-CM40x TIMER Trigger List 14–3
Internal Interface 14–4
External Interface 14–4
GP Timer Operating Modes 14–5
General Operation 14–5
Period, Width and Delay Register Interaction 14–6
Single-Pulse PWMOUT Mode 14–7
Continuous PWMOUT Mode 14–7
Width Capture (WIDCAP) Mode 14–9
Width Capture Mode Overflow 14–12
Windowed Watchdog (WATCHDOG) Modes 14–14
Windowed Watchdog Width Mode 14–14
Windowed Watchdog Period Mode 14–16
Pin Interrupt (PININT) Mode 14–18
External Clock (EXTCLK) Mode 14–18
Trigger Slave Mode 14–19
GP Timer Programming Concepts 14–21
Setting Up Constantly Changing Timer Conditions 14–21
Configuring, Enabling, and Disabling One or More Timers 14–21
Configuring Timer Data and Status Interrupts 14–22
Using the Timer Broadcast Feature 14–22
Timer Illegal States 14–22
Continuous PWMOUT Mode 14–23
Single Pulse PWMOUT Mode 14–24
WIDCAP Mode 14–24
EXTCLK Mode 14–25
WATCHDOG Events 14–25
ADSP-CM40x TIMER Register Descriptions 14–26
Broadcast Delay Register 14–28

Broadcast Period Register 14–29
Broadcast Width Register 14–30
Data Interrupt Latch Register 14–31
Data Interrupt Mask Register 14–32
Error Type Status Register 14–33
Run Register 14–35
Run Clear Register 14–36
Run Set Register 14–37
Status Interrupt Latch Register 14–38
Status Interrupt Mask Register 14–39
Stop Configuration Register 14–40
Stop Configuration Clear Register 14–41
Stop Configuration Set Register 14–42
Timer n Configuration Register 14–43
Timer n Counter Register 14–47
Timer n Delay Register 14–48
Timer n Period Register 14–49
Timer n Width Register 14–50
Trigger Slave Enable Register 14–51
Trigger Master Mask Register 14–52

Capture Timer (CPTMR)

CPTMR Functional Description 15–1
CPTMR Block Diagram 15–1
ADSP-CM40x CPTMR Register List 15–2
ADSP-CM40x CPTMR Trigger List 15–3
ADSP-CM40x CPTMR Interrupt List 15–3
CPTMR Operation 15–3
CPTMR Interrupt Signals 15–4

CPTMR Programming Model (ADSP-CM40x) 15–5
ADSP-CM40x CPTMR Register Descriptions 15–5
Configuration Register 15–7
Counter Register 15–8
Data Interrupt Latch Status Register 15–9
Data Interrupt Mask Register 15–10
Data Interrupt Mask Clear Register 15–11
Data Interrupt Mask Set Register 15–12
Run Register 15–13
Run Clear Register 15–14
Run Set Register 15–15
Interrupt Latch Status Register 15–16
Status Interrupt Mask Register 15–17
Status Interrupt Mask Clear Register 15–18
Status Interrupt Mask Set Register 15–19
On-time Capture Register 15–20

Watchdog Timer (WDOG)

WDOG Features 16–1
WDOG Functional Description 16–1
ADSP-CM40x WDOG Register List 16–1
ADSP-CM40x WDOG Interrupt List 16–2
WDOG Block Diagram 16–2
Internal Interface 16–2
External Interface 16–2
ADSP-CM40x WDOG Register Descriptions 16–3
Count Register 16–4
Control Register 16–5
Watchdog Timer Status Register 16–6

General-Purpose Counter (CNT)

GP Counter Features 17–1

GP Counter Functional Description 17–2

ADSP-CM40x CNT Register List 17–2

ADSP-CM40x CNT Interrupt List 17–3

ADSP-CM40x CNT Trigger List 17–3

GP Counter Operating Modes 17–4

Quadrature Encoder Mode 17–4

Binary Encoder Mode 17–5

Up/Down Counter Mode 17–5

Direction Counter Mode 17–5

Timed Direction Mode 17–5

M/N Scaling 17–6

M/N Stop Detection 17–8

M/N Error Condition 17–9

M/N Restrictions 17–9

GP Counter Programming Model 17–9

GP Counter General Programming Flow 17–9

M/N Scaling Programming Guidelines 17–10

GP Counter Mode Configuration 17–10

Configuring GP Counter Push-Button Operation 17–10

Configuring Zero-Marker-Zeros-Counter Mode 17–10

Configuring Zero-Marker-Error Mode 17–11

Configuring Zero-Once Mode 17–11

Configuring Boundary Auto-Extend Mode 17–11

Configuring Boundary Capture Mode 17–12

Configuring Boundary Compare and Boundary Zero Modes 17–12

Configuring GP Counter Push-Button Operation 17–13

GP Counter Programming Concepts 17–13

CNT Input Noise Filtering 17–13

Capturing Counter Interval and CNT_CNTR Read Timing 17–13
Capturing Time Interval Between Successive Counter Events 17–15
GP Counter Event Control 17–15
Illegal Gray and Binary Code Events 17–16
Up/Down Count Events 17–16
Zero-Count Events 17–16
Overflow Events 17–16
Boundary Match Events 17–17
Zero Marker Events 17–17
ADSP-CM40x CNT Register Descriptions 17–17
Configuration Register 17–18
Command Register 17–21
Counter Register 17–23
Debounce Register 17–24
Interrupt Mask Register 17–26
Maximum Count Register 17–29
M Value for Divider 17–30
Minimum Count Register 17–31
N Value for Divider 17–32
Status Register 17–33

Pulse-Width Modulator (PWM)

PWM Features 18–1
Functional Description 18–1
ADSP-CM40x PWM Trigger List 18–2
ADSP-CM40x PWM Interrupt List 18–2
ADSP-CM40x PWM Register List 18–3
Architectural Concepts 18–5
Block Diagram 18–5
Timer Units 18–6

Channel Timing Control Unit 18–12
Gate Drive Unit 18–23
Output Control Feature Precedence 18–24
Operating Modes 18–24
Sync Operation Modes 18–24
Internal PWM SYNC Generation 18–25
External PWM SYNC Generation 18–25
Output Disable and Cross-Over Modes 18–25
Brushless DC Motor (Electronically Commutated Motor) Control 18–26
Trigger Slave Mode 18–27
Heightened-Precision Edge Placement 18–28
Sample Waveforms for High- and Low-Side with Precision Placement 18–31
Emulation Mode 18–33
Event Control 18–33
Trip Control Unit 18–34
Programming Model 18–36
Programming Model for 3-Phase AC Motor Control 18–36
PWM Initialization for Motor Control 18–38
PWM Enable for Motor Control 18–40
PWM Response to Sync Interrupt for Motor Control 18–40
PWM Disable (and Stop the Motor) for Motor Control 18–40
ADSP-CM40x PWM Register Descriptions 18–41
Channel A Control Register 18–44
Channel A-High Duty-0 Register 18–46
Channel A-High Heightened-Precision Duty-0 Register 18–47
Channel A-High Duty-1 Register 18–48
Channel A-High Heightened-Precision Duty-1 Register 18–49
Channel A-High Full Duty0 Register 18–50
Channel A-High Full Duty1 Register 18–51
Channel A-Low Duty-0 Register 18–52

Channel A-Low Heightened-Precision Duty-0 Register 18–53
Channel A-Low Duty-1 Register 18–54
Channel A-Low Heightened-Precision Duty-1 Register 18–55
Channel A-Low Full Duty0 Register 18–56
Channel A-Low Full Duty1 Register 18–57
Channel B Control Register 18–58
Channel B-High Duty-0 Register 18–60
Channel B-High Heightened-Precision Duty-0 Register 18–61
Channel B-High Duty-1 Register 18–62
Channel B-High Heightened-Precision Duty-1 Register 18–63
Channel B-High Full Duty0 Register 18–64
Channel B-High Full Duty1 Register 18–65
Channel B-Low Duty-0 Register 18–66
Channel B-Low Heightened-Precision Duty-0 Register 18–67
Channel B-Low Duty-1 Register 18–68
Channel B-Low Heightened-Precision Duty-1 Register 18–69
Channel B-Low Full Duty0 Register 18–70
Channel B-Low Full Duty1 Register 18–71
Channel C Control Register 18–72
Channel C-High Pulse Duty Register 0 18–74
Channel C-High Pulse Heightened-Precision Duty Register 0 18–75
Channel C-High Pulse Duty Register 1 18–76
Channel C-High Pulse Heightened-Precision Duty Register 1 18–77
Channel Configuration Register 18–78
Chop Configuration Register 18–85
Channel C-High Full Duty0 Register 18–86
Channel C-High Full Duty1 Register 18–87
Channel C-Low Pulse Duty Register 0 18–88
Channel C-Low Pulse Duty Register 1 18–89
Channel C-Low Duty-1 Register 18–90

Channel C-Low Heightened-Precision Duty-1 Register 18–91
Channel C-Low Full Duty0 Register 18–92
Channel C-Low Full Duty1 Register 18–93
Control Register 18–94
Channel D Control Register 18–98
Channel D-High Duty-0 Register 18–100
Channel D-High Pulse Heightened-Precision Duty Register 0 18–101
Channel D-High Pulse Duty Register 1 18–102
Channel D High Pulse Heightened-Precision Duty Register 1 18–103
Channel D-High Full Duty0 Register 18–104
Channel D-High Full Duty1 Register 18–105
Channel D-Low Pulse Duty Register 0 18–106
Channel D-Low Heightened-Precision Duty-0 Register 18–107
Channel D-Low Pulse Duty Register 1 18–108
Channel D-Low Heightened-Precision Duty-1 Register 18–109
Channel A Delay Register 18–110
Channel B Delay Register 18–111
Channel C Delay Register 18–112
Channel D Delay Register 18–113
Channel D-Low Full Duty0 Register 18–114
Channel D-Low Full Duty1 Register 18–115
Dead Time Register 18–116
Interrupt Latch Register 18–117
Interrupt Mask Register 18–119
Status Register 18–121
Sync Pulse Width Register 18–126
Timer 0 Period Register 18–127
Timer 1 Period Register 18–128
Timer 2 Period Register 18–129
Timer 3 Period Register 18–130

Timer 4 Period Register 18–131
Trip Configuration Register 18–132

Universal Asynchronous Receiver/Transmitter (UART)

UART Features 19–1
UART Functional Description 19–2
ADSP-CM40x UART Register List 19–2
ADSP-CM40x UART Interrupt List 19–3
UART Block Diagram 19–4
UART Architectural Concepts 19–4
Internal Interface 19–4
External Interface 19–5
Hardware Flow Control 19–5
Bit Rate Generation 19–6
Autobaud Detection 19–7
UART Debug Features 19–8
UART Operating Modes 19–9
UART Mode 19–9
IrDA SIR Mode 19–9
Multi-Drop Bus Mode 19–10
UART Data Transfer Modes 19–11
UART Mode Transmit Operation (Core) 19–11
UART Mode LIN Break Command 19–11
UART Mode Receive Operation (Core) 19–12
IrDA Transmit Operation 19–13
IrDA Receive Operation 19–14
MDB Transmit Operation 19–15
MDB Receive Operation 19–15
DMA Mode 19–16
Mixing DMA and Core Modes 19–17
Setting Up Hardware Flow Control 19–17

UART Event Control 19–17
Interrupt Masks 19–18
Interrupt Servicing 19–18
Transmit Interrupts 19–18
Receive Interrupts 19–20
Status Interrupts 19–21
Multi-Drop Bus Events 19–22
UART Programming Model 19–22
Detecting Autobaud 19–22
Using Common Initialization Steps 19–22
Using Core Transfers 19–23
Using DMA Transfers 19–23
Using Interrupts 19–23
Setting Up Hardware Flow Control 19–23
ADSP-CM40x UART Register Descriptions 19–23
Clock Rate Register 19–25
Control Register 19–26
Interrupt Mask Register 19–32
Interrupt Mask Clear Register 19–36
Interrupt Mask Set Register 19–38
Receive Buffer Register 19–40
Receive Shift Register 19–41
Receive Counter Register 19–42
Scratch Register 19–43
Status Register 19–44
Transmit Address/Insert Pulse Register 19–49
Transmit Hold Register 19–50
Transmit Shift Register 19–51
Transmit Counter Register 19–52

Two-Wire Interface (TWI)

TWI Features 20–1

TWI Functional Description 20–2

ADSP-CM40x TWI Register List 20–2

ADSP-CM40x TWI Interrupt List 20–3

External Interface 20–3

Serial Clock Signal (SCL) 20–3

Serial Data Signal (SDA) 20–4

Internal Interface 20–4

TWI Architectural Concepts 20–4

TWI Protocol 20–5

Clock Generation and Synchronization 20–5

Bus Arbitration 20–6

Start and Stop Conditions 20–6

General Call Support 20–7

Fast Mode 20–7

TWI Operating Modes 20–7

Repeated Start 20–7

Transmit Receive Repeated Start 20–8

Receive Transmit Repeated Start 20–8

Clock Stretching 20–9

Clock Stretching During FIFO Underflow 20–9

Clock Stretching During FIFO Overflow 20–10

Clock Stretching During Repeated Start 20–11

TWI Programming Model 20–12

General Setup 20–12

Slave Mode 20–12

Master Mode Program Flow 20–13

Master Mode Clock Setup 20–14

Master Mode Transmit 20–14

Master Mode Receive 20–15
ADSP-CM40x TWI Register Descriptions 20–16
SCL Clock Divider Register 20–17
Control Register 20–18
FIFO Control Register 20–20
FIFO Status Register 20–22
Interrupt Mask Register 20–23
Interrupt Status Register 20–25
Master Mode Address Register 20–28
Master Mode Control Registers 20–29
Master Mode Status Register 20–32
Rx Data Double-Byte Register 20–35
Rx Data Single-Byte Register 20–36
Slave Mode Address Register 20–37
Slave Mode Control Register 20–38
Slave Mode Status Register 20–40
Tx Data Double-Byte Register 20–41
Tx Data Single-Byte Register 20–42

Controller Area Network (CAN)

CAN Features 21–1
CAN Functional Description 21–2
ADSP-CM40x CAN Register List 21–2
ADSP-CM40x CAN Interrupt List 21–4
External Interface 21–4
Architectural Concepts 21–4
Block Diagram 21–5
Mailbox Control 21–6
Protocol Fundamentals 21–7
Data Transfer Modes 21–8

Transmit Operations 21–8
Receive Operation 21–10
Watchdog Mode 21–12
Timestamps 21–13
Remote Frame Handling 21–13
Temporarily Disabling CAN Mailbox 21–14
CAN Operating Modes 21–15
Bit Timing 21–15
CAN Low Power Features 21–16
Built-In Suspend Mode 21–17
Built-In Sleep Mode 21–17
Soft Reset 21–17
CAN Event Control 21–17
CAN Interrupt Signals 21–18
Mailbox Interrupts 21–18
Global Interrupt 21–18
Event Counter 21–20
CAN Warnings and Errors 21–20
Programmable Warning Limits 21–20
Error Handling 21–21
Error Frames 21–21
Error Levels 21–22
CAN Debug and Test Modes 21–23
ADSP-CM40x CAN Register Descriptions 21–25
Abort Acknowledge 1 Register 21–28
Abort Acknowledge 2 Register 21–29
Acceptance Mask (H) Register 21–30
Acceptance Mask (L) Register 21–31
Error Counter Register 21–32
Clock Register 21–33

CAN Master Control Register 21–34
Debug Register 21–36
Error Status Register 21–38
Error Counter Warning Level Register 21–40
Global CAN Interrupt Flag Register 21–41
Global CAN Interrupt Mask Register 21–44
Global CAN Interrupt Status Register 21–46
Interrupt Pending Register 21–49
Mailbox Interrupt Mask 1 Register 21–51
Mailbox Interrupt Mask 2 Register 21–52
Mailbox Receive Interrupt Flag 1 Register 21–53
Mailbox Receive Interrupt Flag 2 Register 21–54
Temporary Mailbox Disable Register 21–55
Mailbox Transmit Interrupt Flag 1 Register 21–56
Mailbox Transmit Interrupt Flag 2 Register 21–57
Mailbox Word 0 Register 21–58
Mailbox Word 1 Register 21–59
Mailbox Word 2 Register 21–60
Mailbox Word 3 Register 21–61
Mailbox ID 0 Register 21–62
Mailbox ID 1 Register 21–63
Mailbox Length Register 21–64
Mailbox Timestamp Register 21–65
Mailbox Configuration 1 Register 21–66
Mailbox Configuration 2 Register 21–67
Mailbox Direction 1 Register 21–68
Mailbox Direction 2 Register 21–69
Overwrite Protection/Single Shot Transmission 1 Register 21–70
Overwrite Protection/Single Shot Transmission 2 Register 21–71
Remote Frame Handling 1 Register 21–72

Remote Frame Handling 2 Register 21–73
Receive Message Lost 1 Register 21–74
Receive Message Lost 2 Register 21–75
Receive Message Pending 1 Register 21–76
Receive Message Pending 2 Register 21–77
Status Register 21–78
Transmission Acknowledge 1 Register 21–80
Transmission Acknowledge 2 Register 21–81
Timing Register 21–82
Transmission Request Reset 1 Register 21–83
Transmission Request Reset 2 Register 21–84
Transmission Request Set 1 Register 21–85
Transmission Request Set 2 Register 21–86
Universal Counter Configuration Mode Register 21–87
Universal Counter Register 21–89
Universal Counter Reload/Capture Register 21–90

Universal Serial Bus (USB)

USB Features 22–1
USB Functional Description 22–2
USB Architectural Concepts 22–2
Multi-Point Support 22–3
On-Chip Bus Interfaces 22–3
FIFO Configuration 22–4
Clocking 22–4
UTMI Interface 22–5
ADSP-CM40x USB Register List 22–5
ADSP-CM40x USB Interrupt List 22–8
ADSP-CM40x USB Trigger List 22–8
USB Block Diagram 22–8

USB Definitions 22–9
USB References 22–11
USB Operating Modes 22–11
Peripheral Mode 22–12
Endpoint Setup 22–12
IN Transactions as a Peripheral 22–12
OUT Transactions as a Peripheral 22–13
Peripheral Transfer Work Flows 22–14
Peripheral Mode Suspend 22–27
Start-of-frame (SOF) Packets 22–27
Soft Connect/Soft Disconnect 22–27
Error Handling As a Peripheral 22–27
Stalls Issued to Control Transfers 22–28
Zero Length OUT Data Packets in Control Transfers 22–29
Host Mode 22–29
Transaction Scheduling 22–29
Endpoint Setup and Data Transfer 22–30
Control Transaction as a Host 22–30
Set up Phase as a Host 22–30
IN Data Phase as a Host 22–31
OUT Data as a Host (Control) 22–32
IN Status Phase as a Host (Following SETUP Phase or OUT Data Phase) 22–32
OUT Status Phase as a Host (Following IN Data Phase) 22–33
Host IN Transactions 22–34
Host OUT Transactions 22–34
Multi-Point Support 22–35
Babble Interrupt 22–37
VBUS Events 22–37
Host Mode Reset 22–39
Host Mode Suspend 22–39
Suspending and Resuming the Controller 22–39

USB Event Control 22–43
Interrupt Signals 22–43
Interrupt Handling 22–44
Reset Signals 22–45
Reset in Peripheral Mode 22–45
USB Reset in Host Mode 22–46
USB Programming Model 22–46
Peripheral Mode Flow Charts 22–46
Host Mode Flow Charts 22–53
DMA Mode Flow Charts 22–58
OTG Session Request 22–62
Starting a Session 22–62
Detecting Activity 22–63
Host Negotiation Protocol 22–63
Data Transfer 22–64
Loading or Unloading Packets from Endpoints 22–64
DMA Master Channels 22–64
DMA Bus Cycles 22–66
Transferring Packets Using DMA 22–67
Individual Rx Endpoint Packet 22–67
Individual Tx Endpoint Packet 22–68
Multiple Rx Endpoint Packets 22–68
Multiple Tx Endpoint Packets 22–69
ADSP-CM40x USB Register Descriptions 22–70
Device Control Register 22–73
DMA Channel n Address Register 22–75
DMA Channel n Count Register 22–76
DMA Channel n Control Register 22–77
DMA Interrupt Register 22–80
EP0 Configuration Information Register 22–82

EP0 Number of Received Bytes Register 22–84
EP0 Configuration and Status (Host) Register 22–85
EP0 Configuration and Status (Peripheral) Register 22–88
EP0 NAK Limit Register 22–91
EP0 Connection Type Register 22–92
EP0 Configuration Information Register 22–93
EP0 Number of Received Bytes Register 22–95
EP0 Configuration and Status (Host) Register 22–96
EP0 Configuration and Status (Peripheral) Register 22–99
EP0 NAK Limit Register 22–102
EP0 Connection Type Register 22–103
Endpoint Information Register 22–104
EPn Number of Bytes Received Register 22–105
EPn Receive Configuration and Status (Host) Register 22–106
EPn Receive Configuration and Status (Peripheral) Register 22–111
EPn Receive Polling Interval Register 22–115
EPn Receive Maximum Packet Length Register 22–116
EPn Receive Type Register 22–117
EPn Transmit Configuration and Status (Host) Register 22–119
EPn Transmit Configuration and Status (Peripheral) Register 22–123
EPn Transmit Polling Interval Register 22–126
EPn Transmit Maximum Packet Length Register 22–127
EPn Transmit Type Register 22–128
FIFO Size 22–130
FIFO Size 22–131
EPn Number of Bytes Received Register 22–132
EPn Receive Configuration and Status (Host) Register 22–133
EPn Receive Configuration and Status (Peripheral) Register 22–138
EPn Receive Polling Interval Register 22–142
EPn Receive Maximum Packet Length Register 22–143

EPn Receive Type Register 22–144
EPn Transmit Configuration and Status (Host) Register 22–146
EPn Transmit Configuration and Status (Peripheral) Register 22–150
EPn Transmit Polling Interval Register 22–153
EPn Transmit Maximum Packet Length Register 22–154
EPn Transmit Type Register 22–155
Function Address Register 22–157
FIFO Byte (8-Bit) Register 22–158
FIFO Half-Word (16-Bit) Register 22–159
FIFO Word (32-Bit) Register 22–160
Frame Number Register 22–161
Full-Speed EOF 1 Register 22–162
ID Control 22–163
Common Interrupts Enable Register 22–164
Index Register 22–166
Receive Interrupt Register 22–167
Receive Interrupt Enable Register 22–168
Transmit Interrupt Register 22–169
Transmit Interrupt Enable Register 22–170
Common Interrupts Register 22–171
Link Information Register 22–173
LPM Attribute Register 22–174
LPM Control Register 22–175
LPM Function Address Register 22–177
LPM Interrupt Enable Register 22–178
LPM Interrupt Status Register 22–180
Low-Speed EOF 1 Register 22–183
MPn Receive Function Address Register 22–184
MPn Receive Hub Address Register 22–185
MPn Receive Hub Port Register 22–186

MPn Transmit Function Address Register 22–187
MPn Transmit Hub Address Register 22–188
MPn Transmit Hub Port Register 22–189
FS PHY Control 22–190
FS PHY Status 22–191
Power and Device Control Register 22–192
RAM Information Register 22–195
EPn Request Packet Count Register 22–196
RX Double Packet Buffer Disable for Endpoints 1 to 3 22–197
Software Reset Register 22–198
Testmode Register 22–199
TX Double Packet Buffer Disable for Endpoints 1 to 3 22–200
VBUS Control Register 22–201
VBUS Pulse Length Register 22–202

Ethernet Media Access Controller (EMAC)

EMAC Features 23–1
EMAC Functional Description 23–2
ADSP-CM40x EMAC Register List 23–2
ADSP-CM40x EMAC Interrupt List 23–7
ADSP-CM40x EMAC Trigger List 23–7
EMAC Definitions 23–7
EMAC Block Diagram and Interfaces 23–8
EMAC CORE Sub-Blocks 23–10
EMAC PHY Interface 23–11
Clock Sources 23–12
EMAC Architectural Concepts 23–13
EMAC System Crossbar Interface (EMAC SCB) 23–13
DMA Controller (EMAC DMA) 23–17
EMAC FIFO Layer (EMAC MFL) 23–35

EMAC CORE 23–38
EMAC Station Management Interface (SMI) 23–50
EMAC Management Counters (MMC) 23–53
EMAC Precision Time Protocol (PTP) Engine 23–55
EMAC Event Control 23–74
EMAC Interrupt Signals 23–75
PHYINT Interrupt Signal 23–76
EMAC Programming Model 23–77
EMAC Programming Steps 23–77
DMA Initialization 23–77
EMAC CORE Initialization 23–78
Performing Normal Transmit and Receive Operations 23–79
Stopping and Starting Transfers 23–79
Interrupts and Interrupt Service Routines 23–80
Enabling Checksum for Transmit and Receive 23–81
Programming the System Time Module 23–81
Programming the PTP for Frame Detection and Timestamping 23–82
Programming for Auxiliary Timestamps 23–83
Programming Fixed Pulse-Per-Second Output 23–83
Programming Flexible Pulse-Per-Second Output 23–83
EMAC Programming Concepts 23–84
IEEE 802.3 Ethernet Packet Structure 23–84
Frame Size Statistics for Application Software 23–85
Software Visualization of Programmable Packet Size 23–85
Ethernet Packet Structure in C 23–86
DMA Descriptor Implementation in C 23–86
PTP Header Structure in C 23–87
ADSP-CM40x EMAC Register Descriptions 23–87
MAC Address 0 High Register 23–92
MAC Address 0 Low Register 23–93

Debug Register **23–94**
DMA SCB Bus Mode Register **23–97**
DMA SCB Status Register **23–99**
DMA Bus Mode Register **23–100**
DMA Interrupt Enable Register **23–103**
DMA Missed Frame Register **23–106**
DMA Operation Mode Register **23–107**
DMA Rx Buffer Current Register **23–111**
DMA Rx Descriptor List Address Register **23–112**
DMA Rx Descriptor Current Register **23–113**
DMA Rx Interrupt Watch Dog Register **23–114**
DMA Rx Poll Demand register **23–115**
DMA Status Register **23–116**
DMA Tx Buffer Current Register **23–120**
DMA Tx Descriptor List Address Register **23–121**
DMA Tx Descriptor Current Register **23–122**
DMA Tx Poll Demand Register **23–123**
FLow Control Register **23–124**
Hash Table High Register **23–126**
Hash Table Low Register **23–127**
Interrupt Mask Register **23–128**
MMC IPC Rx Interrupt Mask Register **23–129**
MMC IPC Rx Interrupt Register **23–135**
Interrupt Status Register **23–140**
MAC Configuration Register **23–142**
MAC Rx Frame Filter Register **23–146**
MMC Control Register **23–149**
MMC Rx Interrupt Mask Register **23–151**
MMC Rx Interrupt Register **23–154**
MMC TX Interrupt Mask Register **23–157**

MMC Tx Interrupt Register 23–161
Rx 1024- to Max-Byte Frames (Good/Bad) Register 23–165
Rx 128- to 255-Byte Frames (Good/Bad) Register 23–166
Rx 256- to 511-Byte Frames (Good/Bad) Register 23–167
Rx 512- to 1023-Byte Frames (Good/Bad) Register 23–168
Rx 64-Byte Frames (Good/Bad) Register 23–169
Rx 65- to 127-Byte Frames (Good/Bad) Register 23–170
Rx alignment Error Register 23–171
Rx Broadcast Frames (Good) Register 23–172
Rx CRC Error Register 23–173
Rx FIFO Overflow Register 23–174
Rx Frame Count (Good/Bad) Register 23–175
Rx ICMP Error Frames Register 23–176
Rx ICMP Error Octets Register 23–177
Rx ICMP Good Frames Register 23–178
Rx ICMP Good Octets Register 23–179
Rx IPv4 Datagrams Fragmented Frames Register 23–180
Rx IPv4 Datagrams Fragmented Octets Register 23–181
Rx IPv4 Datagrams (Good) Register 23–182
Rx IPv4 Datagrams Good Octets Register 23–183
Rx IPv4 Datagrams Header Errors Register 23–184
Rx IPv4 Datagrams Header Errors Register 23–185
Rx IPv4 Datagrams No Payload Frame Register 23–186
Rx IPv4 Datagrams No Payload Octets Register 23–187
Rx IPv4 UDP Disabled Frames Register 23–188
Rx IPv4 UDP Disabled Octets Register 23–189
Rx IPv6 Datagrams Good Frames Register 23–190
Rx IPv6 Good Octets Register 23–191
Rx IPv6 Datagrams Header Error Frames Register 23–192
Rx IPv6 Header Errors Register 23–193

Rx IPv6 Datagrams No Payload Frames Register 23–194
Rx IPv6 No Payload Octets Register 23–195
Rx Jab Error Register 23–196
Rx Length Error Register 23–197
Rx Multicast Frames (Good) Register 23–198
Rx Octet Count (Good) Register 23–199
Rx Octet Count (Good/Bad) Register 23–200
Rx Out Of Range Type Register 23–201
Rx Oversize (Good) Register 23–202
Rx Pause Frames Register 23–203
Rx Runt Error Register 23–204
Rx TCP Error Frames Register 23–205
Rx TCP Error Octets Register 23–206
Rx TCP Good Frames Register 23–207
Rx TCP Good Octets Register 23–208
Rx Unicast Frames (Good) Register 23–209
Rx UDP Error Frames Register 23–210
Rx UDP Error Octets Register 23–211
Rx UDP Good Frames Register 23–212
Rx UDP Good Octets Register 23–213
Rx Undersize (Good) Register 23–214
Rx VLAN Frames (Good/Bad) Register 23–215
Rx Watch Dog Error Register 23–216
SMI Address Register 23–217
SMI Data Register 23–219
Time Stamp Addend Register 23–220
Time Stamp Auxiliary TS Nano Seconds Register 23–221
Time Stamp Auxiliary TM Seconds Register 23–222
Time Stamp Control Register 23–223
Time Stamp High Second Register 23–227

Time Stamp Nanoseconds Register 23–228
Time Stamp Nanoseconds Update Register 23–229
Time Stamp Target Time Nanoseconds Register 23–230
PPS Control Register 23–231
Time Stamp PPS Interval Register 23–234
PPS Width Register 23–235
Time Stamp Low Seconds Register 23–236
Time Stamp Seconds Update Register 23–237
Time Stamp Status Register 23–238
Time Stamp Sub Second Increment Register 23–240
Time Stamp Target Time Seconds Register 23–241
Tx 1024- to Max-Byte Frames (Good/Bad) Register 23–242
Tx 128- to 255-Byte Frames (Good/Bad) Register 23–243
Tx 256- to 511-Byte Frames (Good/Bad) Register 23–244
Tx 512- to 1023-Byte Frames (Good/Bad) Register 23–245
Tx 64-Byte Frames (Good/Bad) Register 23–246
Tx 65- to 127-Byte Frames (Good/Bad) Register 23–247
Tx Broadcast Frames (Good) Register 23–248
Tx Broadcast Frames (Good/Bad) Register 23–249
Tx Carrier Error Register 23–250
Tx Deferred Register 23–251
Tx Excess Collision Register 23–252
Tx Excess Deferral Register 23–253
Tx Frame Count (Good) Register 23–254
Tx Frame Count (Good/Bad) Register 23–255
Tx Late Collision Register 23–256
Tx Multicast Frames (Good) Register 23–257
Tx Multicast Frames (Good/Bad) Register 23–258
Tx Multiple Collision (Good) Register 23–259
Tx Octet Count (Good) Register 23–260

Tx OCT Count (Good/Bad) Register 23–261
Tx Pause Frame Register 23–262
Tx Single Collision (Good) Register 23–263
Tx Unicast Frames (Good/Bad) Register 23–264
Tx Underflow Error Register 23–265
Tx VLAN Frames (Good) Register 23–266
VLAN Tag Register 23–267

Serial Peripheral Interface (SPI)

SPI Features 24–1
SPI Functional Description 24–2
ADSP-CM40x SPI Register List 24–2
ADSP-CM40x SPI Interrupt List 24–3
SPI Block Diagram 24–4
Transfer Protocol 24–5
Clock Considerations 24–6
Controlling Delay Between Frames 24–7
Flow Control 24–8
Slave Select Operation 24–9
Beginning and Ending a Non-DMA SPI Transfer 24–10
Transmit Operation in Non-DMA Mode 24–11
Receive Operation in Non-DMA Mode 24–11
DMA and Interrupt Multiplexing 24–11
Dual I/O Mode 24–11
Quad I/O Mode (SPI2 only) 24–12
Fast Mode 24–13
Memory-Mapped Mode (SPI2 only) 24–14
Memory-Mapped Description of Operation 24–15
Memory-Mapped Architectural Concepts 24–17
Memory-Mapped Read Accesses 24–21

Memory-Mapped High-Performance Features 24–25
Memory-Mapped Mode Error Status Bits 24–26
Memory-Mapped Programming Guidelines 24–27
Programming Example for Configuring SPI Memory Mapped Mode (Winbond W25Q32) 24–30
SPI Interrupt Signals 24–32
Data Interrupts 24–32
Status Interrupts 24–33
Error Conditions 24–33
SPI Programming Concepts 24–34
Master Operation in Non-DMA Modes 24–34
Slave Operation in Non-DMA Modes 24–35
Configuring DMA Master Mode 24–36
Configuring DMA Slave Mode Operation 24–37
ADSP-CM40x SPI Register Descriptions 24–38
Clock Rate Register 24–40
Control Register 24–41
Delay Register 24–47
Masked Interrupt Condition Register 24–48
Masked Interrupt Clear Register 24–50
Interrupt Mask Register 24–53
Interrupt Mask Clear Register 24–55
Interrupt Mask Set Register 24–58
Memory Mapped Read Header 24–61
SPI Memory Top Address 24–65
Receive FIFO Data Register 24–66
Received Word Count Register 24–67
Received Word Count Reload Register 24–68
Receive Control Register 24–69
Slave Select Register 24–72

Status Register 24–75
Transmit FIFO Data Register 24–80
Transmitted Word Count Register 24–81
Transmitted Word Count Reload Register 24–82
Transmit Control Register 24–83

Serial Port (SPORT)

Features 25–1
Signal Descriptions 25–3
Functional Description 25–6
ADSP-CM40x SPORT Register List 25–6
ADSP-CM40x SPORT Interrupt List 25–7
Block Diagram 25–8
Architectural Concepts 25–9
Multiplexer Logic 25–10
Data Types and Companding 25–13
Companding as a Function 25–13
Transmit Path 25–14
Receive Path 25–15
DMA and Interrupt Multiplexing 25–15
Sampling Edge 25–15
Premature Frame Sync Error Detection 25–16
Support for Edge-Detected and Level-Sensitive Frame Syncs 25–17
Serial Word Length 25–18
Operating Modes 25–19
Mode Selection 25–21
Standard Serial Mode 25–21
Frame Sync Options 25–22
Stereo Modes 25–24
I^2S Mode 25–25

Left-Justified Mode 25–26
Right-Justified Mode 25–27
Multichannel Mode 25–29
Packed I²S Mode 25–34
Gated Clock Mode 25–36
Data Transfers 25–37
Data Buffers 25–37
Data Buffer Status 25–38
Single Word (Core) Transfers 25–39
DMA Transfers 25–39
Error Detection 25–40
Interrupts 25–42
SPORT Programming Model 25–42
Setting Up and Starting Transfers 25–43
Setting Up and Starting Multichannel Transfers 25–44
Using DMA for SPORT Transfers 25–45
Using Companding as a Function 25–45
ADSP-CM40x SPORT Register Descriptions 25–46
Half SPORT 'A' Multi-channel 0-31 Select Register 25–48
Half SPORT 'B' Multi-channel 0-31 Select Register 25–49
Half SPORT 'A' Multi-channel 32-63 Select Register 25–50
Half SPORT 'B' Multi-channel 32-63 Select Register 25–51
Half SPORT 'A' Multi-channel 64-95 Select Register 25–52
Half SPORT 'B' Multichannel 64-95 Select Register 25–53
Half SPORT 'A' Multi-channel 96-127 Select Register 25–54
Half SPORT 'B' Multichannel 96-127 Select Register 25–55
Half SPORT 'A' Control 2 Register 25–56
Half SPORT 'B' Control 2 Register 25–57
Half SPORT 'A' Control Register 25–58

Half SPORT 'B' Control Register 25–66
Half SPORT 'A' Divisor Register 25–75
Half SPORT 'B' Divisor Register 25–76
Half SPORT 'A' Error Register 25–77
Half SPORT 'B' Error Register 25–79
Half SPORT 'A' Multi-channel Control Register 25–81
Half SPORT 'B' Multi-channel Control Register 25–83
Half SPORT 'A' Multi-channel Status Register 25–85
Half SPORT 'B' Multi-channel Status Register 25–86
Half SPORT 'A' Rx Buffer (Primary) Register 25–87
Half SPORT 'B' Rx Buffer (Primary) Register 25–88
Half SPORT 'A' Rx Buffer (Secondary) Register 25–89
Half SPORT 'B' Rx Buffer (Secondary) Register 25–90
Half SPORT 'A' Tx Buffer (Primary) Register 25–91
Half SPORT 'B' Tx Buffer (Primary) Register 25–92
Half SPORT 'A' Tx Buffer (Secondary) Register 25–93
Half SPORT 'B' Tx Buffer (Secondary) Register 25–94

Analog-to-Digital Converter Controller (ADCC)

ADCC Features 26–2
ADCC Functional Description 26–3
ADSP-CM40x ADCC Register List 26–8
ADSP-CM40x ADCC Interrupt List 26–10
ADSP-CM40x ADCC Trigger List 26–10
ADCC Signal Descriptions 26–11
ADCC Block Diagram 26–14
ADCC Architectural Concepts 26–15
Core and DMA Interfaces 26–16
Trigger Inputs 26–16
Timers 26–17

Event Register Banks 26–17
Event Comparators 26–18
Pending Event FIFO 26–19
Timing and Control Unit 26–19
ADCC Operating Modes 26–19
Data Transfer Modes 26–20
Core-Driven Data Read Mode 26–20
DMA-Driven Data Read Mode 26–20
DMA Bandwidth Monitoring 26–22
Dual-Bit (Two Signal Line) Interface Mode 26–22
Dual-Bit Interface Data Swap Mode 26–22
Clock Modes 26–24
Chip Select Modes 26–25
Simultaneous Sampling Mode 26–27
ADCC Event Control (SEC and TRU Related) 26–29
Interrupt Status 26–29
Error Status 26–30
Pending, Frame, and Delay Status 26–34
Event Handling Latency 26–35
ADCC Programming Concepts 26–36
ADCC Programming Model 26–37
ADC Voltage Reference (VREF) Selection 26–37
Program Flow 26–38
ADCC Programming Guidelines 26–38
ADSP-CM40x ADCC Register Descriptions 26–40
Base Pointer 0 Register 26–42
DMA Base Pointer 1 Register 26–43
Bandwidth Monitor 0 Register 26–44
Bandwidth Monitor 1 Register 26–45

Circular Buffer Size 0 Register **26–46**
Circular Buffer Size 1 Register **26–47**
ADC Configuration Register **26–48**
Control Register **26–49**
Event Collision Status Register **26–54**
Event Interrupt Mask Register **26–55**
Event Interrupt Mask Clear Register **26–56**
Event Interrupt Mask Set Register **26–57**
Event Interrupt Status Register **26–58**
Event Miss Status Register **26–59**
Pending Events Status Register **26–60**
Error Mask Register **26–61**
Error Mask Clear Register **26–63**
Error Mask Set Register **26–65**
Error Status Register **26–67**
Event n Control Register **26–69**
Event n Data Register **26–71**
Event n Status Register **26–72**
Event Enable Register **26–73**
Event Enable Clear Register **26–74**
Event Enable Set Register **26–75**
Event n Time Register **26–76**
Frame Interrupt Mask Register **26–77**
Frame Interrupt Mask Clear Register **26–78**
Frame Interrupt Mask Set Register **26–79**
Frame Interrupt Status Register **26–80**
Frame Increment 0 Register **26–81**
Frame Increment 1 Register **26–82**
Timer 0 Status Register **26–83**
Timer 1 Status Register **26–84**

Timing Control A (ADC0) Register 26–85
Timing Control A (ADC1) Register 26–86
Timing Control B (ADC0) Register 26–87
Timing Control B (ADC1) Register 26–88
Timer 0 Current Count Register 26–89
Timer 1 Current Count Register 26–90

Digital-to-Analog Converter Controller (DACC)

DACC Features 27–1
DACC Functional Description 27–3
DACC Signal Descriptions 27–3
DACC Architectural Concepts 27–5
Core and DMA Interfaces 27–6
Pending Data FIFO 27–6
DACC Operating Modes 27–6
Data Transfer Modes 27–7
Core-Driven Data Write Mode 27–7
DMA-Driven Data Write Mode 27–7
Data Length and Update Options 27–8
Clock Modes 27–9
Frame Sync Modes 27–10
Broadcast Control Option 27–10
DACC Event Control 27–11
Interrupt Status 27–11
Error Status 27–11
Pending Status 27–12
DACC Programming Model 27–13
DACC Programming Concepts 27–13
DACC Programming Guidelines (ADSP-CM40x Specific) 27–14

ADSP-CM40x DACC Register Descriptions 27–14
Broadcast (Write) Control Register 27–16
Base Pointer 0 Register 27–18
Base Pointer 1 Register 27–19
Count 0 Register 27–20
Count 1 Register 27–21
Current Count 0 Register 27–22
Current Count 1 Register 27–23
Control 0 Register 27–24
Control 1 Register 27–27
Data FIFO 0 Register 27–30
Data FIFO 1 Register 27–31
Error Mask Register 27–32
Error Mask Clear Register 27–34
Error Mask Set Register 27–35
Error Status Register 27–36
Interrupt Mask Register 27–38
Interrupt Mask Clear Register 27–40
Interrupt Mask Set Register 27–41
Interrupt Status Register 27–42
Modify 0 Register 27–44
Modify 1 Register 27–45
Status Register 27–46
Timing Control 0 Register 27–48
Timing Control 1 Register 27–49

Harmonic Analysis Engine (HAE)

HAE Features..... 28–1
HAE Functional Description 28–2
ADSP-CM40x HAE Register List..... 28–2

ADSP-CM40x HAE Interrupt List 28–3
HAE Block Diagram 28–3
HAE Architectural Concepts 28–4
Harmonic Engine 28–4
Harmonic Analyzer 28–5
Data Transfer Module 28–7
Results Memory 28–9
HAE Operating Modes 28–11
HAE Data Transfer Modes 28–11
HAE Event Control 28–11
HAE Interrupt Signals 28–11
HAE Status and Error Signals 28–11
HAE Programming Model 28–12
HAE Programming Concepts 28–13
Theory of Operation 28–13
Initialization 28–14
Harmonic Calculations 28–15
Configuring Harmonic Calculations Update Rate 28–16
ADSP-CM40x HAE Register Descriptions 28–17
Configuration 0 Register 28–18
Configuration 1 Register 28–19
Configuration 2 Register 28–20
Configuration 3 Register 28–22
Configuration 4 Register 28–23
DIDT Coefficient Register 28–24
DIDT Gain Register 28–25
Harmonic n Index Register 28–26
I (Current) Sample Register 28–27
I (Current) Waveform Register 28–28
Results RAM Register 28–29

Run Register 28–30
Status Register 28–31
Voltage Level Register 28–32
V (Voltage) Sample Register 28–33
V (Voltage) Waveform Register 28–34

Sinus Cardinalis (SINC) Filter

SINC Filter Features 29–1
SINC Functional Description 29–2
ADSP-CM40x SINC Register List 29–2
ADSP-CM40x SINC Interrupt List 29–3
ADSP-CM40x SINC Trigger List 29–3
SINC Definitions 29–4
SINC Block Diagram 29–4
SINC Architectural Concepts 29–6
Digital Filter 29–6
DC Gain and Data Resolution 29–7
Frequency Response 29–7
Output Scaling 29–8
SINC Operating Modes 29–9
SINC Data Transfer Modes 29–9
SINC Signal Modes 29–9
SINC Event Control 29–10
SINC Interrupt Signals 29–11
SINC Status and Error Signals 29–11
SINC Programming Model 29–12
SINC Programming Concepts 29–12
Channel Configuration 29–13
Trigger Masking 29–13
Interrupt Masking 29–13

Modulator Clock 29–14
Filter Configuration 29–14
Primary Filter Parameters 29–14
Primary DMA Configuration and Data Interrupts 29–15
Secondary Filter Parameters 29–15
Overload Detection 29–16
ADSP-CM40x SINC Register Descriptions 29–16
Bias for Group 0 Register 29–18
Bias for Group 1 Register 29–19
Clock Control Register 29–20
Control Register 29–22
History Status Register 29–25
Level Control for Group 0 Register 29–27
Level Control for Group 1 Register 29–29
(Amplitude) Limits for Secondary Filter 0 Register 29–31
(Amplitude) Limits for Secondary Filter 1 Register 29–32
(Amplitude) Limits for Secondary Filter 2 Register 29–33
(Amplitude) Limits for Secondary Filter 3 Register 29–34
Pair 0 Secondary (Filter) History n Register 29–35
Pair 1 Secondary (Filter) History n Register 29–36
Pair 2 Secondary (Filter) History n Register 29–37
Pair 3 Secondary (Filter) History n Register 29–38
Primary (Filters) Head for Group 0 Register 29–39
Primary (Filters) Head for Group 1 Register 29–40
Primary (Filters) Pointer for Group 0 Register 29–41
Primary (Filters) Pointer for Group 1 Register 29–42
Primary (Filters) Tail for Group 0 Register 29–43
Primary (Filters) Tail for Group 1 Register 29–44
Rate Control for Group 0 Register 29–45
Rate Control for Group 1 Register 29–47

Status Register 29–49

Reset Control Unit (RCU)

RCU Features 30–1
RCU Functional Description 30–1
ADSP-CM40x RCU Trigger List 30–2
ADSP-CM40x RCU Register List 30–2
RCU Definitions 30–3
RCU Architectural Concepts 30–4
RCU Status and Error Signals 30–4
Resetting a Core Through a System Master 30–4
ADSP-CM40x Specific Information 30–5
ADSP-CM40x RCU Register Descriptions 30–5
Boot Code Register 30–7
Core Reset outputs Control Register 30–10
Core Reset outputs Status Register 30–11
Control Register 30–12
Message Register 30–14
Message Clear Bits Register 30–16
Message Set Bits Register 30–17
Revision ID Register 30–18
System Reset Request Status 30–19
Status Register 30–20
Software Vector Register 0 30–22
SVECT Lock Register 30–23

Boot ROM and Booting the Processor

Boot Loader Stream 31–2
Block Types 31–5

Multi-Application Boot Streams 31–12
Single-Block Boot Streams 31–13
Direct Code Execution 31–14
CRC32 Protection 31–14
SRAM Requirements 31–15
Preboot 31–15
Boot Modes 31–23
No-Boot Mode 31–24
SPI Master Boot Mode 31–25
SPI Slave Boot Mode 31–34
UART Slave Boot Mode 31–37
Security 31–40
Security Features 31–40
Security Functional Description 31–41
Security Definitions 31–41
Enabling Security 31–41
Secure Header 31–42
Secure Debug 31–43
User Debug Key 31–43
Boot Mode Control 31–43
Mass Erase of the SoC Flash Memory 31–45
Security Programming Model 31–47
Boot ROM Programming Model 31–48
Callable Kernel API 31–48
adi_rom_Boot() 31–48
adi_rom_BootKernel() 31–51
adi_rom_MemCopy() 31–51
adi_rom_MemCompare() 31–52
adi_rom_MemCRC() 31–53
adi_rom_MemFill() 31–54

adi_rom_Crc32Init() 31–55
adi_rom_Crc32Poly() 31–56
adi_rom_GetId() 31–56
adi_rom_Ecc() 31–57
adi_rom_GetAddress() 31–58
Page Mode 31–59
Error Handler 31–60
Boot Kernel API 31–61
Kernel Hook Function 31–62
enum ROM_HOOK_CALL_CAUSE 31–63
Kernel Return Feature 31–63
Boot Termination and Application Execution 31–64
Booting Data Structures 31–64
struct ADI_ROM_BOOT_HEADER 31–64
struct ADI_ROM_BOOT_SECHDR 31–65
struct ADI_ROM_BOOT_SECHDR_FLAGS 31–67
struct ADI_ROM_BOOT_SECHDR_KEY 31–68
struct ADI_ROM_BOOT_SPI_FUNC 31–69
struct ADI_ROM_BOOT_SPI_ID 31–70
struct ADI_ROM_BOOT_UART 31–71
struct ADI_ROM_CONSTANTS_TYPE 31–71
struct U64_TypeDef 31–72
struct ADI_ROM_BOOT_BUFFER 31–73
struct ADI_ROM_BOOT_CONFIG 31–73
struct ADI_ROM_BOOT_LOCAL 31–80
struct ADI_ROM_BOOT_REGISTRY 31–81
struct ADI_ROM_BOOT_SPI 31–82
enum ROM_BMODE_RESULT 31–85
enum ROM_DMA_RESULT 31–85
enum ROM_KERNEL_RESULT 31–85
enum ROM_SECHDR_RESULT 31–86

enum ROM_GETADDR_VALUE 31–86
enum ROM_HOOK_CALL_CAUSE 31–87

System Watchpoint Unit (SWU)

SWU Features 32–1
SWU Functional Description 32–1
ADSP-CM40x SWU Register List 32–1
ADSP-CM40x SWU Interrupt List 32–2
ADSP-CM40x SWU Trigger List 32–2
SWU Definitions 32–3
SWU Architectural Concepts 32–3
SWU Flow Diagram 32–3
SWU-to-SCB Interface 32–4
SWU Block Diagram 32–4
System Crossbar Block 32–5
MMR Block 32–5
SWU Operating Modes 32–5
Bandwidth Mode 32–5
Watchpoint Mode 32–5
Match Block 32–5
SWU Event Control 32–6
SWU Interrupts 32–6
SWU Status and Errors 32–6
Triggers 32–6
SWU Programming Model 32–6
SWU Mode Configuration 32–7
Configuring the SWU for Bandwidth Mode 32–7
Configuring the SWU for Watchpoint Mode 32–8
ADSP-CM40x SWU Register Descriptions 32–8

Count Register n 32–10
Control Register n 32–11
Current Register n 32–16
Global Control Register 32–17
Global Status Register 32–18
Bandwidth History Register n 32–22
ID Register n 32–23
Lower Address Register n 32–24
Target Register n 32–25
Upper Address Register n 32–26

JTAG debug and Serial Wire Debug Port (SWJ-DP)

ADSP-CM40x TAPC Register Descriptions 33–1
IDCODE Register 33–3
Run Control Message Register 33–4
Run Control Message Clear Register 33–6
Run Control Message Set Register 33–7
Run Control Message Toggle Register 33–8
SDBGKEY0 Register 33–9
SDBGKEY1 Register 33–10
SDBGKEY2 Register 33–11
SDBGKEY3 Register 33–12
SDBGKEY Control Register 33–13
SDBGKEY Status Register 33–14
USERCODE Register 33–15
User Key Compare0 Register 33–16
User Key Compare1 Register 33–17
User Key Compare2 Register 33–18
User Key Compare3 Register 33–19

ADSP-CM40x Register List

1 Preface

Thank you for purchasing and developing systems using an ADSP-CM40x processor from Analog Devices, Inc.

Purpose of This Manual

The *ADSP-CM40x Processor Hardware Reference* provides architectural information about the ADSP-CM40x processors. This hardware reference provides the main architectural information about these processors. The architectural descriptions cover functional blocks, buses, and ports, including all features and processes that they support. For information about programming the ARM core in the ADSP-CM40x processor, visit the ARM Information Center at:

http://infocenter.arm.com/help/

For timing, electrical, and package specifications, see the *ADSP-CM40x Processor Data Sheet*.

Intended Audience

The primary audience for this manual is a programmer who is familiar with Analog Devices processors. The manual assumes the audience has a working knowledge of the appropriate processor architecture and instruction set. Programmers who are unfamiliar with Analog Devices processors can use this manual, but should supplement it with other texts, such as programming reference books and data sheets, that describe their target architecture.

What's New in This Manual

This is the third preliminary revision (0.3) of the *ADSP-CM40x Mixed Signal Control Processor with ARM Cortex-M4 Hardware Reference*. This revision adds the SPI Flash chapter, the Capture Timer chapter, and the MMR register appendix. It updates the Boot ROM programming model in the Boot ROM and Booting the Processor chapter. It also includes corrected errata associated with this processor.

Technical or Customer Support

You can reach customer and technical support for processors from Analog Devices in the following ways:

- Post your questions in the processors and DSP support community at ***EngineerZone***:

http://ez.analog.com/community/dsp

- Submit your questions to technical support at ***Connect with ADI Specialists***:

  http://www.analog.com/support

- E-mail your questions about software/hardware development tools to:

  processor.tools.support@analog.com

- E-mail your questions about processors and DSPs to:

  processor.support@analog.com (world wide support)

  processor.china@analog.com (China support)

- Phone questions to *1-800-ANALOGD* (USA only)

- Contact your Analog Devices sales office or authorized distributor. Locate one at:

  http://www.analog.com/adi-sales

- Send questions by mail to:

  Analog Devices, Inc.

  Three Technology Way

  P.O. Box 9106

  Norwood, MA 02062-9106 USA

Product Information

Product information can be obtained from the Analog Devices Web site and CrossCore Embedded Studio online Help system.

Analog Devices Web Site

The Analog Devices Web site, http://www.analog.com, provides information about a broad range of products—analog integrated circuits, amplifiers, converters, and digital signal processors.

To access a complete technical library for each processor family, go to: http://www.analog.com/processors/technical_library The manuals selection opens a list of current manuals related to the product as well as a link to the previous revisions of the manuals. When locating your manual title, note a possible errata check mark next to the title that leads to the current correction report against the manual.

Also note, MyAnalog.com is a free feature of the Analog Devices Web site that allows customization of a Web page to display only the latest information about products you are interested in. You can choose to receive weekly e-mail notifications containing updates to the Web pages that meet your interests, including documentation errata against all manuals. MyAnalog.com provides access to books, application notes, data sheets, code examples, and more.

Visit MyAnalog.com to sign up. If you are a registered user, just log on. Your user name is your e-mail address.

EngineerZone

EngineerZone is a technical support forum from Analog Devices. It allows you direct access to ADI technical support engineers. You can search FAQs and technical information to get quick answers to your embedded processing and DSP design questions.

Use EngineerZone to connect with other DSP developers who face similar design challenges. You can also use this open forum to share knowledge and collaborate with the ADI support team and your peers. Visit http://ez.analog.com to sign up.

Notation Conventions

Text conventions used in this manual are identified and described as follows. Additional conventions, which apply only to specific chapters, may appear throughout this document.

Example	Description
File > ***Close***	Titles in reference sections indicate the location of an item within the CrossCore Embedded Studio IDE's menu system (for example, the ***Close*** command appears on the ***File*** menu).
`{this | that}`	Alternative required items in syntax descriptions appear within curly brackets and separated by vertical bars; read the example as `this` or `that`. One or the other is required.
`[this | that]`	Optional items in syntax descriptions appear within brackets and separated by vertical bars; read the example as an optional `this` or `that`.
`[this, …]`	Optional item lists in syntax descriptions appear within brackets delimited by commas and terminated with an ellipse; read the example as an optional comma-separated list of `this`.
`.SECTION`	Commands, directives, keywords, and feature names are in text with `Letter Gothic` font.
`filename`	Non-keyword placeholders appear in text with italic style format.
NOTE:	***NOTE:*** For correct operation, ... A note provides supplementary information on a related topic. In the online version of this book, the word ***NOTE:*** appears instead of this symbol.
CAUTION:	***CAUTION:*** Incorrect device operation may result if ... ***CAUTION:*** Device damage may result if ... A caution identifies conditions or inappropriate usage of the product that could lead to undesirable results or product damage. In the online version of this book, the word ***CAUTION:*** appears instead of this symbol.

Example	Description
ATTENTION:	*ATTENTION:* Injury to device users may result if ... A warning identifies conditions or inappropriate usage of the product that could lead to conditions that are potentially hazardous for devices users. In the online version of this book, the word *ATTENTION:* appears instead of this symbol.

Register Documentation Conventions

Register diagrams use the following conventions:

- The descriptive name of the register appears at the top with the short form of the name.
- If a bit has a short name, the short name appears first in the bit description, followed by the long name.
- The reset value appears in binary in the individual bits and in hexadecimal to the left of the register.
- Bits marked *X* have an unknown reset value. Consequently, the reset value of registers that contain such bits is undefined or dependent on pin values at reset.
- Shaded bits are reserved

NOTE: To ensure upward compatibility with future implementations, write back the value that is read for reserved bits in a register, unless otherwise specified.

Register description tables use the following conventions:

- Each bit's or bit field's access type appears beneath the bit number in the table in the form (read-access/write-access). The access types include:
  - R = read, RC = read clear, RS = read set, R0 = read zero, R1 = read one, Rx = read undefined
  - W = write, NW = no write, W1C = write one to clear, W1S = write one to set, W0C = write zero to clear, W0S = write zero to set, WS = write to set, WC = write to clear, W1A = write one action
- Many bit and bit field descriptions include enumerations, identifying bit values and related functionality. Unless otherwise indicated (with a prefix), these enumerations are decimal values.

2 ARM Cortex-M4 Core Memory Sub-System

The ADSP-CM40x family of processors are based on the ARM® Cortex®-M4 processor core with floating-point unit and integrated SRAM memory, flash memory, accelerators, and peripherals. The processor provides suffi-cient memory to support micro-controller based applications. This memory includes 384K bytes of internal SRAM that can be partitioned in to blocks of code and data (64K bytes of configurable memory blocks). The processor also includes a static memory controller for interface to external devices or memories. The Cortex memory interface includes support for two SPI based flash memories (one within the processor and another external SPI flash) that can be memory-mapped for high performance code execution through SPI quad/dual I/O read modes. An included internal 16K byte code cache improves the execute-in-place functionality of flash memories significantly.

Note the following terms, which are used in this chapter:

- *Cortex core* refers to the ARM Cortex-M4 core with floating-point support and core peripherals.
- *Cortex memory* refers to the portions of the Cortex memory map that are part of the memory model for the Cortex core (for example, SRAM and Cache), but are not part of the system memory (memory mapped registers) or external memory.
- *ADSP-CM40x processor* or *processor* refers to the combination of the Cortex core, Cortex memory, system pe-ripherals (for example, UART, SPI, and SPORT), and system memories.

This document describes the ARM Cortex-M4 core and memory architecture used on the ADSP-CM40x processor, but does not provide detailed programming information for the ARM processor. For more information about programming the ARM processor, visit the ARM Information Center:

- http://infocenter.arm.com/help/

The applicable documentation for programming the ARM Cortex-M4 processor include:

- Cortex®-M4 Devices Generic User Guide
- CoreSight™ ETM™-M4 Technical Reference Manual
- Cortex®-M4 Technical Reference Manual

Cortex-M4 Memory Features

The ARM Cortex-M4 core memory architecture includes the following features:

- An internal memory sub-system supporting:
  - Up to 384K bytes of zero waitstate and configurable SRAM
  - 16K bytes of zero waitstate code cache
  - 32K bytes of boot ROM
- A high performance bus architecture involving:
  - Cortex core internal bus matrix
  - Memory bus matrix that connects to the Cortex core over the I-Code, D-Code, and SYS buses
- Cacheable external-memory interfaces, which support:
  - 32M byte x up to 4 banks of static memory control connected to asynchronous memories (SRAM, flash, FPGA)
  - 2M bytes of internal SPI flash (within the processor package)
  - Up to 16M bytes of external SPI flash

Cortex-M4 Memory Functional Description

The following sections provide the functional description for the Cortex-M4 core memory sub-system:

- ADSP-CM40x M4P Register List
- ADSP-CM40x M4P Interrupt List
- Cortex-M4 Memory Internal Buses Block Diagram
- Cortex-M4 Memory Map
- Cortex-M4 Memory
- Cortex-M4 Memory - Bit Banding
- Cortex-M4 Memory - Translation Memory Blocks (MEMX and MEMY)
- Cortex-M4 Memory - Synchronization Sequence

ADSP-CM40x M4P Register List

The ARM Cortex-M4 platform module (M4P) provides the interface to the L1 code cache and the main SRAM. A set of registers governs M4P operations. For more information on M4P functionality, see the M4P register descriptions.

Table 2-1: ADSP-CM40x M4P Register List

Name	Description
	Bus Fault Error Information Register

Table 2-1: ADSP-CM40x M4P Register List (Continued)

Name	Description
M4P_BUSFLT	
M4P_CACHE_CFG	Code Cache Configuration and Status Register
M4P_CACHE_CNTCTL	Cache Counter Control Register
M4P_CACHE_DFILL	Cache DCODE Line Fill Counter Register
M4P_CACHE_DMISS	Cache DCODE Miss Counter Register
M4P_CACHE_DREF	Cache DCODE Reference Counter Register
M4P_CACHE_IFILL	Cache ICODE Line Fill Counter Register
M4P_CACHE_IMISS	Cache ICODE Miss Counter Register
M4P_CACHE_IREF	Cache ICODE Reference Counter Register
M4P_CACHE_MEMX	MEMX Space Configuration Register
M4P_CACHE_MEMY	MEMY Space Configuration Register
M4P_CACHE_PEADDR	Code Cache Parity Error Address Register
M4P_SRAM_CFG	SRAM Configuration Register
M4P_SRAM_PEADDR_CORE	SRAM Parity Error Address (Core) Register
M4P_SRAM_PEADDR_DMA	SRAM Parity Error Address (DMA) Register
M4P_STCALIB	SysTick Calibration Register

ADSP-CM40x M4P Interrupt List

Table 2-2: ADSP-CM40x M4P Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
2	M4P0_L1CC_PERR	M4P0 L1 Cache Code Parity Error		
3	M4P0_CORE_SRAM_PERR	M4P0 SRAM Core Parity Error	Level	
4	M4P0_DMA_SRAM_PERR	M4P0 SRAM DMA Parity Error		
5	M4P0_BUS_FAULT	M4P0 Bus Fault	Edge	
6	M4P0_LOCKUP	M4P0 Lockup Error (Fault only; not an interrupt)	Level	
7	M4P0_SRAM_PERR_FLT	M4P0 SRAM Parity Error (Fault only; not an interrupt)		

Cortex-M4 Memory Internal Buses Block Diagram

The ARM Cortex-M4 core and memory architecture has two types of buses for instruction access: I-Code and D-Code. I-Code is for accessing instructions, and D-Code is typically meant for literals used in the instruction

assembly language. The SYS bus is used to access the data. These buses are further distributed to the internal memories (cache, SRAM, boot ROM) through a bus matrix. MEM buses (MEM_ICODE, MEM_DCODE, MEM_SYS) are the memory buses that connect to the user-configurable internal SRAM space. Depending on the configuration (partitioning between CODE and DATA), these buses are interfaced to code blocks (over the D-Code, I-Code buses) or to data blocks (through the SYS bus) The cache memory only has code buses involved, because it resides in the memory region and is meant for accelerating execution speed when executing instructions from flash memories. Accesses to memories in the SYSCLK domain must be through the system fabric interface, and these can incur additional latencies.

The ARM Cortex-M4 core and memory architecture does not have a strict memory space definition for code and data access, within each other's memory space. In other words, the Cortex core permits placing data in code regions and vice versa; this may not be optimal because it causes bus contention. Because the internal SRAM supports user partitions between code and data (up to 6 partitions), there is typically no need to inter-mix the placements. Similarly, code cache can be used to cache raw user data (such as arrays or buffers) through the D-Code interface to the cache memory.

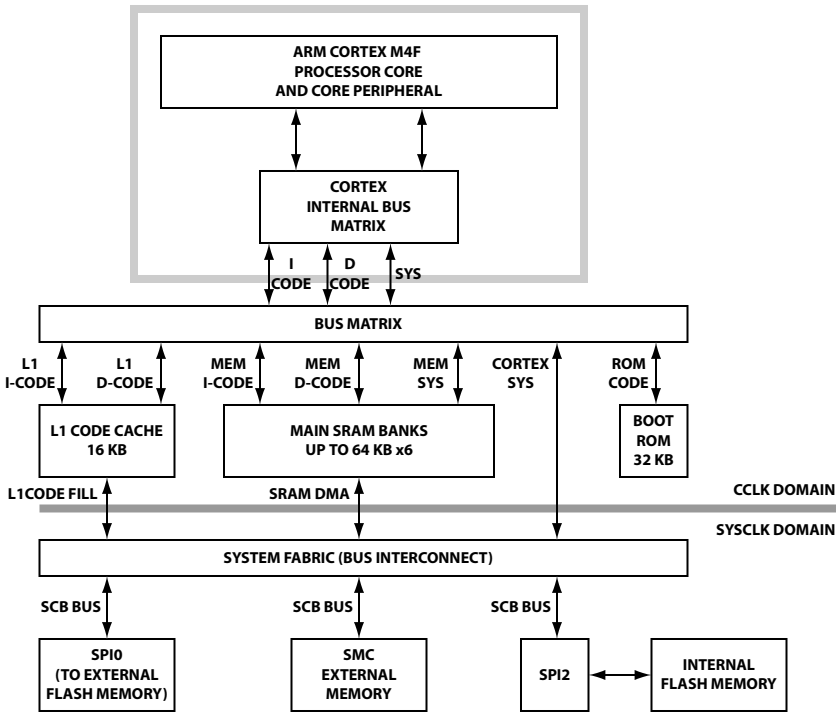

Figure 2-1: Internal Buses of the ARM Cortex-M4 Core and Memory Architecture

Cortex-M4 Memory Map

The processor uses the standard memory map/model, which is documented for the ARM Cortex-M4 core. By retaining the standardized memory mapping, it becomes easier to port applications across ARM Cor-tex-M4 based products. Only the physical implementation of memories inside the processor memory model differs from other vendors. The processor has a fixed default memory map that provides up to 4GB of addressable memory. For the processor memory map, see the product data sheet.

Cortex-M4 Memory

The ARM Cortex-M4 core memory sub-system of the ADSP-CM40x processor provides a number of memory types:

- *Code RAM and Data RAM*

  The processor has in-built SRAM support that can be configured as blocks of 64K bytes between code and data memory spaces. Up to 6 such blocks are available for configuration. The total space is always limited by the maximum size of SRAM (see data sheet). The code space typically contains instructions and literal (constant) data. It supports read/write access by the Cortex core and read/write DMA access by system devices. Data SRAM space can contain read/write data and can also be accessed by the Cortex core and DMA.

- *Boot ROM*

  The processor has several mechanisms for automatically loading internal and external memory after a reset. A 32K byte boot ROM is executed at system reset in order to perform all boot-related functionality. This space supports read-only access by the Cortex core only. Refer to the *Boot ROM and Booting the Processor* chapter for more information on how the ROM operates and how booting is performed.

- *Internal Quad SPI Flash (within package)*

  The processor contains a 2M byte flash memory space that is memory mapped for the Cortex core to access directly. This space can contain instructions and literal (constant) data. The space supports read-only access by the Cortex core. Write accesses to this space are ignored. Programming of the flash device is performed using the register access interface on the SPI2 peripheral. Code access from SPI2 flash is cacheable through MEMX space.

- *External SPI Flash*

  The processor optionally may be connected to up to 16M bytes of external flash memory through the SPI0 peripheral, which is memory mapped for the Cortex core to access directly. The external flash is similar to the integrated SPI flash code space: Read-only, cacheable, and programmed using the SPI0 register interface. Code access from SPI0 flash is cacheable through MEMY space.

- *External Asynchronous Memory Banks*

  The static memory controller (SMC) may be programmed to control up to four banks of external memories or memory-mapped devices, with very flexible timing parameters. Each bank occupies a 32M byte segment regardless of the size of the device used, so that these banks are only contiguous if each is fully populated with 32M bytes of memory. Typical use of SMC memory banks is to use an external SRAM for extending the memory availability in the system. Refer to the Static Memory Controller chapter for more information. Code access from SMC is cacheable through MEMY space.

Cortex-M4 Memory Map - Code and Data Regions

The Cortex memory code region has the following features:

- Performs accesses on I-Code (read) and D-Code (read/write) interfaces

- Contains the following:
  - 4K byte boot ROM
  - SRAM ConfigBanks, as configured
  - Internal Flash

The SRAM region has the following features:

- The Cortex core performs accesses on the SYS (read/write) interface
- In the ADSP-CM40x Cortex-M4 Platform, this region contains
  - SRAM ConfigBanks, as configured

The Cortex memory provides a bit-banding alias region for the address region containing the main SRAM, mapping each 32-bit word in SRAM to 32 1-bit words in the alias region.

Cortex-M4 Memory Accessibility - Cortex Core Perspective

Table 2-3: Memory Accessibility – Cortex Core Perspective

Address Range	Range Size	Type, Attributes, Access	Description
CODE region (I-CODE, D-CODE buses)			
0x0000_0000 – 0x0000_7FFF	32KB	Normal, R/O	Boot ROM
0x1000_0000 – 0x1005_FFFF	Up to 384KB	Normal, Non-shareable, R/W	Main SRAM, code partition
0x1800_0000 – 0x181F_FFFF	2 MB	Normal, Non-shareable, Cacheable, R/O	Cacheable region (MEMX)
0x1900_0000 – 0x19FF_FFFF	4 MB	Normal, Non-shareable, Cacheable, R/O	Cacheable region (MEMY)
SRAM region and above (SYS bus)			
0x2000_0000 – 0x2005_FFFF	Up to 384KB	Normal, Shareable, Non-cacheable, R/W	Main SRAM, data partition
0x4000_0000 – 0x401F_FFFF		Execute-never (XN), Non-cacheable	System Space. (interfaced to CORTEX_SYS bus)
0x5000_0000 – 0x50FF_FFFF		Normal, Non-shareable, Non-Cacheable, R/O	System Space. (interfaced to CORTEX_SYS bus)
0x6000_0000 – 0x6DFF_FFFF	128 MB	Normal, Non-shareable, R/W	System Space. Forwarded to external system (on CORTEX_SYS port).
0xE000_0000- 0xE00F_FFFF		Execute-never (XN), Non-cacheable	ARM Private Peripheral Bus (PPB) space
0xF000_0000– 0xF000_FFFF		Execute-never (XN), Non-cacheable	Cortex-M4 processor and memory architecture control register space

Table 2-3: Memory Accessibility – Cortex Core Perspective (Continued)

Address Range	Range Size	Type, Attributes, Access	Description
0xF800_0000– 0xFFFF_FFFF		Execute-never (XN), Non-cacheable	System registers, forwarded to external system (on CORTEX_SYS port)

Cortex-M4 Memory Accessibility - User/Application Perspective (Read Access)

In the ***Memory accessibility – User / Application Perspective (Read access)*** table, note that:

- *Yes* – indicates access is allowed.
- *No* – indicates access is not allowed.

Table 2-4: Memory accessibility – User / Application Perspective (Read Access)

Access Path/Type		Slave Memory Space				
Path	**Read Access**	**Internal SRAM (Code, Data)**	**Peripheral MMRs**	**SPI0 Interfaced External Flash**	**SPI2 interfaced Internal Flash**	**SMC Interfaced External Memory**
Core	Core read (instruction fetch)	Yes	No - ('Device' space has Execute-Never attribute, per ARM)	Yes - (only in Memory-Mapped Mode)	Yes - (only in Memory-Mapped Mode)	Yes
	Core read (memory load)	Yes	Yes	Yes - (only in Memory-Mapped Mode)	Yes - (only in Memory-Mapped Mode)	Yes
	Core read from Device using peripheral MMRs	Not applicable	Not applicable	Yes – (Not applicable in Memory-Mapped Mode)	Yes – (Not applicable in Memory-Mapped Mode)	Not applicable
PDMA	Device read using peripheral's own PDMA channel	Not applicable	Not applicable	Yes – (SPI0 can do PDMA read from flash (to SRAM or SMC)) (Not applicable in Memory-Mapped Mode)	No – SPI2 PDMA not provided	Not applicable
	Slave memory read by another peripheral's DMA	Yes	No	No – (other peripherals can't DMA from SPI0 slave memory)	No– (other peripherals can't DMA from SPI2 slave memory)	Yes
MDMA	MDMA read from slave memory space	Yes	Yes - (can MDMA from peripheral	Yes – (only in Memory-Mapped Mode)	Yes – (only in Memory-Mapped Mode)	Yes

Table 2-4: Memory accessibility – User / Application Perspective (Read Access) (Continued)

Access Path/Type		Slave Memory Space				
Path	**Read Access**	**Internal SRAM (Code, Data)**	**Peripheral MMRs**	**SPI0 Interfaced External Flash**	**SPI2 interfaced Internal Flash**	**SMC Interfaced External Memory**
			MMRs of only-CANx, GPIO, TRU, SEC, USB)			

Cortex-M4 Memory Accessibility - User/Application Perspective (Write Access)

In the ***Memory Accessibility – User / Application Perspective (Write Access)*** table, note that:

- *Yes* – indicates access is allowed.
- *No* – indicates access is not allowed.

Table 2-5: Memory accessibility – User / Application Perspective (Write Access)

Access Path/Type		Slave Memory Space				
Path	**Write Access**	**Internal SRAM (Code, Data)**	**Peripheral MMRs**	**SPI0 Interfaced External Flash**	**SPI2 interfaced Internal Flash**	**SMC Interfaced External memory**
Core	Core write (memory store)	Yes	Yes	No - (Cache is read-only)	No - (Cache is read-only)	Yes
	Core write to device using peripheral MMRs	Not applicable	Not applicable	Yes – (Not applicable in Memory-Mapped Mode)	Yes – (Not applicable in Memory-Mapped Mode)	Not applicable
PDMA	Device write - using peripheral's own PDMA channel	Not applicable	Not applicable	Yes – (SPI0 can do PDMA write to flash (from SRAM or SMC)) (Not applicable in Memory-Mapped Mode)	No – (SPI2 PDMA not provided)	Not applicable
	Slave memory write - by another peripheral's DMA	Yes	No	No – (other peripherals can't DMA to SPI0 slave memory)	No – (other peripherals can't DMA to SPI2 slave memory)	Yes
MDMA	MDMA write to slave memory space	Yes	Yes - (can MDMA to peripheral	No – (SPI0 slave memory port does not support writes)	No – (SPI2 slave memory port does not support writes)	Yes

Table 2-5: Memory accessibility – User / Application Perspective (Write Access) (Continued)

Access Path/Type		Slave Memory Space				
Path	**Write Access**	**Internal SRAM (Code, Data)**	**Peripheral MMRs**	**SPI0 Interfaced External Flash**	**SPI2 interfaced Internal Flash**	**SMC Interfaced External memory**
			MMRs of only-CANx, GPIO, TRU, SEC, USB)			

Cortex-M4 Memory - Bit Banding

In the Cortex memory, a bit-band region maps each word in a bit-band alias region to a single bit in the bit-band region as shown in the figure. The bit-band regions occupy the lowest 1M byte of the SRAM and peripheral memory regions. This feature supports the following:

- Bit banding allows physical representation of a stream of bit data.
- Bit banding provides atomic read-modify-write operations to bit data (locked access).
- Bit banding usage typically includes user can access of a bit in an memory mapped register address OR user toggling a flag in a variable, both in a locked manner.
- Bit banding works with a combination of region and alias, where alias word access is mapped to bit access in region.
- A word access to the SRAM or peripheral bit-band alias regions maps to a single bit in the SRAM or peripheral bit-band region
- Bit band accesses can use byte, half-word, or word transfers. The bit band transfer size matches the transfer size of the instruction making the bit band access.
- Typically, users must write their own macros to access the bit banding regions.

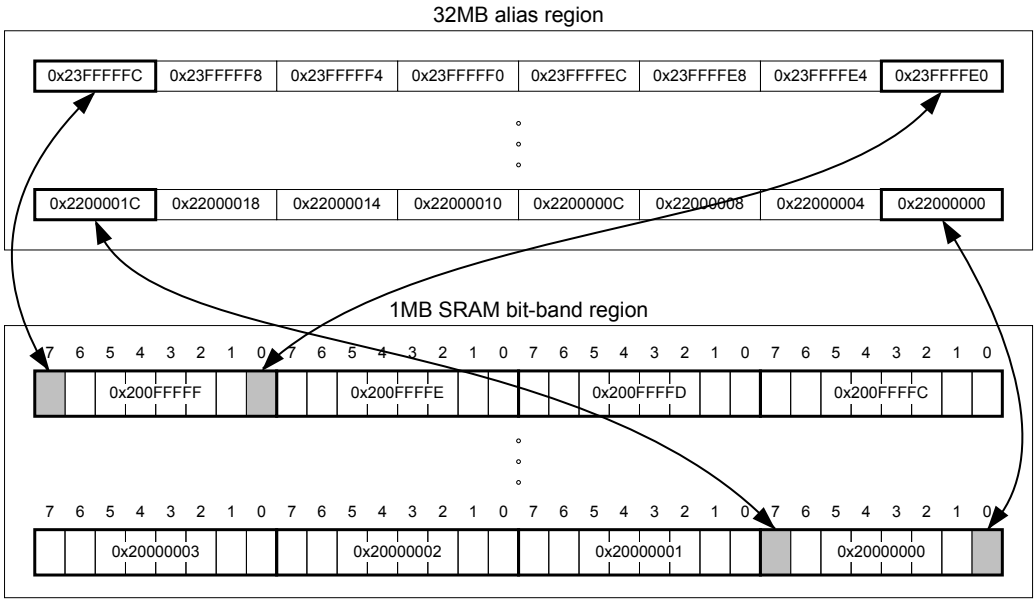

Figure 2-2: Bit Band Mapping

Cortex-M4 Memory - Translation Memory Blocks (MEMX and MEMY)

The cache memory resides in the code region of Cortex-M4 memory. The SPI2 interfaced flash available in the package is also mapped to the code region of Cortex-M4 memory. To improve performance, the memory accessed through SPI0 and the SMC must also be cached, but these blocks reside outside the code region. This architecture makes it preferable to have a direct address translation inside the processor package. This translation makes sure that all accesses are initiated through D-Code / I-Code, achieving efficient throughput. Otherwise, the instruc-tion fetches would contend with data accesses on the same SYS (data) bus. These translation memory blocks are called memory X (MEMX) and memory Y (MEMY). Noted that the cache controller can only access these memory blocks while making a memory mapped access from SPI0, SPI2, or SMC interfaced memory.

The translator is assigned for a specific memory space (SPI0 or SPI2 or SMC SRAM) by programming the MEMX and MEMY registers. The ***System (Physical), Application (Virtual), and Cache Memory*** figure illustrates the flow of a cacheable access as performed by the application. The user only has to ensure that MEMX and MEMY registers are appropriately programmed and has to build/compile the application code (using the cache) against these regions.

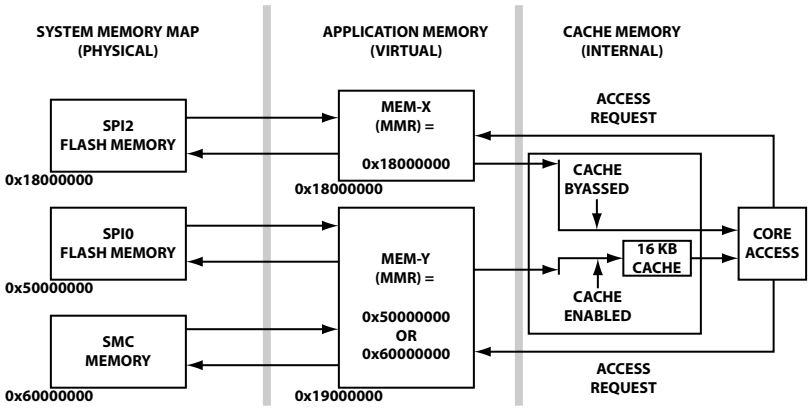

Figure 2-3: System (Physical), Application (Virtual), and Cache Memory

As shown, data is read from system memory blocks through MEMX or MEMY. When using DMA, the access must be directly from SPI system memory space. It is possible to directly access the SMC connected to asynchronous memory without using the MEMX/MEMY locations. Although direct core accesses are possible, you must still access the memory through MEMX/MEMY if caching is preferred for better performance.

The control access permission field in the MEMX and MEMY registers set access permissions for accesses by the core that bypass cache and attempt to access the physical device directly. (Writes might lead to cache incoherence.)

Note that:

- The physical and logical space of SPI2 internal flash is the same.
- The ***System (Physical), Application (Virtual), and Cache Memory*** figure only depicts the most common use case, involving the internal flash as a device always in use. The MEMX region is typically always assigned to SPI2 internal flash. The processor allows the application to assign any memory block (SPI0 flash, SPI2 flash, or SMC SRAM) to either MEMX or MEMY.

Cortex-M4 Memory - Synchronization Sequence

The processor includes multiple bus interfaces, which operate independently. Fetch, load, and store operations on any of these interfaces may cause side effects to be visible on the other interfaces (for example, reading I-Code after writing D-Code) or visible by the processor instruction sequence (for example, interrupts). When the precise order of these operations is important, synchronization barriers must be used.

For synchronization between accesses within Main SRAM (I-Code versus D-Code), a barrier instruction by itself is sufficient: ISB, DSB, or DMB as appropriate. For details, see the ARMv7-M Architecture Reference Manual.

To synchronize an access to D-Code or System space with context-altering side effects, so that the side effects are ensured to be visible by a subsequent instruction, a Cortex core system event synchronization sequence must be used, as follows:

```
STR xxx; /* instruction with side effects */
SEV; /* Send Event instruction */
xSB; /* ISB (Instruction Sync Barrier) or DSB (Data Sync Barrier)*/
LDR xxx; /* side effects visible */
```

Cortex-M4 Cache

The ARM Cortex-M4 processor core and memory architecture of the ADSP-CM40x processor includes a cache controller, accelerating execution-in-place (XiP) from SPI flash. The cache controller features a 16K byte zero-wait-state cache to store recently used code, without any manual overlay management.

The cache controller and cache memory are intended for program storage (I-Code for instructions and D-Code for literals). However, the cache also may be employed to store raw user data (such as data buffers). The cache does not have a write port, so it works only when reading from a cacheable space.

The cache can store code/data from either MEMX or MEMY regions, and these are the actual cacheable regions. These virtual translation blocks in turn read from SPI0, SPI2, or SMC async banks, based on how the MEMX/MEMY registers are configured.

Cache Controller Features

The cache controller features include:

- Up to 16K bytes of internal cache space, that can be used to store code or data.
- Support for split or shared cache between I-Code and D-Code space.
- Efficient organization through 8 x 2 KB SRAM banks.
- Way configurations of 4 Ways, 2 Ways, 1 Way
- Line configurations of 32-bits, 64-bits, 128-bits and 256-bits.
- Parity error detection and management.
- Option to bypass cache completely or partially.
- Performance features to improve throughput:
    - Linear access
    - Wrap-based critical word access
    - Merging of continuous linear accesses.
    - Preemptive access

Cache Structural Organization

The cache memory is composed of 8 x 2K byte SRAM banks. The cache can be split between I-Code and D-Code access, although shared cache is preferred in most applications.

The ***Cache Way Configurations*** table lists the supported way configurations.

Table 2-6: Cache Way Configurations

M4P_CACHE_CFG.CORG Bit Field Organization	Cache Segregation	D-CODE Cache Size, Ways	I-CODE Cache Size, Ways
000	Shared	16kB, 4 way 3	
001	Shared	16kB, 2 way 2	
010	Shared	16kB, 1 way 1	
011	Reserved		
100	Independent	4 kB, 1 Way	12 kB, 3 Way
101	Independent	8 kB, 2 Way	8 kB, 2 Way
110	Independent	12 kB, 3 Way	4 kB, 1 Way
111	Reserved		

The ***Cache Line Configurations*** table lists the support line configurations across all way configurations:

Table 2-7: Cache Line Configurations

Line Configuration
32-bit
64-bit
128-bit
256-bit

A *cache line* is composed of one or more 32-bit data words, as selected by the CACHE_CFG line size field (`M4P_CACHE_CFG.ILINE` or `M4P_CACHE_CFG.DLINE`). Line sizes from 4 to 32 bytes are supported. Each word is stored along with an address tag, so processor core accesses can be matched to cache contents (hit or miss).

A *cache set* refers to a group of memory words which share the same low-order part of the address, but may reside in different high-order address locations. The low order address part is called the *set address*, and the high order is the *tag address*. The cache can only hold a limited number of members of the same set. This is the number of ways. A round robin scheme is used for cache line replacement among the ways in a cache set.

Each Way is composed of multiple 8K byte SRAM memory banks and the banks are accessed alternately for efficient access of odd and even addresses. Inside each bank, there are a total of 512 rows of 32-bit words.

The cache is enabled by default and the default settings in the cache configuration register are typically optimal to run most applications. Only special cases, such as wrap mode or command skip modes, must be configured in conjunction with similar settings in the SPI controller and flash memory. In the default use case, even the MEMX is assigned to internal flash. This means the user need not worry about setting up the cache controller or MEMX.

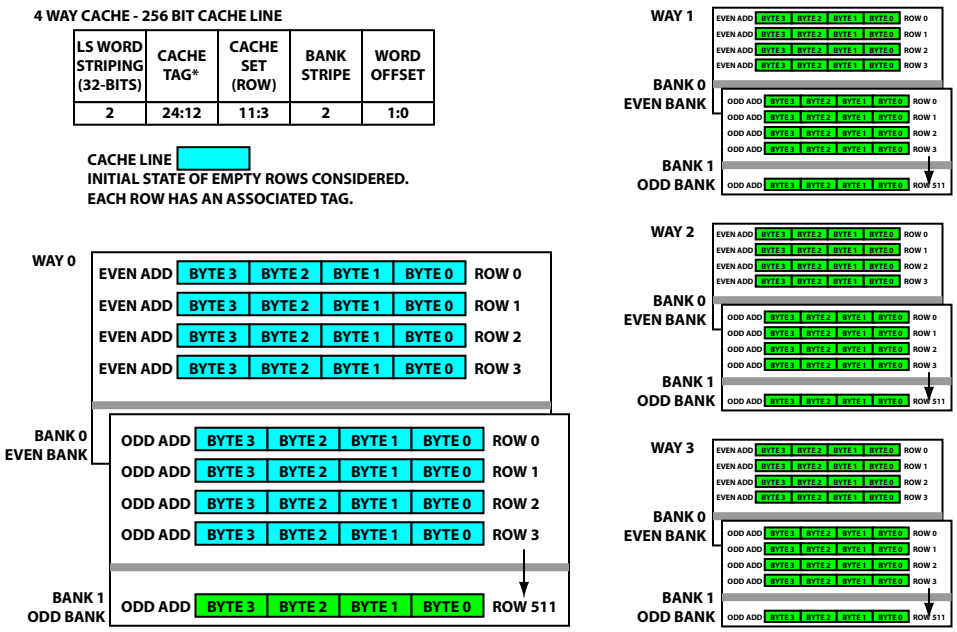

Figure 2-4: Cache Ways and Lines in Default Configurations

Clearing the Cache

The cache and its associated state machine (TAG, valid, parity, and others) are cleared by hardware in any of the following situations:

- De-assertion of reset
- Writing a 1 to the cache clear request bit (`M4P_CACHE_CFG.CLEAR`)
- A change to cache configuration selected by the cache organization bits (`M4P_CACHE_CFG.CORG`)

CAUTION: Hardware clearing of cache after the de-assertion of reset is required to avoid false parity errors.

Bypassing the Cache

Cache access can be bypassed by setting the configuration as partial bypass (cache miss bypass) or full bypass (both cache miss and cache hit bypass). The application may bypass the cache any time during execution, thus preserving the contents of cache when the banks are full (a cache miss can update cache memory), but at the cost of additional cycles for execution. Bypassing is available in both CMODE (cache mode without parity error) and PEMODE (cache mode with parity error) operation.

Using the Cache Counters

Counters are provided for application diagnostic purposes:

- Reference counters (`M4P_CACHE_IREF` and `M4P_CACHE_DREF`) count references (read accesses) through ICODE and DCODE

- Miss Counts (M4P_CACHE_IMISS and M4P_CACHE_DMISS) count references that were misses (so hits = references - misses)
- Line Fill Counters (M4P_CACHE_IFILL and M4P_CACHE_DFILL) count line fills first triggered by each ICODE, DCODE interface
- Each of the six counter registers has 24 bits (max range 16M counts) plus a sticky overflow bit.

The counter enable bit (M4P_CACHE_CNTCTL.ENCNT) enables counting and if this bit is zero, counters stop counting but they still hold their value at that instance. The user can also specify whether to count only committed accesses, or speculative accesses. User can zero all the counters at one shot by setting the M4P_CACHE_CNTCTL.CNTZERO bit. When used in combination with the M4P_CACHE_CNTCTL.SAMPLE bit, it saves the counts to the shadow registers and then clears the live counts, so no event gets dropped or double counted.

When the M4P_CACHE_CNTCTL.SAMPLE bit is set to 1, all the counters are sampled at once to a shadow set of registers for later inspection, while the real counters continue to count. The M4P_CACHE_CNTCTL.SAMPLE bit must be written back to 0 to see the live, up-to-date counters. This can be used to take a snapshot of the counters at an instant in time, so that combinations of register values (for example, references-misses) are legal and meaningful. Otherwise, if you sampled the live counters and read the number of references followed by the number of misses, there might be some new misses that came in between the two reads which would not be in the references count, so the number of hits would be calculated incorrectly.

Using Cache Parity Control

Cache line data and attributes are parity protected (this encompasses all internal SRAM usage by cache). Parity errors may be ignored, until they are associated with a cache set being returned to the processor. Any parity errors encountered during normal cache fetching (Code/Data FETCH asserted) trigger data reacquisition from backing memory, but only if data is immediately required by the processor. Data is not be automatically re-acquired if a parity error is associated with a way which does not tag match.

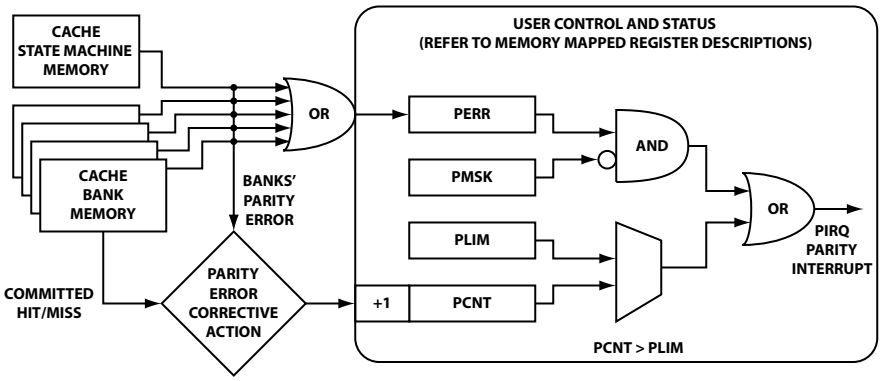

Figure 2-5: Parity Error Interrupt Generation

The ***Parity Error Interrupt Generation*** figure shows the cache parity control logic. Whenever a parity error interrupt is requested, status bit PIRQ will be asserted. Writing a "1" to this status bit clears the interrupt request. Status bit PERR (Parity Error) will be asserted if any cache Way of a cache Set associated with a fetch return to the processor

has a parity error. Whenever PERR is asserted, PIRQ bit will also be asserted if PMSK bit (Parity Error Mask) isn't asserted. PERR is cleared whenever PIRQ is cleared.

On a rising edge of PIRQ, the location of the triggering parity error will be copied into the Parity Error Address register. The cache state machine memory also provides parity error indication and contributes to parity error (PERR). This internal memory does not provide corrective mechanism as are found in the cache memory banks. Parity errors of the Cache State Machine memory do *not* increment PCNT.

The `M4P_CACHE_CFG.PLIM` bit field represents a user specified limit for PCNT[2:0]. PCNT[2:0] is incremented each time a word is fetched from backing memory due to a cache miss with an associated parity error and is cleared whenever PIRQ is cleared (whenever a 1 is written to bit PIRQ). Because fills preferentially overwrite cache ways which have parity errors, PCNT[2:0] does not over count the number of words in cache which were corrupted and resulted in hardware correction.

The a user may prefer to allow hardware to service a number of parity errors without the processor being notified. Multiple parity errors in multiple ways cause a single hardware correction fetch. PCNT only increments by 1.

On PIRQ being asserted, cache mode immediately switches from that specified by the `M4P_CACHE_CFG.CMODE` bits to that specified by the `M4P_CACHE_CFG.PEMODE` bits. Switching cache mode on parity error allows cache forwarding and updating behavior to be immediately modified according to user specifications.

- If in cache mode 00 (`M4P_CACHE_CFG.PEMODE`), cache is fully operational:
  - Cache contents are returned to the processor upon fetches with cache hits (TAG matches)
  - Cache contents are updated from backing memory upon fetches with cache misses
- If in cache mode 01 (`M4P_CACHE_CFG.PEMODE`), cache forwards but does not update:
  - Cache contents are returned to the processor upon fetches with cache hits (TAG matches)
  - Cache contents are not updated from backing memory upon fetches with cache misses (return from backing memory provided only to processor)
- In cache mode 11 (`M4P_CACHE_CFG.PEMODE`), cache forwarding and updating are both disabled (or known as "Fully bypassed")
  - This allows direct access of backing memory rather than forwarding from cache, and preserves state of cache

Cortex-M4 Code and Data SRAM

The unified internal SRAM space provides both code and data memory for the ARM Cortex-M4 processor core, allowing a configurable partition between code and data space. In addition, the SRAM features a DMA access port for read/write access by other master devices on the system fabric. The SRAM and all its interfaces operate in the ARM Cortex-M4 core clock domain (CCLK). The SRAM supports exclusive accesses. The SRAM can be accessed at the maximum CCLK speed in zero-wait-state. For access timing information, see the product data sheet.

SRAM Features

The SRAM has the following features:

- Up to 160K byte SRAM Capacity
- Zero wait-state performance at maximum CCLK speed
- Dynamically configurable between code space and data SRAM space partitions
- ECC protection (SEC-DED)
- Exclusive access support
- Two 32-bit buses for ARM Cortex-M4 core access to code space (MEM_ICODE, MEM_DCODE)
- One 32-bit bus for ARM Cortex-M4 core access to SRAM space (MEM_SYS)
- One 32-bit bus for system DMA access to code and SRAM spaces (SRAM_DMA)
    - Supports one DMA access, read or write, per core clock at maximum CCLK speed
- Defined maximum DMA response latency due to collisions with core activity (8 cycles maximum)

SRAM Block Diagram

The ***ARM Cortex-M4 SRAM Block Diagram*** figure shows the structure of the SRAM and associated buses.

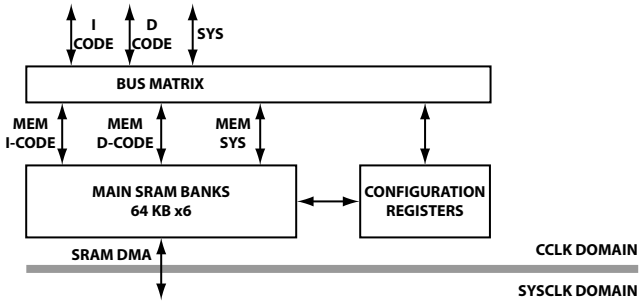

Figure 2-6: ARM Cortex-M4 SRAM Block Diagram

SRAM Bank Organization

The SRAM resources are divided into banks for efficiency (as shown in the ***SRAM Address Fields And Mapping*** figure), reducing the conflicts between accesses from various sources. These sources include: core fetch versus core load/store versus DMA. Usually, a user does not need to worry about the internal SRAM organization, other than the conflict management case:

- MSB Striping (banking) divides memory by 32K byte regions into separate ConfigBanks.

  Accesses to different ConfigBanks never cause conflict.
- LSB striping divides memory into four 32-bit lanes, so that accesses in different lanes do not conflict.

Effect is that randomly distributed memory accesses within the same 64K byte ConfigBanks only cause stalls 25% of the time.

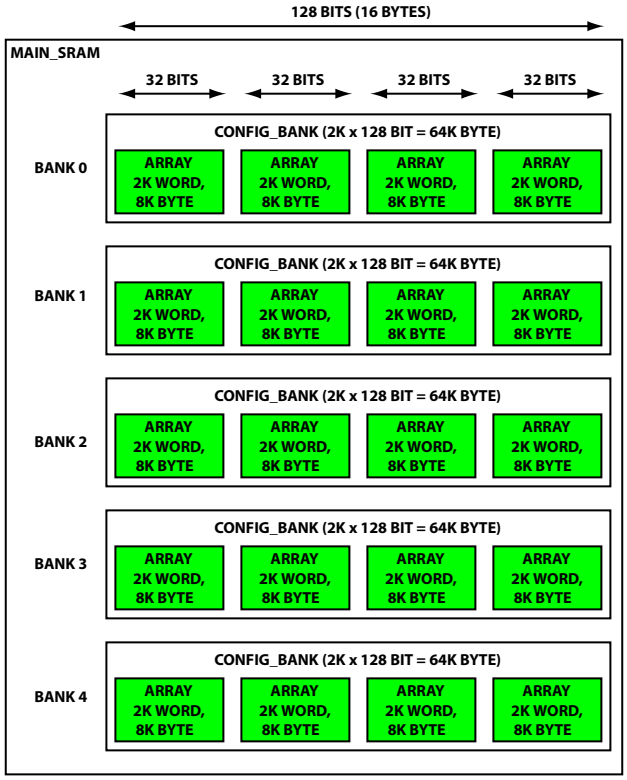

Figure 2-7: SRAM Address Fields And Mapping

Table 2-8: SRAM Address Fields And Mapping

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB (CfgBank)				Word Address (12 bit, 4K)												LSB (Array-Bank)		W (x32) (Byte)	

SRAM Partitioning using ConfigBanks

The address range of main SRAM depends on the configuration settings in the `M4P_SRAM_CFG.CDBANKS` register field as shown in the see ***Memory Map Configuration, Code and SRAM Regions*** figure. The figure shows an implimentation using five config banks and available memory is denoted by the shaded regions.

Table 2-9: SRAM Bank Configuration Base Addresses

Config	31	30	29	28	27	26	25	24	23	22	21	20	Description
CODE	0	0	0	1	0	0	0	0	0	0	0	0	CODE at 0x1000_0000
DATA	0	0	1	0	0	0	0	0	0	0	0	0	DATA at 0x2000_0000

The memory resources in the ARM Cortex-M4 memory main SRAM are divided into two or more ConfigBanks. Each ConfigBank may appear in either the code or the SRAM region, but may not appear in both. The settings in the supervisor-only `M4P_SRAM_CFG.CDBANKS` register field specify how many contiguous ConfigBanks appear in the CODE segment, starting at the lowest address (0x1000_0000.) The remaining ConfigBanks appear in a continuous address range in the DATA segment, starting at the highest populated address (0x2000_0000 + SRAM_SIZE 1).

When the value of the `M4P_SRAM_CFG.CDBANKS` register is changed, resulting in changes to the populated regions in the memory map, the contents of newly-accessible memory ranges are UNSPECIFIED. The user should not assume that the contents of any specific memory range are transferable between the CODE region to the SRAM region when the `M4P_SRAM_CFG.CDBANKS` register is changed.

It is important to manage the `M4P_SRAM_CFG.CDBANKS` register when booting or initializing an application, so that the intended memory map is configured before the application is copied from the boot source into the active locations in the CODE or SRAM regions. For example, the program may choose a linker control file which specifies 96 KB of CODE (0x1000_0000 to 0x1001_7FFF) and 64 KB of DATA SRAM (0x2001_8000 to 0x2002_7FFF). The linker may automatically support copying read-only sections of the application from nonvolatile (Flash) memory regions to the SRAM regions at start-up time in an init function. It is important that the `M4P_SRAM_CFG.CDBANKS` register is initialized to the intended value (here, 3) before init is called. This task can be performed in the Reset handler.

Programs should not attempt an active access to a memory bank when changing the config for that bank, including instruction fetches. A program that changes the `M4P_SRAM_CFG.CDBANKS` register must reside either outside SRAM (for example in Flash), or in an SRAM bank that is unaffected by that change. Programs should implement the following items. The code that is used may varies depending on the assembler.

- disable interrupts
- instruction-sync-boundary (ISB)
- cdbank change memory write
- instruction-sync-boundary (ISB)
- enable interrupts

NOTE: The initial SP (Stack Pointer) recorded in the vector table should be set to point to the top of populated data SRAM. This address is always populated as long as there is at least one ConfigBank allocated to DATA.

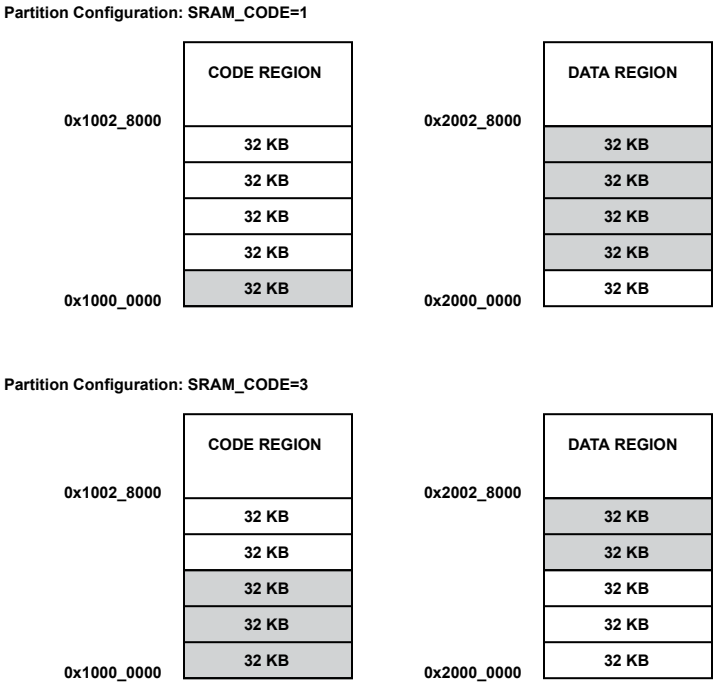

Figure 2-8: Memory Map Configuration, Code and SRAM Regions

SRAM Interface Coherence Specification

The SRAM coherence relations are defined among the four data interfaces on the main SRAM. Note that this is a definition of the ordering relationships of interface transactions, which is not necessarily the same as the ordering model of instructions at the software programmer's application level. For more information on the ARM architecture memory ordering model, see the ARM v7-M Architecture Reference Manual ('Memory access order' section).

The ***Coherence Relations for SRAM Data Interfaces*** table summarizes the coherence relations among the four data interfaces on the main SRAM.

Table 2-10: Coherence Relations for SRAM Data Interfaces

		Cortex-M4 Port Reads			System Reads
		MEM_ICODE	MEM_DCODE	MEM_SYS	SRAM_DMA
Cortex-M4 Port Writes	MEM_ICODE	n/a	n/a	n/a	Non-Coherent
	MEM_DCODE	Coherent	Coherent	n/a	Non-Coherent
	MEM_SYS	n/a	n/a	Coherent	Non-Coherent
System Writes	SRAM_DMA	n/a	n/a	Non-Coherent	Coherent

Some important points to keep in mind regarding the coherence relations include:

- Each of the four interfaces is coherent with itself. For example, any successful (non-error) write on an interface is visible to all subsequent reads on the same interface.

- All ARM Cortex-M4 core ports must be coherent with one another. Because the MEM_SYS port's address map does not overlap with the other two, no coherence relationship between them is defined. The overall relation reduces to the statement that the MEM_ICODE port is coherent with the MEM_DCODE port. A write on the MEM_DCODE interface is visible by all subsequent non-simultaneous read transactions on MEM_ICODE.
- The MEM_xCODE and MEM_SYS interfaces are not coherent with SRAM_DMA. A write on the SRAM_DMA interface is not necessarily visible by subsequent reads on MEM_xCODE or MEM_SYS (and vice versa), unless a Cortex memory system synchronization barrier sequence is used.

Using Synchronization to Achieve SRAM Coherency

System Synchronization assures that a forced coherency can be achieved with otherwise non-coherent ports. There are basically two cases that can arrive at user level, when it comes to non-coherency, as consistent with the above table. The following sequences show the scenarios to insert a synchronization sequence between core and DMA accesses.

The order of operations to achieve SRAM coherency for a core write followed by a DMA read are:

1. Core Write
2. System Synchronization
3. DMA read

The order of operations to achieve SRAM coherency for a prioritized DMA write (automatically done in hardware when DMA is held off for 8 clock cycles) followed by a core read are:

1. Core Write
2. System Synchronization
3. DMA write
4. Core read

SRAM Write Buffers

The SRAM implements write buffers for temporarily storing a write transaction received on an interface that has not yet been physically written into the memory array. Subsequent reads which match the address stored in a write buffer will be forwarded to the interface performing the read, to the extent required to satisfy the requirements of interface coherence. The following write buffers are defined:

- DCODE interface write buffer – one 32-bit transaction
- CSYS interface write buffer– one 32-bit transaction
- SRAM_DMA interface write buffer– one 32-bit transaction

If an interface receives another write transaction while the corresponding write buffer is occupied, the interface is stalled.

SRAM Write Collisions and Write Priority

If two interfaces receive simultaneous, non-exclusive write transactions targeting one or more bytes with matching system memory addresses, then a write collision is said to have occurred. A priority scheme is used to determine which data will be stored in the memory. Write collisions are not errors. Instead, the collisions are handled as if the two transactions had occurred one after another, with the highest priority access occurring last (and thus storing the final value in the memory).

Write priority is determined in the following order:

- DCODE and CSYS interfaces have the highest priority. Because these interfaces target disjoint memory spaces, the priority between them is neither significant nor defined.
- SRAM_DMA has lower priority.

SRAM Access Collisions, Priority, and Stalling

Each ArrayBank is a single-ported SRAM unit. It can only perform one transaction in each CCLK cycle, whether read or write. Access collisions are possible when more than one access is attempted to the same ArrayBank, due to the following considerations:

- Read transactions on up to three interfaces may target a given ArrayBank at the same time (depending on partitioning).
- Write transactions stored in write buffers may also compete to access an ArrayBank. Due to the pipelined nature of writes, a write may collide with a read on the same interface (for example, a write followed by a read, if not colliding at a matching address so as to permit forwarding.)
- Write buffers have variable priority depending on whether a write is stalled due to the write buffer being full.

DMA transactions also have variable priority, in order to guarantee a maximum response latency to the system fabric.

For ConfigBanks assigned to the CODE region, the possible competing access sources are: DCODE read, ICODE read, DCODE write buffer, and SRAM_DMA. For ConfigBanks assigned to the SRAM (data) region, the possible competing access sources are: CSYS read, CSYS write buffer, and SRAM_DMA. The priority in which conflicting sub-transaction accesses to the same LSB-striped ArrayBank are resolved is as follows:

1. ***High-Priority DMA*** has highest priority. This is a transaction on the SRAM_DMA interface which has been stalled and whose stall counter has reached its maximum value (8-1). On the 8'th cycle, the priority of the DMA access is elevated to the highest level and completes immediately.
2. ***DCODE-READ*** has next highest priority.
3. ***High-Priority Write*** is next. This is a transaction in either the DCODE or CSYS write buffer (but not both) as appropriate to the partition assignment of the containing ConfigBank, where (a) the write buffer is full and (b) another write transaction is being attempted on the corresponding DCODE or CSYS interface, causing a stall of that interface. In this event, the priority of that write buffer is elevated.
4. ***ICODE-READ*** has next highest priority.

5. *Normal-Priority Write* is next.
6. *Normal-Priority DMA* has the lowest priority.

SRAM Exclusive Accesses, Global Exclusive Monitor

The SRAM memory has the ARM memory model attributes of normal, shareable memory. Exclusive accesses to this memory space are supported, thus providing semaphores between two software tasks. The main SRAM thus provides a global monitor for exclusive operations, as specified in the ARMv7-M Architecture Manual.

The ARM Cortex-M4 instruction set includes pairs of synchronization primitives. These provide a non-blocking mechanism that a thread or process can use to obtain exclusive access to a memory location. Software can use them to perform a guaranteed read-modify-write memory update sequence, or for a semaphore mechanism.

A pair of synchronization primitives comprises:

- A load-exclusive instruction

  Used to read the value of a memory location, requesting exclusive access to that location.

- A store-exclusive instruction

  Used to attempt to write to the same memory location, returning a status bit to a register. If this bit is:

  - 0 - It indicates that the thread or process gained exclusive access to the memory, and the write succeeds.
  - 1 - It indicates that the thread or process did not gain exclusive access to the memory, and no write was performed.

The pairs of load-exclusive and store-exclusive instructions are:

- the word instructions LDREX and STREX,
- the halfword instructions LDREXH and STREXH, and
- the byte instructions LDREXB and STREXB.

NOTE: The effect of the CLREX instruction on the global monitor is UNDEFINED. If the configuration of the SRAM banks is changed (by modifying the M4P_SRAM_CFG register), the states of all active exclusive memory monitors are unspecified.

SRAM Parity Protection

Parity protection is available for all banks of SRAM. Features include:

- Works for both code and data (the entire SRAM is parity protected)
- Error detection can be enabled separately for core and DMA
- Dedicated interrupt to inform user about parity errors
- Status registers to show what interface triggered the error (I-Code, D-Code, SYS)

The M4P_CACHE_CFG register configures core and DMA parity error generation. The M4P_SRAM_PEADDR_CORE register displays the status of parity errors detected in main SRAM resulting from transactions initiated by the Cortex-M4 core.

- The register captures the address and byte mask of the first parity error(s) detected in any aligned 32-bit word since the last time the corresponding parity error status was cleared in M4P_SRAM_CFG.PERRCORE.
- If core parity error interrupts are enabled in M4P_SRAM_CFG.PERRCORE and a parity error is detected on a core interface, an M4P0_L1CC_PERR SRAM (parity error in code space) interrupt is asserted.

The M4P_SRAM_PEADDR_DMA register displays the status of parity errors detected in main SRAM resulting from transactions initiated by the DMA interface.

- The register captures the address and byte mask of the first parity error(s) detected in any aligned 32-bit word since the last time the corresponding parity error status was cleared in M4P_SRAM_PEADDR_DMA.STAT.
- If DMA parity error interrupts are enabled by M4P_SRAM_CFG.PERRDMA and a parity error is detected on the DMA interface, an M4P0_L1CC_PERR SRAM (parity error in code space) interrupt is asserted.

SRAM Posted System Writes (NormSysWrite versus PostSysWrite)

The M4P_SRAM_CFG.POSTWR bit controls the behavior of nonexclusive writes by the ARM Cortex-M4 core to system space (outside the ARM Cortex-M4 memory). This allows control of the trade off between performance and precise error detection.

In NORMSYSWRITE mode (M4P_SRAM_CFG.POSTWR =0), non-exclusive ARM Cortex-M4 core writes to system space are performed with normal system fabric (bus interconnect) write transactions. The Cortex core must wait several core clock cycles for the target peripheral or device to return a bus response (OKAY or ERROR) before proceeding with further instructions. If an ERROR response is returned, the Cortex core generates a precise HardFault exception at the instruction that caused the error. Typically, an ERROR response results from an invalid address, an invalid data size, or a protection violation such as writing to a read-only location or accessing a privileged resource in non-privileged mode.

In POSTSYSWRITE mode (M4P_SRAM_CFG.POSTWR =1), non-exclusive ARM Cortex-M4 core writes to system space are posted, meaning that the Cortex memory interface accepts the write transaction from the Cortex core and immediately returns an OKAY response. The Cortex core proceeds with subsequent instructions, while independently, the Cortex memory interface forwards the posted write to the system fabric (bus interconnect). Only one write may be posted at any time, so if a subsequent write arrives while the first is pending, the Cortex core is stalled until the Cortex memory interface can accept the new transaction.

If a system fabric (bus interconnect) ERROR response is returned from a posted write, the Cortex core cannot signal an exact HardFault exception because it has already gone on to execute further instructions. IN this case, the Cortex memory captures the address of the erroneous transaction in the M4P_BUSFLT.ADDR field, sets the M4P_BUSFLT.STAT bit to 1, and asserts the interrupt. The handler for this interrupt can determine the offending address using the M4P_BUSFLT register and take appropriate action. The interrupt is cleared by writing a 1 to the M4P_BUSFLT.STAT bit.

ADSP-CM40x M4P Register Descriptions

ARM Cortex-M4 Platform (M4P) contains the following registers.

Table 2-11: ADSP-CM40x M4P Register List

Name	Description
M4P_BUSFLT	Bus Fault Error Information Register
M4P_CACHE_CFG	Code Cache Configuration and Status Register
M4P_CACHE_CNTCTL	Cache Counter Control Register
M4P_CACHE_DFILL	Cache DCODE Line Fill Counter Register
M4P_CACHE_DMISS	Cache DCODE Miss Counter Register
M4P_CACHE_DREF	Cache DCODE Reference Counter Register
M4P_CACHE_IFILL	Cache ICODE Line Fill Counter Register
M4P_CACHE_IMISS	Cache ICODE Miss Counter Register
M4P_CACHE_IREF	Cache ICODE Reference Counter Register
M4P_CACHE_MEMX	MEMX Space Configuration Register
M4P_CACHE_MEMY	MEMY Space Configuration Register
M4P_CACHE_PEADDR	Code Cache Parity Error Address Register
M4P_SRAM_CFG	SRAM Configuration Register
M4P_SRAM_PEADDR_CORE	SRAM Parity Error Address (Core) Register
M4P_SRAM_PEADDR_DMA	SRAM Parity Error Address (DMA) Register
M4P_STCALIB	SysTick Calibration Register

Bus Fault Error Information Register

The M4P_BUSFLT register captures the status and address of bus fault errors resulting from inexact posted writes by the Cortex to System space. Posted writes are enabled by the M4P_SRAM_CFG.POSTWR bit. The M4P_BUSFLT.STAT bit drives the M4P_BUS_FAULT interrupt, and is set on the detection of a posted write bus fault, and is cleared if written with a 1.

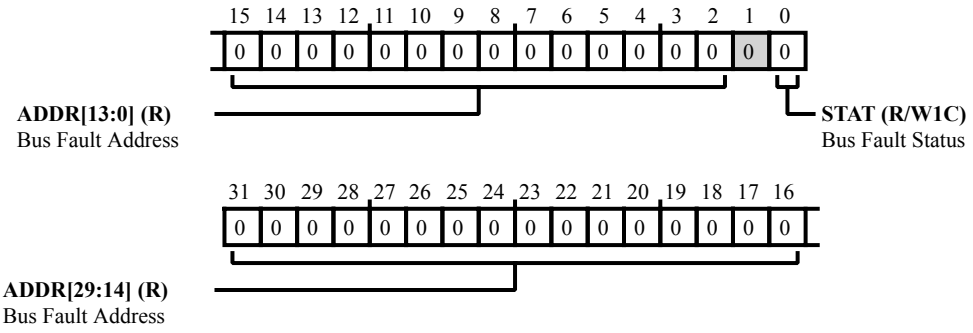

Figure 2-9: M4P_BUSFLT Register Diagram

Table 2-12: M4P_BUSFLT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:2 (R/NW)	ADDR	Bus Fault Address. The M4P_BUSFLT.ADDR bit field provides the address of bus fault errors resulting from inexact posted writes by the Cortex to System space.
0 (R/W1C)	STAT	Bus Fault Status. The M4P_BUSFLT.STAT bit provides the status of bus fault errors resulting from inexact posted writes by the Cortex to System space.

Code Cache Configuration and Status Register

The M4P_CACHE_CFG register controls cache configuration and reports cache status. There are no restrictions limiting when any of the bits in this register may be changed.

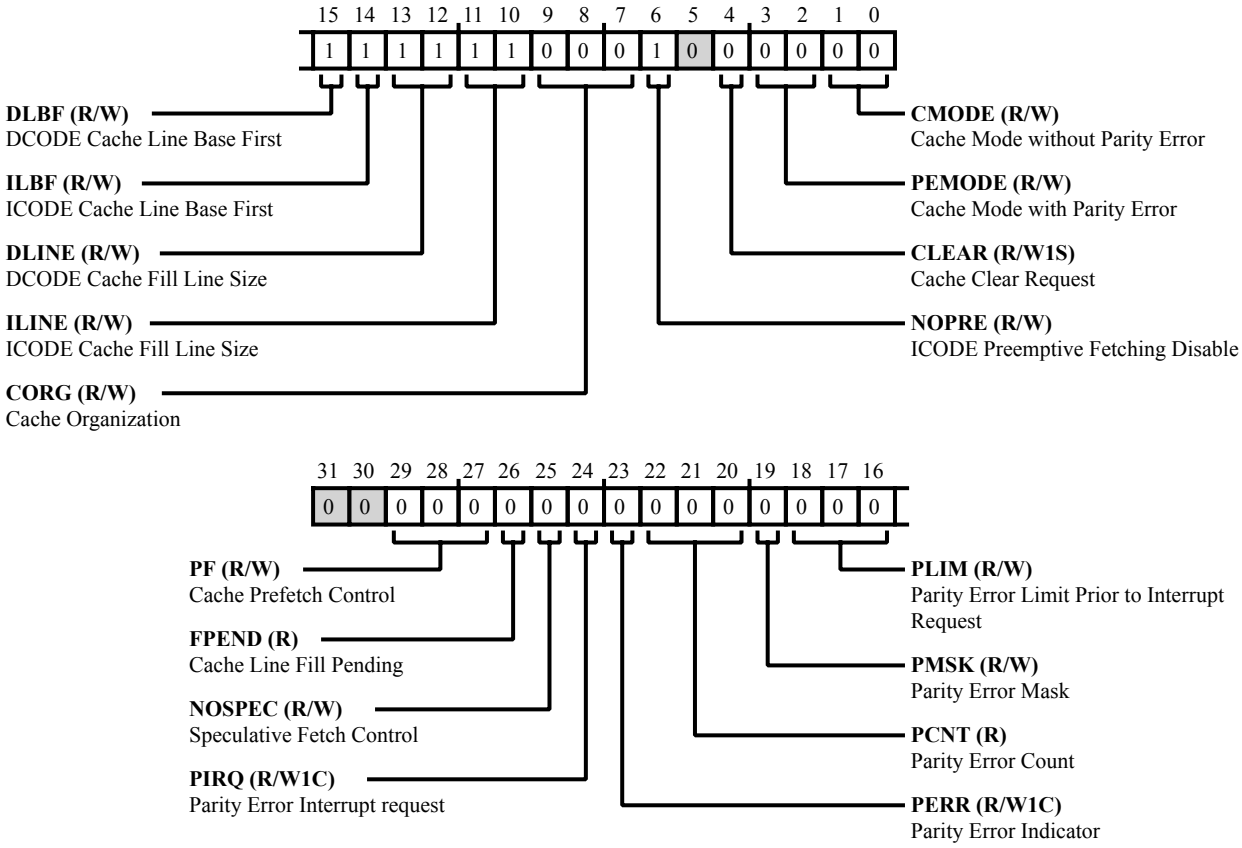

Figure 2-10: M4P_CACHE_CFG Register Diagram

Table 2-13: M4P_CACHE_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:27 (R/W)	PF	Cache Prefetch Control. The M4P_CACHE_CFG.PF bits control the cache prefetcher. A setting of =000 disables the cache prefetch. Settings of =001 to =111 increase the cache prefetch prediction interval, where 111 is the most aggressive prediction.
26 (R/NW)	FPEND	Cache Line Fill Pending. The M4P_CACHE_CFG.FPEND bit indicates if any cache line fills are pending in the system fabric. The bit is set to 1 when a fill is initiated by an ICODE or DCODE cache miss, or when a speculative ICODE or DCODE cache fill is initiated (if speculative fills are enabled). The bit is cleared to 0 when all SCB read responses have been

Table 2-13: M4P_CACHE_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		received by the cache and associated data has been drained into the cache tag and data arrays.	
25 (R/W)	NOSPEC	Speculative Fetch Control. The M4P_CACHE_CFG.NOSPEC bit controls speculative fetching.	
		0	Enable Speculative Fetching
		1	Disable Speculative Fetching
24 (R/W1C)	PIRQ	Parity Error Interrupt request. The M4P_CACHE_CFG.PIRQ bit reflects the state of the Parity Error interrupt request signal provided to the processor. This signal, and this bit, are set if either PCNT[2:0] (Parity Error Count) is greater than PLIM[2:0] (Parity Error Limit), or if PERR is asserted while not masked by PMSK (i.e., PERR and !PMSK). This bit is cleared if a 1 is written to it, unless currently being set. The location of the parity error which causes this bit to transition from LOW to HIGH is stored in the Parity Error Address Reporting register.	
23 (R/W1C)	PERR	Parity Error Indicator. The M4P_CACHE_CFG.PERR bit is asserted if a fetch return provided to the processor from cache, prior to corrective action possibly being taken by hardware, was found to contain a parity error anywhere within the associated cache Set. This bit is cleared if a 1 is written to it. Unlike PCNT[2:0] which reports parity errors only as corrective action is necessitated for immediate processor fetch return, this bit reports all parity errors directly or indirectly associated with return provided to the processor.	
22:20 (R/NW)	PCNT	Parity Error Count. Until saturated at b#111, the M4P_CACHE_CFG.PCNT bits are incremented by hardware each time a word must be fetched from backing memory due to a cache miss with an associated parity error. Since hardware never allows Ways with parity errors to hit, and Ways with parity errors are preferentially chosen for cache replacement, these bits count the number of parity errors repaired or overwritten. These bits are cleared whenever a parity error interrupt is cleared, that is, when a 1 is written to bit PIRQ.	
19 (R/W)	PMSK	Parity Error Mask. The M4P_CACHE_CFG.PMSK bit selects whether or not a Parity error interrupt is requested if bit PERR is asserted.	
		0	Parity error interrupt is requested if bit PERR is asserted
		1	Parity error interrupt is not requested based on state of bit PERR
18:16 (R/W)	PLIM	Parity Error Limit Prior to Interrupt Request. A Parity Error Interrupt will be requested any time PCNT[2:0] exceeds the limit specified by the M4P_CACHE_CFG.PLIM bits. These bits therefore specify the number of parity errors which may be repaired by hardware (through updates retrieved from backing memory) prior to a Parity Error interrupt being requested. But, interrupting	

Table 2-13: M4P_CACHE_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		the processor upon each discovered parity error might not always be desirable. Note that parity errors do not trigger fetches from backing memory (increment of PCNT[2:0]) if a cache hit is simultaneously found in a uncorrupted cache Way.	
		0	Generate a Parity Error interrupt request if PCNT[2:0] is 1 or greater
		1	Generate a Parity Error interrupt request if PCNT[2:0] is 2 or greater
		2	Generate a Parity Error interrupt request if PCNT[2:0] is 3 or greater
		3	Generate a Parity Error interrupt request if PCNT[2:0] is 4 or greater
		4	Generate a Parity Error interrupt request if PCNT[2:0] is 5 or greater
		5	Generate a Parity Error interrupt request if PCNT[2:0] is 6 or greater
		6	Generate a Parity Error interrupt request if PCNT[2:0] is 7 or greater
		7	Never Generate a Parity Error interrupt request
15 (R/W)	DLBF	DCODE Cache Line Base First. The `M4P_CACHE_CFG.DLBF` bit specifies the first word to be fetched from backing memory during line fills initiated by DCODE cache misses. Either the critical word, which is the word requested by the processor which initiated the fill, or the cache line base may be specified. Critical-word-first fetch has the advantage of reducing the latency of delivering the word which caused the cache line miss (and thus shortening the processor stall), but at a potential efficiency cost. Memory systems such as SPI which support line-wrap accesses may impose an overhead per burst which might be avoided for linearly incrementing accesses which span multiple cache lines. Line-base-first mode may offer higher efficiency in such cases.	
		0	Commence cache fills with critical word (Quicker critical word return to cache, slower full line retrieval from SPI memory)
		1	Commence cache fills with line base word (Slower critical word return to cache, quicker full line retrieval from SPI memory)

Table 2-13: M4P_CACHE_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	ILBF	ICODE Cache Line Base First. The M4P_CACHE_CFG.ILBF bit specifies the first word to be fetched from backing memory during line fills initiated by ICODE cache misses. Either the critical word, which is the word requested by the processor which initiated the fill, or the cache line base may be specified. Critical-word-first fetch has the advantage of reducing the latency of delivering the word which caused the cache line miss (and thus shortening the processor stall), but at a potential efficiency cost. Memory systems such as SPI which support line-wrap accesses may impose an overhead per burst which might be avoided for linearly incrementing accesses which span multiple cache lines. Line-base-first mode may offer higher efficiency in such cases.	
		0	Commence cache fills with critical word (Quicker critical word return to cache, slower full line retrieval from SPI memory)
		1	Commence cache fills with line base word (Slower critical word return to cache, quicker full line retrieval from SPI memory)
13:12 (R/W)	DLINE	DCODE Cache Fill Line Size. The M4P_CACHE_CFG.DLINE bits specify the size of a cache fill initiated upon a DCODE fetch miss.	
		0	32 bits (1 word, 4 bytes)
		1	64 bits (2 words, 8 bytes)
		2	128 bits (4 words, 16 bytes)
		3	256 bits (8 words, 32 bytes)
11:10 (R/W)	ILINE	ICODE Cache Fill Line Size. The M4P_CACHE_CFG.ILINE bits specify the size of a cache fill initiated upon an ICODE fetch miss.	
		0	32 bits (1 word, 4 bytes)
		1	64 bits (2 words, 8 bytes)
		2	128 bits (4 words, 16 bytes)
		3	256 bits (8 words, 32 bytes)
9:7 (R/W)	CORG	Cache Organization. Cache is comprised of eight independent 2kB banks of SRAM providing a total of 16kB. By default these banks are organized into a four way, set associative cache shared by the ICODE and DCODE ports. Each cache Way is constructed from two banks of SRAM which are address striped such that even and odd 32 bit words may be accessed simultaneously. Striping reduces the likelihood of processor stalls by reducing the likelihood that simultaneous demands of cache memory (Fill writes, DCODE fetches, and ICODE fetches) collide within a particular bank, and need to be sequenced by	

Table 2-13: M4P_CACHE_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		hardware. The M4P_CACHE_CFG.CORG bits may be used to select alternate cache organization so that cache organization is best optimized for the nature of code being executed. Cache Ways may be sacrificed from the default to either increase address striping, or create two independent smaller caches with one dedicated to each processors code port (ICODE and DCODE). Cache organization may be changed at any time, however since previously cached data may become inaccessible, increased cache missing immediately after a cache configuration changes should be anticipated.	
		0	Single 16 kB four way cache LS striped by 2 shared by ICODE and DCODE
		1	Single 16 kB two way cache LS striped by 4 shared by ICODE and DCODE
		2	Single 16 kB one way cache LS striped by 8 shared by ICODE and DCODE
		3	Reserved
		4	4 kB one Way DCODE Cache and 12 kB three Way ICODE Cache
		5	8 kB two Way DCODE Cache and 8 kB two Way ICODE Cache
		6	12 kB three Way DCODE Cache and 4 kB one Way ICODE Cache
		7	Reserved
6 (R/W)	NOPRE	ICODE Preemptive Fetching Disable. By default, cache banks seek to preemptively acquire data immediately sequential to the current ICODE address. The M4P_CACHE_CFG.NOPRE bit (preemptive acquisition) improves SRAM pseudo-porting, but may result in greater power consumption if the processor never requires data preemptively acquired.	
		0	ICODE preemptive fetching from cache enabled
		1	ICODE preemptive fetching from cache disabled
4 (R/W1S)	CLEAR	Cache Clear Request. When a 1 is written to the M4P_CACHE_CFG.CLEAR bit, hardware will initiate a process of erasing all cache line data and attributes. Approximately 512 processor clocks will be required. While clearing is in progress, this bit will return a 1 when read. Hardware will clear this bit to 0 when cache clearing has completed and all outstanding fill operations started prior to the clear have ended. Access to backing memory will be possible while cache is being cleared, but fetches from this backing memory will not be stored in cache.	

Table 2-13: M4P_CACHE_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:2 (R/W)	PEMODE	Cache Mode with Parity Error. The `M4P_CACHE_CFG.PEMODE` bits specify the level of cache servicing provided when a Parity Error interrupt (see bit `M4P_CACHE_CFG.PIRQ`) is being requested by cache.	
		0	Full Operation: Cache provides data upon hit; cache contents updated upon miss
		1	Partial Bypass: Cache provides data upon hit; cache bypassed upon miss (cache content not updated)
		2	Reserved
		3	Full Bypass: Cache state preserved and inaccessible except through backdoor access (see bit CBEN)
1:0 (R/W)	CMODE	Cache Mode without Parity Error. The `M4P_CACHE_CFG.CMODE` bits specify the level of cache servicing provided when a Parity Error interrupt (see bit PIRQ) is not being requested by cache.	
		0	Full Operation: Cache provides data upon hit; cache contents updated upon miss
		1	Partial Bypass: Cache provides data upon hit; cache bypassed upon miss (cache content not updated)
		2	Reserved
		3	Full Bypass: Cache state preserved and inaccessible

Cache Counter Control Register

The M4P_CACHE_CNTCTL register controls operation of the cache profiling counters, which support measuring cache activity during code execution. These counters include M4P_CACHE_IREF, M4P_CACHE_DREF, M4P_CACHE_IMISS, M4P_CACHE_DMISS, M4P_CACHE_IFILL, and M4P_CACHE_DFILL.

Each counter is 24 bits plus 1 (sticky) overflow bit. The sticky status can be cleared by clearing the counter or by writing zero to the count register. Each counter consists of a main and shadow register.

All counters may be sampled simultaneously from the main registers into the shadow registers to collect a consistent set of data. To enable consistent counting without doubled or missed counts, the main counters may be zeroed just after sampling. The main counters continue counting cache activity after sampling to the shadow registers.

When working with the cache counters, it is important to understand the difference between committed and speculative cache activity. Cache activity may be initiated either by a committed cache access from the M4 core over the I-code or D-code buses or may be initiated by speculative accesses by the M4 core. A committed access is an address issued in a bus cycle, which is acknowledged by HREADY. A speculative cache access is initiated by the M4P cache observing the address bus prior to HREADY (for example, during a wait state). For more information, see the cache structure/operation description.

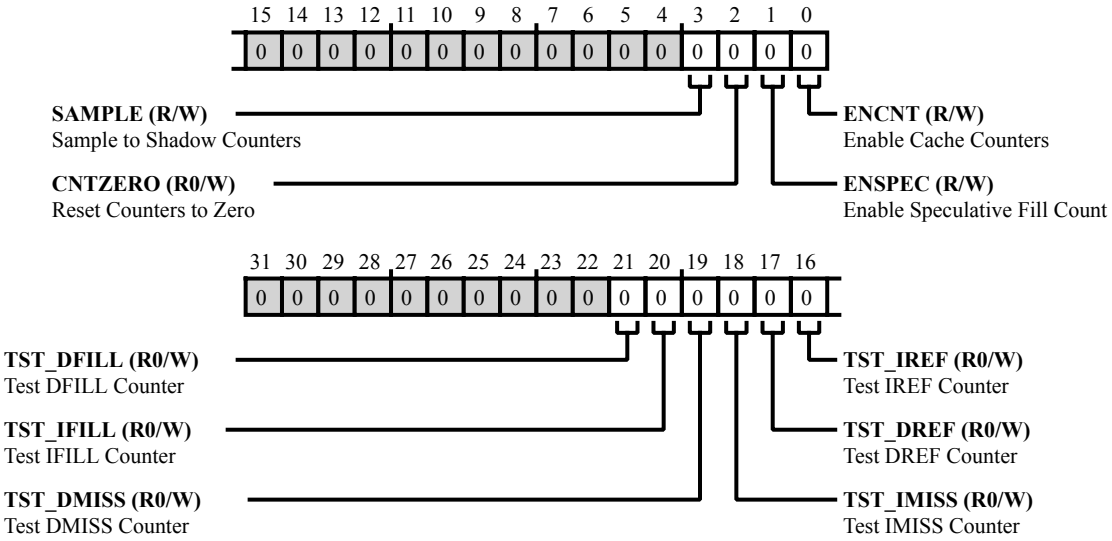

Figure 2-11: M4P_CACHE_CNTCTL Register Diagram

Table 2-14: M4P_CACHE_CNTCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R0/W)	TST_DFILL	Test DFILL Counter. The M4P_CACHE_CNTCTL.TST_DFILL bit enables increment of the associated counter, M4P_CACHE_DFILL.	
		0	No Count
		1	Count
20 (R0/W)	TST_IFILL	Test IFILL Counter. The M4P_CACHE_CNTCTL.TST_IFILL bit enables increment of the associated counter, M4P_CACHE_IFILL.	
		0	No Count
		1	Count
19 (R0/W)	TST_DMISS	Test DMISS Counter. The M4P_CACHE_CNTCTL.TST_DMISS bit enables increment of the associated counter, M4P_CACHE_DMISS.	
		0	No Count
		1	Count
18 (R0/W)	TST_IMISS	Test IMISS Counter. The M4P_CACHE_CNTCTL.TST_IMISS bit enables increment of the associated counter, M4P_CACHE_IMISS.	
		0	No Count
		1	Count
17 (R0/W)	TST_DREF	Test DREF Counter. The M4P_CACHE_CNTCTL.TST_DREF bit enables increment of the associated counter, M4P_CACHE_DREF.	
		0	No Count
		1	Count
16 (R0/W)	TST_IREF	Test IREF Counter. The M4P_CACHE_CNTCTL.TST_IREF bit enables increment of the associated counter, M4P_CACHE_IREF.	
		0	No Count
		1	Count

Table 2-14: M4P_CACHE_CNTCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	SAMPLE	Sample to Shadow Counters. The `M4P_CACHE_CNTCTL.SAMPLE` bit controls both the update of the shadow registers and the read access to these registers. When this bit is zero, the main (free-running) counters are returned by an memory-mapped register read. When this bit is written 0-to-1, the main counter values are transferred to the shadow registers; if the count zero bit is also written from 0-to-1 at that time, the main registers are zeroed after the sample is taken. While the `M4P_CACHE_CNTCTL.SAMPLE` bit's value is 1, reads to any of the count registers return the sampled value held in the shadow registers. When the bit is written to 0 again, the values that are in the main registers are again visible.	
		0	Show Main Counters
		1	Show Shadow Counters
2 (R0/W)	CNTZERO	Reset Counters to Zero. The `M4P_CACHE_CNTCTL.CNTZERO` bit resets all cache counters.	
		0	No Action
		1	Reset Cache Counters
1 (R/W)	ENSPEC	Enable Speculative Fill Count. The `M4P_CACHE_CNTCTL.ENSPEC` bit selects whether the cache counters count only committed cache activity (if =0) or whether the cache counters count all cache activity (include both committed and speculative accesses).	
		0	Disable Speculative Count
		1	Enable Speculative Count
0 (R/W)	ENCNT	Enable Cache Counters. The `M4P_CACHE_CNTCTL.ENCNT` bit enables cache counter increment operations.	
		0	Disable Cache Counters
		1	Enable Cache Counters

Cache DCODE Line Fill Counter Register

The M4P_CACHE_DFILL register records cache misses in the cache line fills initiated by cache misses on the D-code bus and indicates count overflow (MSB). As each line fill operation takes a discrete length of time on the interface to the cache memory backing store, the rate of line fills is an indication of efficiency. If the number of line fills time the duration per line fills (in SysTicks), this may indicate that code performance is not limited by cache activity.

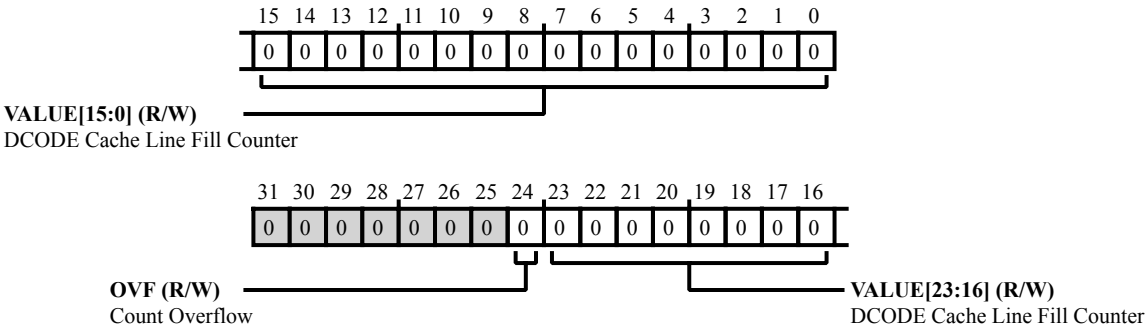

Figure 2-12: M4P_CACHE_DFILL Register Diagram

Table 2-15: M4P_CACHE_DFILL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24 (R/W)	OVF	Count Overflow. The M4P_CACHE_DFILL.OVF bit (if set) indicates an overflow of the counter. This bit is a sticky status bit, remaining set until cleared with a write.
23:0 (R/W)	VALUE	DCODE Cache Line Fill Counter. The M4P_CACHE_DFILL.VALUE bits hold the count for cache line fills for accesses to D-code.

Cache DCODE Miss Counter Register

The M4P_CACHE_DMISS register records cache misses in the cache memory region on the D-code bus and indicates count overflow (MSB). Counts include misses only. The hit count can be calculated from the reference count minus the miss count.

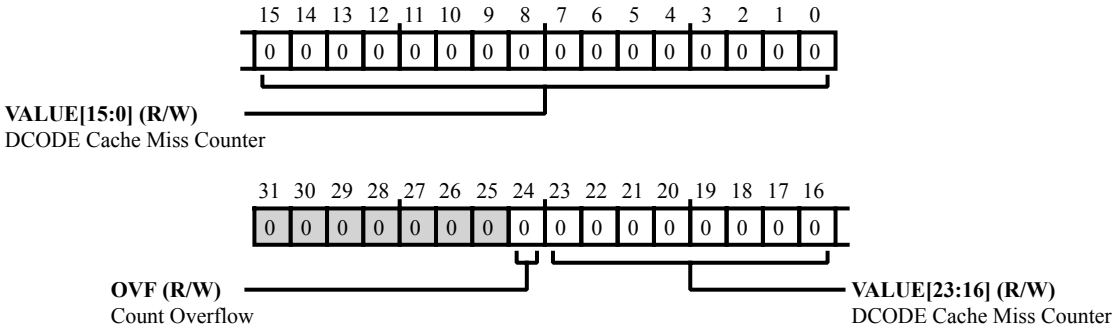

Figure 2-13: M4P_CACHE_DMISS Register Diagram

Table 2-16: M4P_CACHE_DMISS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24 (R/W)	OVF	Count Overflow. The M4P_CACHE_DMISS.OVF bit (if set) indicates an overflow of the counter. This bit is a sticky status bit, remaining set until cleared with a write.
23:0 (R/W)	VALUE	DCODE Cache Miss Counter. The M4P_CACHE_DMISS.VALUE bits hold the count for cache misses to D-code.

Cache DCODE Reference Counter Register

The M4P_CACHE_DREF register records references to the cache memory region on the D-code bus, which includes literal fetches from data embedded in code space. Counts include all hits plus all misses and indicates count overflow (MSB).

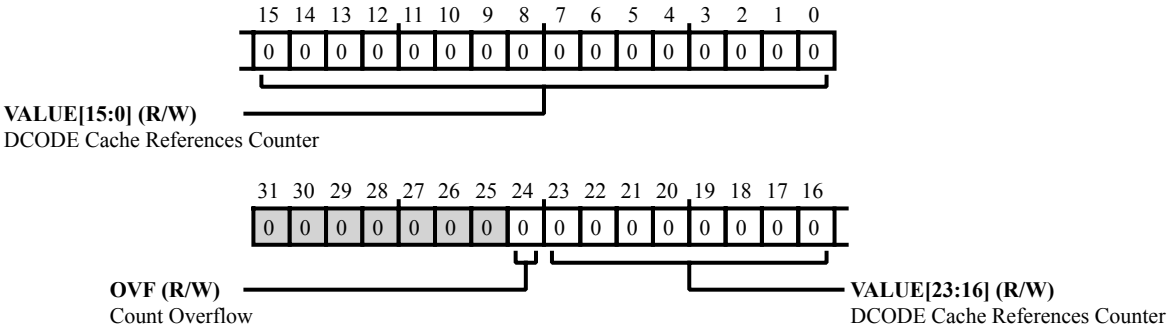

Figure 2-14: M4P_CACHE_DREF Register Diagram

Table 2-17: M4P_CACHE_DREF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24 (R/W)	OVF	Count Overflow. The M4P_CACHE_DREF.OVF bit (if set) indicates an overflow of the counter. This bit is a sticky status bit, remaining set until cleared with a write.
23:0 (R/W)	VALUE	DCODE Cache References Counter. The M4P_CACHE_DREF.VALUE bits hold the count for cache accesses (references) to D-code.

Cache ICODE Line Fill Counter Register

The M4P_CACHE_IFILL register records cache misses in the cache line fills initiated by cache misses on the I-code bus and indicates count overflow (MSB). As each line fill operation takes a discrete length of time on the interface to the cache memory backing store, the rate of line fills is an indication of efficiency. If the number of line fills time the duration per line fills (in SysTicks), this may indicate that code performance is not limited by cache activity.

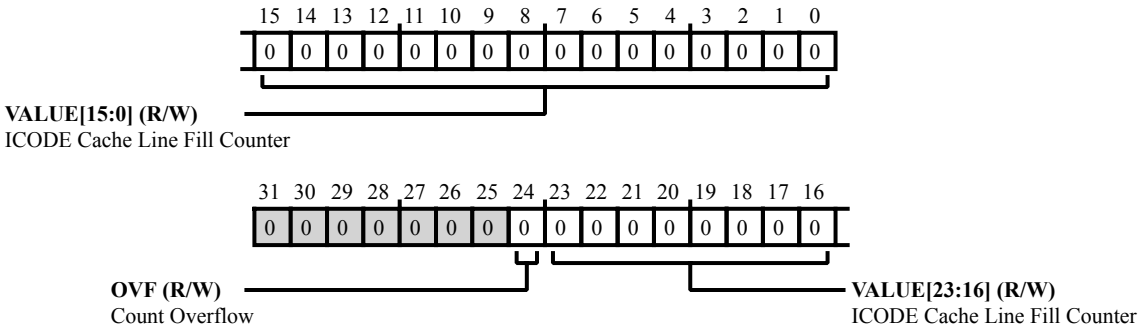

Figure 2-15: M4P_CACHE_IFILL Register Diagram

Table 2-18: M4P_CACHE_IFILL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24 (R/W)	OVF	Count Overflow. The M4P_CACHE_IFILL.OVF bit (if set) indicates an overflow of the counter. This bit is a sticky status bit, remaining set until cleared with a write.
23:0 (R/W)	VALUE	ICODE Cache Line Fill Counter. The M4P_CACHE_IFILL.VALUE bits hold the count for cache line fills for accesses to I-code.

Cache ICODE Miss Counter Register

The M4P_CACHE_IMISS register records cache misses in the cache memory region on the I-code bus and indicates count overflow (MSB). Counts include misses only. The hit count can be calculated from the reference count minus the miss count.

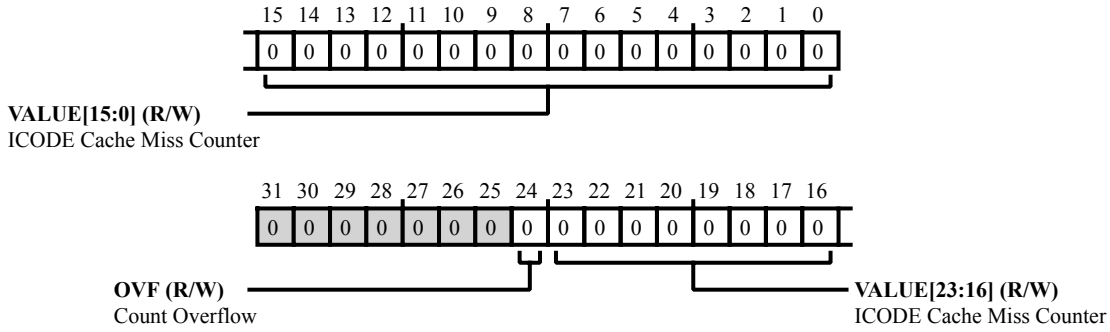

Figure 2-16: M4P_CACHE_IMISS Register Diagram

Table 2-19: M4P_CACHE_IMISS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24 (R/W)	OVF	Count Overflow. The M4P_CACHE_IMISS.OVF bit (if set) indicates an overflow of the counter. This bit is a sticky status bit, remaining set until cleared with a write.
23:0 (R/W)	VALUE	ICODE Cache Miss Counter. The M4P_CACHE_IMISS.VALUE bits hold the count for cache misses to I-code.

Cache ICODE Reference Counter Register

The M4P_CACHE_IREF register records references to the cache memory region on the I-code bus, which includes code fetches and vector table reads. Counts include all hits plus all misses and indicates count overflow (MSB).

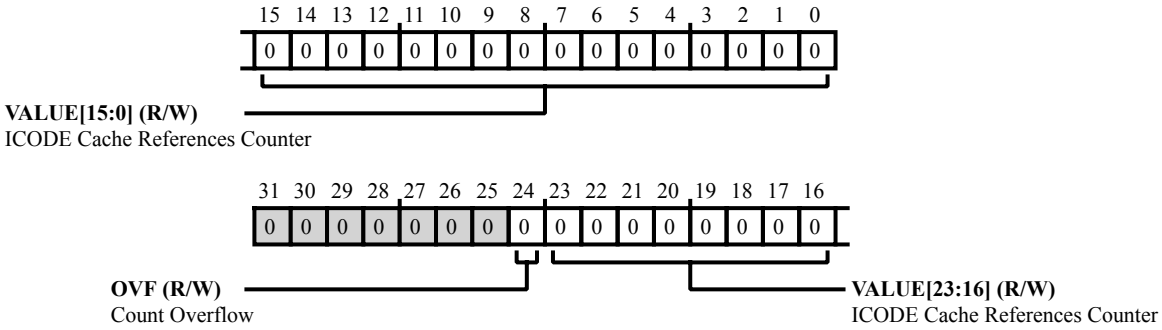

Figure 2-17: M4P_CACHE_IREF Register Diagram

Table 2-20: M4P_CACHE_IREF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24 (R/W)	OVF	Count Overflow. The M4P_CACHE_IREF.OVF bit (if set) indicates an overflow of the counter. This bit is a sticky status bit, remaining set until cleared with a write.
23:0 (R/W)	VALUE	ICODE Cache References Counter. The M4P_CACHE_IREF.VALUE bits hold the count for cache accesses (references) to I-code.

MEMX Space Configuration Register

The M4P_CACHE_MEMX register selects the MEMX space configuration.

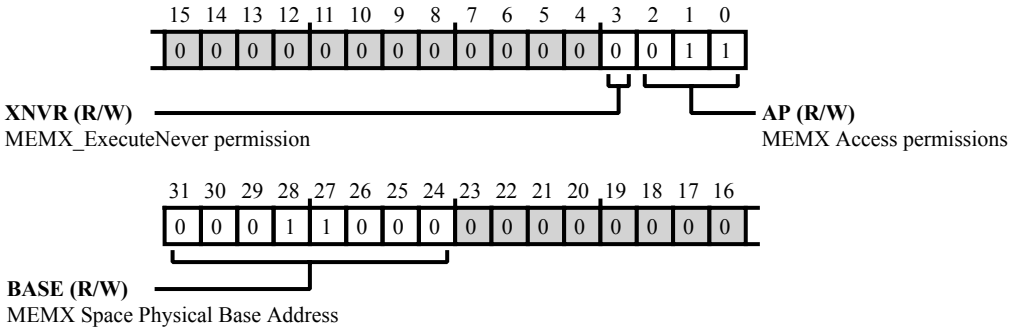

Figure 2-18: M4P_CACHE_MEMX Register Diagram

Table 2-21: M4P_CACHE_MEMX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:24 (R/W)	BASE	MEMX Space Physical Base Address. The M4P_CACHE_MEMX.BASE bits specify the 8 physical (bus) address MSBs to be used for serving code cache line fills to the MEMX region in application address space.	
3 (R/W)	XNVR	MEMX_ExecuteNever permission. The M4P_CACHE_MEMX.XNVR bits specify the core instruction fetch access permissions to the 16MB region of physical memory specified by M4P_CACHE_MEMX.BASE. This uses the same access codes as the ARM MPU; see the ARM v7M Architecture Reference Manual sec. B3.5.7.	
		0	Execution of fetched instructions permitted
		1	Execution of fetched instructions not permitted
2:0 (R/W)	AP	MEMX Access permissions. The M4P_CACHE_MEMX.AP bits specify the core access permissions to the 16MB region of physical memory specified by M4P_CACHE_MEMX.BASE. This uses the same access codes as the ARM MPU; see the ARM v7M Architecture Reference Manual sec. B3.5.7.	
		0	Any access generates a permission fault
		1	Privileged access only
		2	Privileged access and non-privileged Read access only
		3	Full access
		5	Privileged Read-Only
		6	Privileged and unprivileged Read-Only

Table 2-21: M4P_CACHE_MEMX Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		7	Privileged and unprivileged Read-Only

MEMY Space Configuration Register

The M4P_CACHE_MEMY register selects the MEMY space configuration.

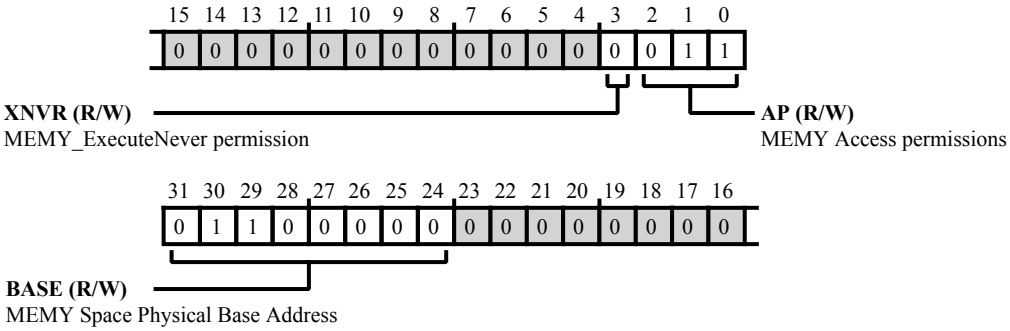

Figure 2-19: M4P_CACHE_MEMY Register Diagram

Table 2-22: M4P_CACHE_MEMY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:24 (R/W)	BASE	MEMY Space Physical Base Address. The M4P_CACHE_MEMY.BASE bits specify the 8 physical (bus) address MSBs to be used for serving code cache line fills to the MEMY region in application address space.	
3 (R/W)	XNVR	MEMY_ExecuteNever permission. The M4P_CACHE_MEMY.XNVR bits specify the core instruction fetch access permissions to the 16MB region of physical memory specified by M4P_CACHE_MEMY.BASE. This uses the same access codes as the ARM MPU; see the ARM v7M Architecture Reference Manual sec. B3.5.7.	
		0	Execution of fetched instructions permitted
		1	Execution of fetched instructions not permitted
2:0 (R/W)	AP	MEMY Access permissions. The M4P_CACHE_MEMY.AP bits specify the core access permissions to the 16MB region of physical memory specified by M4P_CACHE_MEMY.BASE. This uses the same access codes as the ARM MPU; see the ARM v7M Architecture Reference Manual sec. B3.5.7.	
		0	Any access generates a permission fault
		1	Privileged access only
		2	Privileged access and non-privileged Read access only
		3	Full access
		5	Privileged Read-Only
		6	Privileged and unprivileged Read-Only

Table 2-22: M4P_CACHE_MEMY Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		7	Privileged and unprivileged Read-Only

Code Cache Parity Error Address Register

The M4P_CACHE_PEADDR register is loaded each rising edge of M4P_CACHE_CFG.PIRQ. It can report two different kinds of parity error: an error in the cache data/attribute banks, indicated by M4P_CACHE_PEADDR.BNKERR, or an error in the state machine memory, indicated by M4P_CACHE_PEADDR.REPERR.

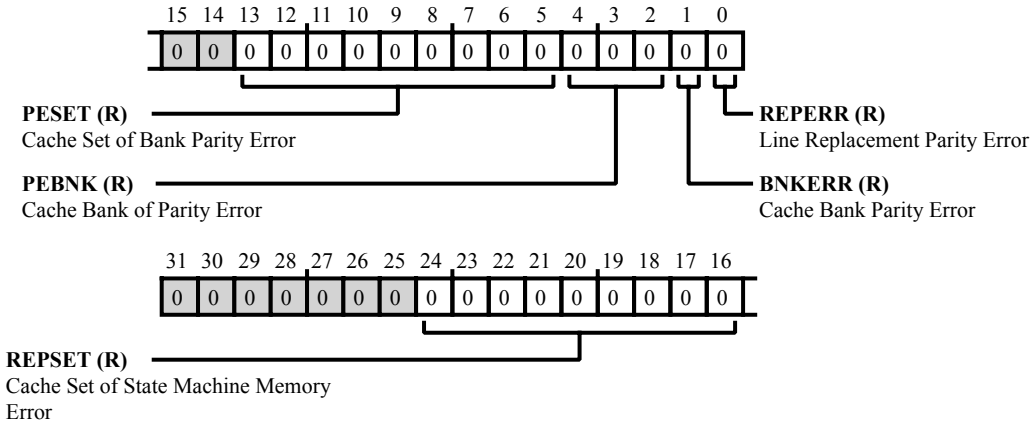

Figure 2-20: M4P_CACHE_PEADDR Register Diagram

Table 2-23: M4P_CACHE_PEADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:16 (R/NW)	REPSET	Cache Set of State Machine Memory Error. The M4P_CACHE_PEADDR.REPSET bits provide the cache set where the state machine memory error occurred.
13:5 (R/NW)	PESET	Cache Set of Bank Parity Error. The M4P_CACHE_PEADDR.PESET bits provide the cache set where the bank parity error occurred.
4:2 (R/NW)	PEBNK	Cache Bank of Parity Error. The M4P_CACHE_PEADDR.PEBNK bits provide the cache bank where the parity error occurred.
1 (R/NW)	BNKERR	Cache Bank Parity Error. The M4P_CACHE_PEADDR.BNKERR bit indicates that a parity error has occurred in a cache data/attribute bank. If M4P_CACHE_PEADDR.BNKERR is 1, then the set number of the error (the array address) is available in M4P_CACHE_PEADDR.PESET and the bank number of the error is available in M4P_CACHE_PEADDR.PEBNK.
0 (R/NW)	REPERR	Line Replacement Parity Error. The M4P_CACHE_PEADDR.REPERR bit indicates if a parity error has been detected in the cache Line Replacement array (REP). If M4P_CACHE_PEADDR.REPERR is 1, then the M4P_CACHE_PEADDR.REPSET field indicates the address in the array (the cache set number) of the error.

SRAM Configuration Register

The M4P_SRAM_CFG register selects the SRAM configuration.

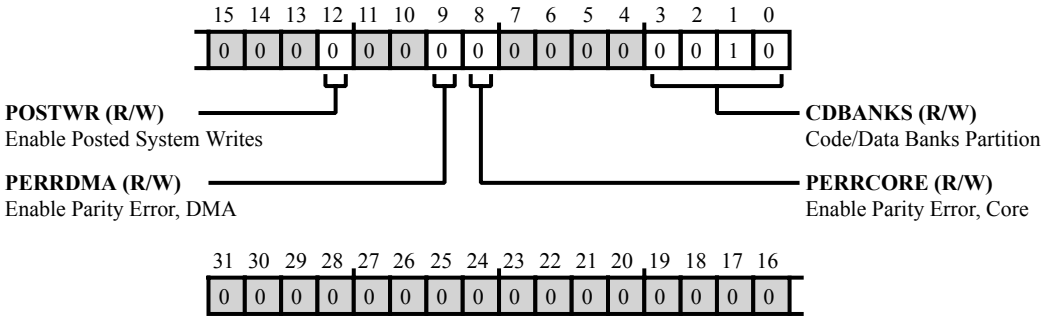

Figure 2-21: M4P_SRAM_CFG Register Diagram

Table 2-24: M4P_SRAM_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	POSTWR	Enable Posted System Writes. The M4P_SRAM_CFG.POSTWR bit controls the behavior of non-exclusive writes by the Cortex to System space (outside the M4P platform). This allows control of the tradeoff between performance and precise error detection. For more information, see the "SRAM Posted System Writes (NormSysWrite versus PostSysWrite)" section.	
		0	Normal System Writes Non-exclusive Cortex writes to System space are performed with normal system fabric (bus interconnect) write transactions
		1	Posted System Writes Non-exclusive Cortex writes to System space are posted
9 (R/W)	PERRDMA	Enable Parity Error, DMA. Enables the Parity Error Core Interrupt. If M4P_SRAM_CFG.PERRDMA is 1 and a parity error is detected in the Main SRAM due to a transaction initiated on one the DMA interface (as indicated by the M4P_SRAM_PEADDR_DMA.STAT bit, then the SRAM_PE_IRQ_DMA interrupt is asserted. If M4P_SRAM_CFG.PERRDMA is 0, then the SRAM_PE_IRQ_DMA interrupt will be deasserted.	
8 (R/W)	PERRCORE	Enable Parity Error, Core. Enables the Parity Error Core Interrupt. If M4P_SRAM_CFG.PERRCORE is 1 and a parity error is detected in the Main SRAM due to a transaction initiated on one of the three M4- Cortex interfaces (as indicated by the M4P_SRAM_PEADDR_CORE.STAT bit, then the SRAM_PE_IRQ_CORE interrupt is asserted. If M4P_SRAM_CFG.PERRCORE is 0, then the SRAM_PE_IRQ_CORE interrupt will be deasserted.	
3:0 (R/W)	CDBANKS	Code/Data Banks Partition.	

Table 2-24: M4P_SRAM_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		The M4P_SRAM_CFG.CDBANKS field controls the partitioning of Main SRAM resources between the CODE address range and the SRAM (DATA) address range . The M4P_SRAM_CFG.CDBANKS field indicates how many of the 32KByte Config Banks of SRAM are allocated to the CODE region; the remaining available banks are allocated to the SRAM (DATA) region. When the M4P_SRAM_CFG.CDBANKS field is changed, the contents of any banks which are deducted from a given region are lost. The contents of any banks which are added to a given region are undefined, but should be zeroed immediately for security purposes.	
		0	No CODE Config Banks: all assigned to Data
		1	1 CODE Config Bank
		2	2 CODE Config Banks
		3	3 CODE Config Banks
		4	4 CODE Config Banks
		5	5 CODE Config Banks
		6	6 CODE Config Banks

SRAM Parity Error Address (Core) Register

The M4P_SRAM_PEADDR_CORE register displays the status of parity errors detected in Main SRAM resulting from transactions initiated by the Cortex M4 Core. The register captures the address and byte mask of the first parity error(s) detected in any aligned 32-bit word since the last time the corresponding parity error status was cleared (by writing a 1 to M4P_SRAM_PEADDR_CORE.STAT.) If core parity error interrupts are enabled by M4P_SRAM_CFG.PERRCORE, and a parity error is detected on a core interface, then an SRAM_PEIRQ_CODE interrupt will be asserted. If another core parity error is detected while a first core parity error is still pending, then the M4P_SRAM_PEADDR_CORE.MSTAT bit is asserted, and the SRAM Parity Error Non-Maskable Interrupt (NMI) is asserted.

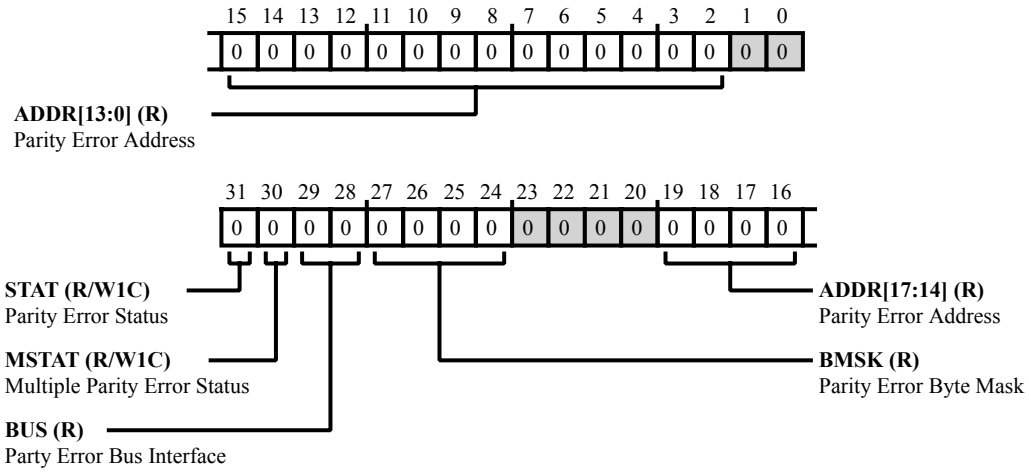

Figure 2-22: M4P_SRAM_PEADDR_CORE Register Diagram

Table 2-25: M4P_SRAM_PEADDR_CORE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	STAT	Parity Error Status. The M4P_SRAM_PEADDR_CORE.STAT bit indicates either: 0 = no parity error detected on this interface. 1 = one or more parity errors detected. Cleared by writing a 1. If M4P_SRAM_PEADDR_CORE.STAT is 1 and M4P_SRAM_CFG.PERRCORE is set, then the SRAM_PEIRQ_CORE interrupt is asserted to the M4 Cortex. If M4P_SRAM_PEADDR_CORE.STAT is 1 and another parity error is detected on the same interface, then M4P_SRAM_PEADDR_CORE.MSTAT is set to 1, and the SRAM_NMI Non-Maskable Interrupt is asserted to the Cortex-M4.
30 (R/W1C)	MSTAT	Multiple Parity Error Status. The M4P_SRAM_PEADDR_CORE.MSTAT bit indicates (when =1) that multiple parity errors have been detected in the Main SRAM on transactions initiated by any of the M4 Cortex interfaces, since the time that the last such parity error (if any) was cleared by writing a 1 to M4P_SRAM_PEADDR_CORE.STAT.

Table 2-25: M4P_SRAM_PEADDR_CORE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		The M4P_SRAM_PEADDR_CORE.MSTAT bit will also be set if, on the same core clock cycle, two or more transactions initiated on the M4 Cortex interfaces detect parity errors. In that event, the M4P_SRAM_PEADDR_CORE.BUS field indicate which interface transaction is captured in the M4P_SRAM_PEADDR_CORE register.	
29:28 (R/NW)	BUS	Party Error Bus Interface. The M4P_SRAM_PEADDR_CORE.BUS bits indicates which M4 Cortex bus interface initiated the transaction which detected a parity error.	
		0	Parity Error From I-CODE interface
		1	Parity Error From D-Code Interface
		2	Parity Error from SYS interface
27:24 (R/NW)	BMSK	Parity Error Byte Mask. The M4P_SRAM_PEADDR_CORE.BMSK bits are a 4-bit mask of the bytes with parity errors in the 32-bit location at the address in M4P_SRAM_PEADDR_CORE.ADDR. For each bit, 1= parity error detected, 0 = no parity error detected.	
19:2 (R/NW)	ADDR	Parity Error Address. The M4P_SRAM_PEADDR_CORE.ADDR bits are address bits [19:2] of the location in Main SRAM containing the byte(s) with detected parity errors.	

SRAM Parity Error Address (DMA) Register

The M4P_SRAM_PEADDR_DMA register displays the status of parity errors detected in Main SRAM resulting from transactions initiated by the DMA interface. The register captures the address and byte mask of the first parity error(s) detected in any aligned 32-bit word since the last time the corresponding parity error status was cleared (by writing a 1 to M4P_SRAM_PEADDR_DMA.STAT.) If DMA parity error interrupts are enabled by M4P_SRAM_CFG.PERRDMA, and a parity error is detected on the DMA interface, then an SRAM_PEIRQ_CODE interrupt will be asserted. If another DMA parity error is detected while a first DMA parity error is still pending, then the M4P_SRAM_PEADDR_DMA.MSTAT bit is asserted, and the SRAM Parity Error Non-Maskable Interrupt (NMI) is asserted.

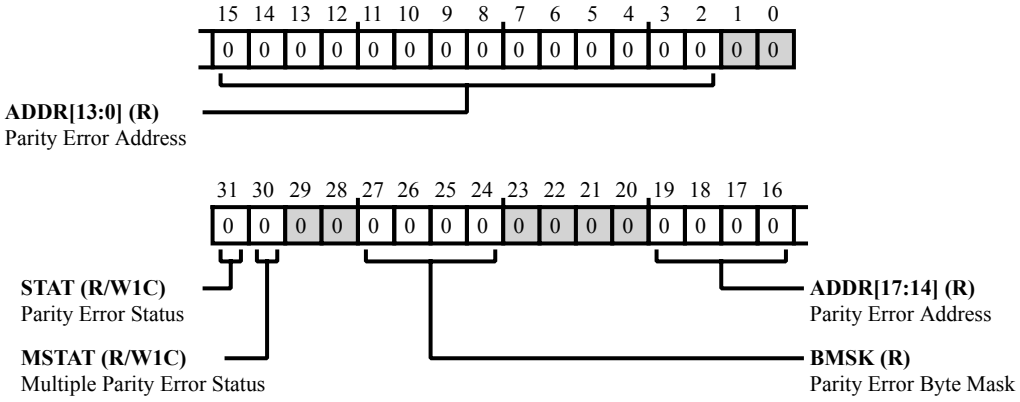

Figure 2-23: M4P_SRAM_PEADDR_DMA Register Diagram

Table 2-26: M4P_SRAM_PEADDR_DMA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	STAT	Parity Error Status. The M4P_SRAM_PEADDR_DMA.STAT bit indicates either: 0 = no parity error detected on this interface. 1 = one or more parity errors detected. Cleared by writing a 1. If M4P_SRAM_PEADDR_DMA.STAT is 1 and M4P_SRAM_CFG.PERRDMA is set, then the SRAM_PEIRQ_DMA interrupt is asserted to the M4 Cortex. If M4P_SRAM_PEADDR_DMA.STAT is 1 and another parity error is detected on the same interface, then M4P_SRAM_PEADDR_DMA.MSTAT is set to 1, and the SRAM_NMI Non-Maskable Interrupt is asserted to the M4 Cortex.
30 (R/W1C)	MSTAT	Multiple Parity Error Status. The M4P_SRAM_PEADDR_DMA.MSTAT bit indicates (when =1) that multiple parity errors have been detected in the Main SRAM on transactions initiated by the DMA interface, since the time that the last such parity error (if any) was cleared by writing a 1 to M4P_SRAM_PEADDR_DMA.STAT.
27:24 (R/NW)	BMSK	Parity Error Byte Mask.

Table 2-26: M4P_SRAM_PEADDR_DMA Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		The M4P_SRAM_PEADDR_DMA.BMSK bits are a 4-bit mask of the bytes with parity errors in the 32-bit location at the address in M4P_SRAM_PEADDR_DMA.ADDR. For each bit, 1= parity error detected, 0 = no parity error detected.
19:2 (R/NW)	ADDR	Parity Error Address. The M4P_SRAM_PEADDR_DMA.ADDR bits are address bits [19:2] of the location in Main SRAM containing the byte(s) with detected parity errors.

SysTick Calibration Register

The M4P_STCALIB register allows the programmer to define the calibration value for the ARM Processor's SysTick timer, according to the frequency of the installed crystal and the divisor settings of the CGU and PLL. The value programmed into this register will appear in the ARM SYST_CALIB Read-Only register.

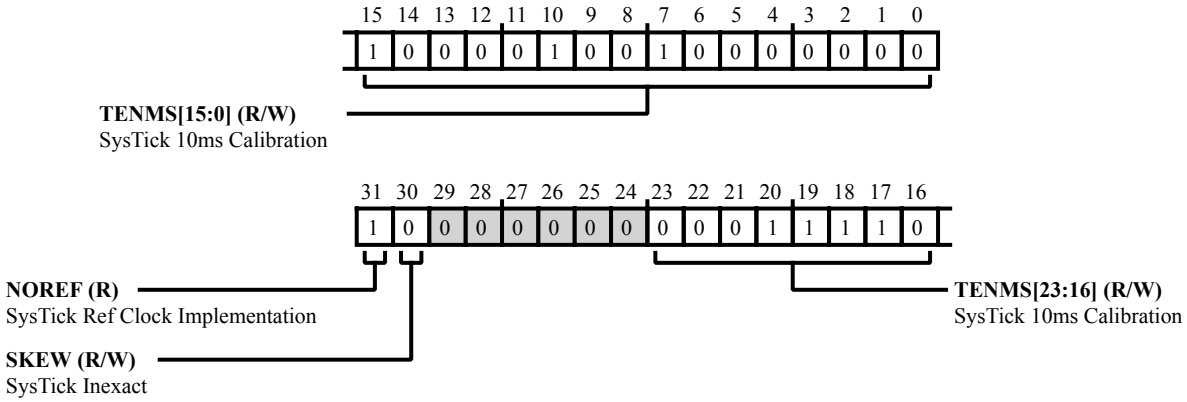

Figure 2-24: M4P_STCALIB Register Diagram

Table 2-27: M4P_STCALIB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/NW)	NOREF	SysTick Ref Clock Implementation. The M4P_STCALIB.NOREF indicates whether an external SysTick reference clock is implemented. In the ADI M4 Platform, no SysTick external reference clock is implemented. The processor internal reference clock is used for SYSTICK.	
		0	SYSTICK External Reference Clock Is Implemented
		1	SYSTICK External Reference Clock is Not Implemented
30 (R/W)	SKEW	SysTick Inexact. The M4P_STCALIB.SKEW bit is set to 1 if the calibration value specified by M4P_STCALIB.TENMS does not provide an exact multiple of 10ms. Otherwise, set this bit to 0. For example, the frequency of a 166.66 (6 repeating). MHz core clock corresponds to a TENMS value of 1,666,666.66, which is not an exact integer, so in that case M4P_STCALIB.SKEW whould be set to 1.	
23:0 (R/W)	TENMS	SysTick 10ms Calibration. The M4P_STCALIB.TENMS bit is set to an integer 24-bit value usable to compute a 10ms delay from the user-programmed frequency of the M4P core clock. For example, for a 200MHz core clock, set this value to (200MHz * 10ms) = 24'd2_000_000 = 24'h1e_8480.	

3 System Crossbars (SCB)

Modern Systems-on-Chip (SoCs) contain multi-cores, memory controllers, security blocks, Ethernet controllers and other high speed peripherals. As system integration increases, SoCs need to provide bus connectivity that allows better throughput to reduce performance bottlenecks. While traditional point-to-point connection buses have performed well in smaller systems, there is a need to use advanced switching based bus architectures for efficient handling of data transfer between multiplicity of data sources and sinks in the system. Additionally, mixing various traffic types in the same SoC (for example control, communication/Ethernet/USB and computing) while sharing the same bus resources, create different requirements from the Quality of Service (QoS) point of view.

The system crossbars (SCB) are the fundamental building blocks of a switch-fabric style for (on-chip) system bus interconnection. The SCBs connect system bus masters to system bus slaves, providing concurrent data transfer between multiple bus masters and multiple bus slaves. The SCB architecture addresses the challenges described above--providing sustainable throughput for simultaneous transactions in the system with configurable Quality of Service for each type of transaction (traffic) as required. A hierarchical model, built from multiple SCBs, provides a power and area efficient system interconnect, which satisfies the performance and flexibility requirements of a specific system.

SCB Features

The SCBs provide the following features:

- Efficient, pipelined bus transfer protocol for sustained throughput
- Full-duplex bus operation for flexibility and reduced latency
- Concurrent bus transfer support to allow multiple bus masters to access bus slaves simultaneously
- Protection model (privileged or secure) support for selective bus interconnect protection
- QoS based arbitration model
- Programmable quality of service

SCB Functional Description

The following sections provide a functional description of the SCB.

Architectural Concepts

This section describes the components of the SCB and the modules connected to it. The basic elements in the SCB are SCB masters, slaves master interfaces, and slave interfaces.

Masters

The controllers in the system that raise the data request in the form of a read/write transaction on the SCB are called masters. The system bus masters include peripheral Direct Memory Access (DMA) channels. These include the Serial Port (SPORT) DMA, and SPI DMAs, among others. Also included are the Memory-to-Memory DMA channels (MDMA), the M4 processor Core, and the L1 code fill block.

Slaves

Slaves respond to the requests raised by any SCB Master. The system bus slaves include on-chip and off-chip memory devices and controllers such as L1 SRAM, the Static Memory Controller (SMC) for SRAM and flash devices, SPI flash, and the System Memory-Mapped registers (MMRs). Each system bus slave has its own latency characteristics, operating in a given clock domain. For example, a L1 SRAM access is faster than an SMC memory accesses.

Master Interface

The master interface connects the slave to the SCB which in turn connects to the master on the other end. In effect, the master interface connects the slave to the master concealing underlying cross bar details.

Slave Interface

The slave interface provides the interface for the masters to the SCB.

SCB Block Diagrams

The SCB architectural model is illustrated in the ***SCB Overview*** figure. This figure shows a high-level overview of the SCB and associated connections to system masters and slaves. A variable number of masters connect to a variable number of slaves in each SCB. In this example, all SIs connect to all MIs as indicated by the lines connecting them.

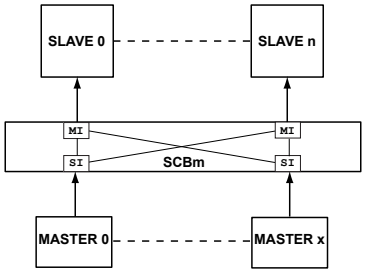

Figure 3-1: SCB Overview

Hierarchy Block Diagram

A system interconnect built from multiple SCBs in a hierarchical model is illustrated in the ***SCB Hierarchy Overview*** figure. The system master node level SCBs master connects multiple SIs to a single MI, which in turn connects to an SI of the system slave level node SCB.

As discussed above, all the masters in the system are distributed across different SCBs. A given SCB at system master node level connects directly to the system masters. These SCBs (SCB1 – SCB4) connect to SCB0 through its SIs forming a hierarchal structure. While a master has to access any slave, its first access goes through the SCB it is connected to, and then, through SCB0, reaches its intended slave.

In this example, all SIs are connected to all MIs.

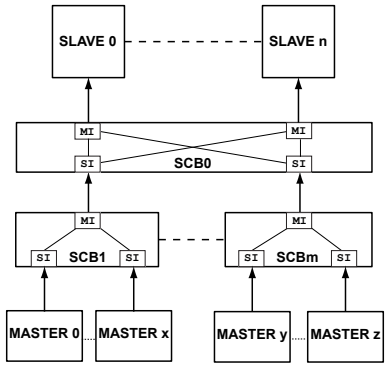

Figure 3-2: SCB Hierarchy Overview

NOTE: For an overall diagram of all SCB interconnections, see the ADSP-CM40x SCB Block Diagram.

ADSP-CM40x SCB Block Diagram

The ***ADSP-CM40x SCB Block Diagram*** figure shows the SCB block diagram for the ADSP-CM40x processors.

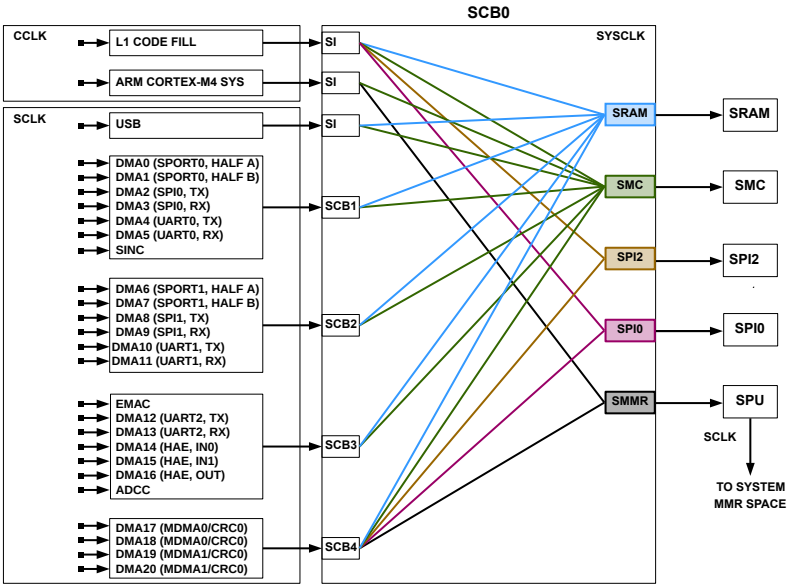

Figure 3-3: ADSP-CM40x SCB Block Diagram

While this figure is useful just for the overview it provides, it is also useful to observe the following relationships that are highlighted.

- The hierarchy of SCBs manages system bus interconnections, multiplexing, and arbitration among the cores and peripherals on the processor.
- The SCBs connections support DMA channels for some peripherals and support dedicated connections for others (such as USB). The connections also support memory-mapped register access for internal memory (L1) and for external memory (FLASH and others).
- The peripherals (and their SCBs) are in the *SCLK* clock domain. SCB0 is in the *SYSCLK* domain. The processor core is the *CCLK* clock domain. Synchronizations across clock domains affect SCB performance.
- Each peripheral has a latency for access across the SCB. The latency varies with the nature of the peripheral. Also, the number of active peripherals (especially for cases where multiple peripherals are active on a shared SCB) affects SCB performance.
- The master IDs in the figure correspond to the masters shown in the ***ADSP-CM40x SCB Bus Master IDs at SRAM, SMC, SPI1, SPI0, and SMMR*** table and the ***ADSP-CM40x SCB Register List*** table. For example, SCB_MST00 corresponds to L1CODE_FILL (L1CODE FILL), SCB_MST01 corresponds to CORTEX_SYS (ARM CORTEX-M4 SYS) and so on. When using the SWU to perform transaction monitoring, configure a particular master ID. The SWU then monitors transactions between that particular master and slave according to the configured value.

The following are definitions of acronyms that appear in the figure:

DMA0-DMA20

Indicate DMA channels for peripherals supporting DMA transfers.

SCB0-SCB4

Indicate SCB interfaces, connecting the system bus masters and slaves.

SCLK, SYSCLK, CCLK

Indicate clock domains in which the specific SCBs operate. For more information on clock domains, see the Clock Generation Unit (CGU) chapter and the product data sheet.

L1

Indicates the on-core (L1) internal memory.

C0

Indicates the processor core 0 (C0).

SMC

Indicates the static memory controller (SMC) interface.

SPORT0, SPORT1, SPORT2 - Half A/B

Indicate serial port interfaces and their full-duplex halves.

SPI0, SPI1 - RX/TX

Indicate serial peripheral interfaces ports with receive or transmit paths.

EMAC

Indicates the Ethernet MAC interface.

MDMA0, MDMA1, MDMA2, MDMA3

Indicate memory DMA 0 through 3 interfaces.

CRC0

Indicates the cyclic redundancy check (CRC) interface.

USB

Indicates the universal serial bus (USB) interface.

SMMR

Indicates the system memory-mapped register interface.

ADSP-CM40x SCB Bus Master IDs

The SCB bus master ID tables indicate which masters are connected to each of the slave ports of SCB0. The tables also indicate the precise value of the ID as seen by the slave. These values are useful for SWU programming. The 27 masters listed in the ***ADSP-CM40x Bus Master IDs at SRAM, SMC, SPI1, SPI0, and SMMR*** table correspond to the MST QOS registers mentioned in the ***ADSP-CM40x SCB Register List*** table in their correct order. For example MST00 corresponds to L1CODE_FILL, MST01 corresponds to CORTEX_SYS and so on.

NOTE: For an overall diagram of all SCB interconnections, see the ADSP-CM40x SCB Block Diagram.

Table 3-1: ADSP-CM40x Bus Master IDs at SRAM, SMC, SPI1, SPI0, and SMMR

Master	SRAM	ASYNC	SPI	MMR
L1CODE_FILL	8'b000xx000	8'b000xx000	8'b000xx000	N/A
CORTEX_SYS	N/A	8'b00000001	N/A	8'b00000001
USB	8'b00000010	8'b00000010	N/A	N/A
DMA0 (SP0A)	8'b0010x000	8'b0010x000	N/A	N/A
DMA1 (SP0B)	8'b0010x001	8'b0010x001	N/A	N/A
DMA6 (SP1A)	8'b0100x010	8'b0100x010	N/A	N/A
DMA7 (SP1B)	8'b0100x011	8'b0100x011	N/A	N/A
DMA2 (SPI0_TX)	8'b0010x010	8'b0010x010	N/A	N/A
DMA3 (SPI0_RX)	8'b0010x011	8'b0010x011	N/A	N/A
DMA8 (SPI1_TX)	8'b0100x100	8'b0100x100	N/A	N/A
DMA9 (SPI1_RX)	8'b0100x101	8'b0100x101	N/A	N/A
DMA4 (UART0_TX)	8'b0010x100	8'b0010x100	N/A	N/A
DMA5 (UART0_RX)	8'b0010x101	8'b0010x101	N/A	N/A
DMA10 (UART1_TX)	8'b0100x000	8'b0100x000	N/A	N/A
DMA11 (UART1_RX)	8'b0100x001	8'b0100x001	N/A	N/A
DMA12 (UART2_TX)	8'b0110x101	8'b0110x101	N/A	N/A
DMA13 (UART2_RX)	8'b0110x110	8'b0110x110	N/A	N/A
SINC	8'b0010x110	8'b0010x110	N/A	N/A
EMAC	8'b0110x000	8'b0110x000	N/A	N/A
ADCC	8'b0110x010	8'b0110x010	N/A	N/A
DMA14 (HAE_IN0)	8'b0110x001	8'b0110x001	N/A	N/A
DMA15 (HAE_IN1)	8'b0110x011	8'b0110x011	N/A	N/A

Table 3-1: ADSP-CM40x Bus Master IDs at SRAM, SMC, SPI1, SPI0, and SMMR (Continued)

Master	SRAM	ASYNC	SPI	MMR
DMA16 (HAE_OUT)	8'b0110x100	8'b0110x100	N/A	N/A
DMA17 (MDMA0_RD)	8'b1000x001	8'b1000x001	8'b1000x001	8'b1000x001
DMA18 (MDMA0_WR)	8'b1000x000	8'b1000x000	8'b1000x000	8'b1000x000
DMA19 (MDMA1_RD)	8'b1000x011	8'b1000x011	8'b1000x011	8'b1000x011
DMA20 (MDMA1_WR)	8'b1000x010	8'b1000x010	8'b1000x010	8'b1000x010

System Crossbars

Describes the system crossbar functionality.

The System Crossbars (SCB) are the fundamental building blocks of the system bus interconnect. The SCB (often referred to as the system interconnect fabric), is a collection of inter-connection units connecting system masters to slave memory spaces. The SCB connects one or more master devices to one or more memory-mapped slave devices. Each connected master can be a core that originates an SCB transaction, or a master interface of an upstream SCB cascaded interconnect. Each connected slave can be the final target of an SCB transaction or a slave interface of a downstream cascaded SCB interconnect (forming a hierarchy of SCBs).

Each SCB that has multiple masters and slaves share the total bandwidth of the SCB. (In a M:N configuration where M masters are connected to N slaves through the SCBx.)

The SCB provides separate channels for reads and writes. Read and write accesses through a given SCB do not share bandwidth. All the SCBs are 32 bits wide and run at *SCLK* speed, and can provide a bandwidth of up to 400 Mbps for reads and writes separately (when *SCLK* = 100 MHz). Only SCB0, which is the major SCB in the SCB hierarchy, has the multiple paths between multiple master and slave interfaces.

See ADSP-CM40x SCB Block Diagram.

All other SCBs in the chip connect to SCB0 through different slave interfaces. Other primary masters (DMAs, cores, and so on) in the system are distributed across these small SCBs. For a given SCB, all the master and slaves share the total bandwidth of the SCB. (Only SCB0 is the exception). Since different DMA channels are scattered across different SCBs (SCB1, SCB2 SCB3, and so on), they do not conflict for the bandwidth as long as they are in different SCB and are accessing different slaves. SCB0 allows for concurrent data transfer between multiple bus masters and multiple bus slaves, providing flexibility, and full-duplex operation. For example, the data transfer between SCB4 (one of the MDMA channels), and SMC controller (accessing SRAM memory) can happen in parallel to SCB2 (SPI RX/TX DMA) accessing memory mapped SPI memory with both the transfers occurring at up to 400 MBPS. So if system accesses are carefully architected, SCB has a potential of providing sufficient sustained bandwidth in the end system.

Since the SCBs support burst transfers, it is important to configure the requesting master appropriately to make best use of available SCB bandwidth. For a DMA master, choosing the appropriate `DMA_CFG.MSIZE` value, is important from both a functional and a performance perspective. The value in the `DMA_CFG.PSIZE` bit field determines the

width of the peripheral bus in use. It can be configured to 1-byte, 2-bytes, or 4-bytes. The DMA_CFG.MSIZE value determines the actual size of the SCB bus in use. It also determines the minimum number of bytes which are transferred from or to memory corresponding to a single DMA request or grant. The transfer can be 1-, 2-, 4-, 8-, 16-, or 32-bytes. If the DMA_CFG.MSIZE value is greater than the SCB bus width, the SCB performs burst transfers according to the width defined in DMA_CFG.MSIZE. When DMA_CFG.MSIZE is less than the SCB bus width, bursting is not supported and partial bus use results.

Each of the SCB unit in the fabric consists of N Slave interfaces (MSTn). Each of these interfaces has controls for read quality of service, write quality of service, and functional mode. A subset of these matrices includes controls for IB(Interface Block) sync mode, and bus functional mode. For more details on IB, see the clock domain synchronization section.

ADSP-CM40x SCB Arbitration

The SCB uses QoS based arbitration to prioritize each slave-to-master interface (master interface) and each master-to-slave interface (slave interface). Each slave has a quality of service (QoS) programmable feature that affects arbitration. This allows to increase the priority of a specific master and allow the chances of it getting higher priority when multiple masters are requesting over same SCB.

The slave interface of the crossbar (where the masters such as DMA connect to) performs few functions and arbitration is one of them. There are programmable register to decide the QoS for each master. The programmable QoS registers can be viewed as being associated with SCBx. There is separate Quality of service control for read (RQoS) and write (WQoS). For example, the QoS registers for DMA0-5, SINC can be viewed as residing in SCB4. Whenever a transaction is received at DMA0, the programmed QoS value is associated with that transaction and is arbitrated with the rest of the masters at SCB4. Consider the following situation.

- At SCB1, masters (1, 2, and 3) have RQOS values of (6, 4, and 2, respectively).
- At SCB2, masters (4, 5, and 6) have RQOS values of (12, 13, and 1, respectively).
- Master 1 wins at SCB1, and master 5 wins at SCB2, as defined by each having the largest RQOS value for its respective group.
- In a perfect competition at SCB0 masters 4 and 5 had the highest overall RQOS values, and they compete for arbitration directly at SCB0. Because of the mini-SCBs, master 1 which has a much lower RQOS value, is able to win against master 4 and make it all the way to SCB0.

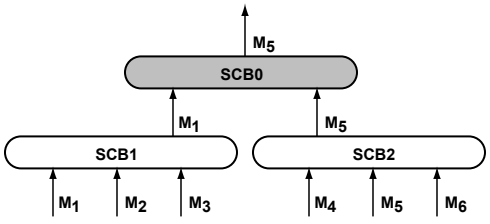

Figure 3-4: SCB Arbitration

NOTE: In most cases, the default RQOS/WQOS register values should be sufficient to achieve optimum throughput. For more information, see ADSP-CM40x SCB Programming Model

Table 3-2: ADSP-CM40x Bus Master Reset QoS Values

Master ID	Master	read_qos Reset Value	write_qos Reset value
0	L1CODE_FILL	1	1
1	CORTEX_SYS	1	1
2	USB	1	1
3	DMA0 (SP0A)	1	1
4	DMA1 (SP0B)	1	1
5	DMA6 (SP1A)	1	1
6	DMA7 (SP1B)	1	1
7	DMA2 (SPI0_TX)	1	1
8	DMA3 (SPI0_RX)	1	1
9	DMA8 (SPI1_TX)	1	1
10	DMA9 (SPI1_RX)	1	1
11	DMA4 (UART0_TX)	1	1
12	DMA5 (UART0_RX)	1	1
13	DMA10 (UART1_TX)	1	1
14	DMA11(UART1_RX)	1	1
15	DMA12 (UART2_TX)	1	1
16	DMA13 (UART2_RX)	1	1
17	DMA20 (MDMA1_WR)	0	0
18	DMA19 (MDMA1_RD)	0	0
19	SINC	1	1
20	EMAC	1	1
21	ADCC	1	1
22	DMA14 (HAE_IN0)	1	1
23	DMA15 (HAE_IN1)	1	1
24	DMA16 (HAE_OUT)	1	1
25	DMA18(MDMA0_WR)	0	0
26	DMA17 (MDMA0_RD)	0	0

Clock Domain Synchronization

Most of the masters in the system operate at the same clock as the SCB, which is *SCLK*. So, there is no need to synchronize. The M4 core and L1 code fill blocks (used for cache fill from SPI flash memory or SMC memory) operate in *CCLK* domain. The SCB provides the option to program the different synchronization schemes for these masters through Master0 and Master1 IB Sync Mode registers (SCB_MST00_IB_SYNC, SCB_MST01_IB_SYNC).

These registers perform clock domain crossing synchronization from *CCLK* to *SCLK*. The configuration of these registers depends on *CCLK* and *SCLK* configuration.

ADSP-CM40x SCB Register List

The system cross bar (SCB), which is often referred to as the system interconnect fabric, is a collection of interconnection units connecting system masters to slave memory spaces. Each unit in the fabric consists of a matrix of master interfaces (MSTn). Each of these matrices has a controls for read quality of service, write quality of service, and functional mode. A subset of these matrices include controls for IB sync mode and bus functional mode. A set of registers govern SCB operations. For more information on SCB functionality, see the SCB register descriptions.

Table 3-3: ADSP-CM40x SCB Register List

Name	Description
SCB_MST00_IB_RQOS	Master 0 Read Quality of Service
SCB_MST00_IB_SYNC	Master 0 IB Sync Mode
SCB_MST00_IB_WQOS	Master 0 Write Quality of Service
SCB_MST01_IB_RQOS	Master 1 Read Quality of Service
SCB_MST01_IB_SYNC	Master 1 IB Sync Mode
SCB_MST01_IB_WQOS	Master 1 Write Quality of Service
SCB_MST02_RQOS	Master 2 Read Quality of Service
SCB_MST02_WQOS	Master 2 Write Quality of Service
SCB_MST03_RQOS	Master 3 Read Quality of Service
SCB_MST03_WQOS	Master 3 Write Quality of Service
SCB_MST04_RQOS	Master 4 Read Quality of Service
SCB_MST04_WQOS	Master 4 Write Quality of Service
SCB_MST05_RQOS	Master 5 Read Quality of Service
SCB_MST05_WQOS	Master 5 Write Quality of Service
SCB_MST06_RQOS	Master 6 Read Quality of Service
SCB_MST06_WQOS	Master 6 Write Quality of Service
SCB_MST07_RQOS	Master 7 Read Quality of Service
SCB_MST07_WQOS	Master 7 Write Quality of Service

Table 3-3: ADSP-CM40x SCB Register List (Continued)

Name	Description
SCB_MST08_RQOS	Master 8 Read Quality of Service
SCB_MST08_WQOS	Master 8 Write Quality of Service
SCB_MST09_RQOS	Master 9 Read Quality of Service
SCB_MST09_WQOS	Master 9 Write Quality of Service
SCB_MST10_RQOS	Master 10 Read Quality of Service
SCB_MST10_WQOS	Master 10 Write Quality of Service
SCB_MST11_RQOS	Master 11 Read Quality of Service
SCB_MST11_WQOS	Master 11 Write Quality of Service
SCB_MST12_RQOS	Master 12 Read Quality of Service
SCB_MST12_WQOS	Master 12 Write Quality of Service
SCB_MST13_RQOS	Master 13 Read Quality of Service
SCB_MST13_WQOS	Master 13 Write Quality of Service
SCB_MST14_RQOS	Master 14 Read Quality of Service
SCB_MST14_WQOS	Master 14 Write Quality of Service
SCB_MST15_RQOS	Master 15 Read Quality of Service
SCB_MST15_WQOS	Master 15 Write Quality of Service
SCB_MST16_RQOS	Master 16 Read Quality of Service
SCB_MST16_WQOS	Master 16 Write Quality of Service
SCB_MST17_RQOS	Master 17 Read Quality of Service
SCB_MST17_WQOS	Master 17 Write Quality of Service
SCB_MST18_RQOS	Master 18 Read Quality of Service
SCB_MST18_WQOS	Master 18 Write Quality of Service
SCB_MST19_RQOS	Master 19 Read Quality of Service
SCB_MST19_WQOS	Master 19 Write Quality of Service
SCB_MST20_RQOS	Master20 Read Quality of Service
SCB_MST20_WQOS	Master 20 Write Quality of Service
SCB_MST21_RQOS	Master 21 Read Quality of Service
SCB_MST21_WQOS	Master 21 Write Quality of Service
SCB_MST22_RQOS	Master 22 Read Quality of Service
SCB_MST22_WQOS	Master 22 Write Quality of Service
SCB_MST23_RQOS	Master 23 Read Quality of Service

Table 3-3: ADSP-CM40x SCB Register List (Continued)

Name	Description
SCB_MST23_WQOS	Master 23 Write Quality of Service
SCB_MST24_RQOS	Master 24 Read Quality of Service
SCB_MST24_WQOS	Master 24 Write Quality of Service
SCB_MST25_RQOS	Master 25 Read Quality of Service
SCB_MST25_WQOS	Master 25 Write Quality of Service
SCB_MST26_RQOS	Master 26 Read Quality of Service
SCB_MST26_WQOS	Master 26 Write Quality of Service

ADSP-CM40x SCB Programming Model

As previously noted, the SCB arbitration model for the master and slave SCBs is QoS based. Each slave interface has a QoS value (or priority) associated with its read and write channels. These QoS selections are 4-bit values, which are present in the read QoS register (SCB_MSTxx_IB_RQOS) and write QoS register (SCB_MSTxx_IB_WQOS) register of each SCB master.

At the entry point to the infrastructure, all transactions are allocated a programmable, local QoS value. The arbitration of the transaction throughout the infrastructure uses this QoS. At any arbitration node, a fixed priority exists for transactions with a different QoS. The highest value has the highest priority. If there are coincident transactions at an arbitration node with the same QoS value that require arbitration, then the network uses a least recently used (LRU) algorithm. At each switch, the master with the highest QoS gains access, and that switch output takes the QoS value of the winner for that transaction. At the next switch slave interface, that master uses the QoS value of the winner.

SCB fabric registers occupy 1M byte of address space. The QoS registers in this space have values from 0 (lowest priority) to 15 (highest priority).

NOTE: Because the SCB arbitration of the ADSP-CM40x processor is fixed (not programmable), these SCBs do not have slot numbers for modifying read/write arbitration settings.

FIFO Synchronization

The FIFO associated with every channel is implemented to support clock domain crossing functionality.

The synchronization scheme used in the FIFO can be changed in the FIFO Sync_mode register. By default, FIFO is a pure asynchronous FIFO. Programming can reduce register access latency while CCLK:SYSCLK frequency ratio is n:1(n integer) or 1:1. Program the FIFO mode register to the respective values in the sync mode bits of the FIFO Sync mode register as follows:

- 0-Sync 1:1
- 1-Sync n:1

- 4-async

Table 3-4: FIFO Sync Modes and Actions

Original Mode	Required Mode	Action
ASYNC	Sync 1:1 or n:1	Change the clocks, then change the register
Sync 1:1 or n:1	ASYNC	Change the register, then change the clocks

NOTE: This synchronization feature is applicable only for CCLK:SYSCLK ratios of 1:1 and n:1, but the feature is not applicable for ratios of m:n. For example, the synchronization feature is applicable for a CCLK:SYSCLK ratio of 200 MHz:100 MHz. This feature is not applicable for a ratio of 250 MHz:100 MHz.

ADSP-CM40x SCB Programming Concepts

The SCB arbitration model is fixed (not programmable, but each slave does have a quality of service (QoS) programmable feature that affects arbitration.

Each slave interface has a QoS value (or priority) associated with its read and write channels. At the entry point to the infrastructure, all transactions are allocated this priority value.

Consider a case where a UART peripheral and an SPI peripheral (which are attached to same SCB) simultaneously make a write access to the SMC. The peripheral with a higher write QoS value programmed in its register is granted access.

It is important to note the following restrictions:

- Do not write to SPI0 and SPI2 from any of the masters. SPI cannot write to flash in memory-mapped mode, it can only read in this mode. To program the flash, configure the SPI in normal transmit mode and write through the SPI registers.
- L1CODE is read-only.
- MDMA channels are unidirectional. The mdma_rd is read-only, and the mdma_wr is write-only.
- MDMA access to MMR space is allowed only for selected peripherals (for example, CAN, TRU, and SEC).

ADSP-CM40x SCB Register Descriptions

System Cross Bar (SCB) contains the following registers.

Table 3-5: ADSP-CM40x SCB Register List

Name	Description
SCB_MST00_IB_RQOS	Master 0 Read Quality of Service
SCB_MST00_IB_SYNC	Master 0 IB Sync Mode

Table 3-5: ADSP-CM40x SCB Register List (Continued)

Name	Description
SCB_MST00_IB_WQOS	Master 0 Write Quality of Service
SCB_MST01_IB_RQOS	Master 1 Read Quality of Service
SCB_MST01_IB_SYNC	Master 1 IB Sync Mode
SCB_MST01_IB_WQOS	Master 1 Write Quality of Service
SCB_MST02_RQOS	Master 2 Read Quality of Service
SCB_MST02_WQOS	Master 2 Write Quality of Service
SCB_MST03_RQOS	Master 3 Read Quality of Service
SCB_MST03_WQOS	Master 3 Write Quality of Service
SCB_MST04_RQOS	Master 4 Read Quality of Service
SCB_MST04_WQOS	Master 4 Write Quality of Service
SCB_MST05_RQOS	Master 5 Read Quality of Service
SCB_MST05_WQOS	Master 5 Write Quality of Service
SCB_MST06_RQOS	Master 6 Read Quality of Service
SCB_MST06_WQOS	Master 6 Write Quality of Service
SCB_MST07_RQOS	Master 7 Read Quality of Service
SCB_MST07_WQOS	Master 7 Write Quality of Service
SCB_MST08_RQOS	Master 8 Read Quality of Service
SCB_MST08_WQOS	Master 8 Write Quality of Service
SCB_MST09_RQOS	Master 9 Read Quality of Service
SCB_MST09_WQOS	Master 9 Write Quality of Service
SCB_MST10_RQOS	Master 10 Read Quality of Service
SCB_MST10_WQOS	Master 10 Write Quality of Service
SCB_MST11_RQOS	Master 11 Read Quality of Service
SCB_MST11_WQOS	Master 11 Write Quality of Service
SCB_MST12_RQOS	Master 12 Read Quality of Service
SCB_MST12_WQOS	Master 12 Write Quality of Service
SCB_MST13_RQOS	Master 13 Read Quality of Service
SCB_MST13_WQOS	Master 13 Write Quality of Service
SCB_MST14_RQOS	Master 14 Read Quality of Service
SCB_MST14_WQOS	Master 14 Write Quality of Service
SCB_MST15_RQOS	Master 15 Read Quality of Service

Table 3-5: ADSP-CM40x SCB Register List (Continued)

Name	Description
SCB_MST15_WQOS	Master 15 Write Quality of Service
SCB_MST16_RQOS	Master 16 Read Quality of Service
SCB_MST16_WQOS	Master 16 Write Quality of Service
SCB_MST17_RQOS	Master 17 Read Quality of Service
SCB_MST17_WQOS	Master 17 Write Quality of Service
SCB_MST18_RQOS	Master 18 Read Quality of Service
SCB_MST18_WQOS	Master 18 Write Quality of Service
SCB_MST19_RQOS	Master 19 Read Quality of Service
SCB_MST19_WQOS	Master 19 Write Quality of Service
SCB_MST20_RQOS	Master20 Read Quality of Service
SCB_MST20_WQOS	Master 20 Write Quality of Service
SCB_MST21_RQOS	Master 21 Read Quality of Service
SCB_MST21_WQOS	Master 21 Write Quality of Service
SCB_MST22_RQOS	Master 22 Read Quality of Service
SCB_MST22_WQOS	Master 22 Write Quality of Service
SCB_MST23_RQOS	Master 23 Read Quality of Service
SCB_MST23_WQOS	Master 23 Write Quality of Service
SCB_MST24_RQOS	Master 24 Read Quality of Service
SCB_MST24_WQOS	Master 24 Write Quality of Service
SCB_MST25_RQOS	Master 25 Read Quality of Service
SCB_MST25_WQOS	Master 25 Write Quality of Service
SCB_MST26_RQOS	Master 26 Read Quality of Service
SCB_MST26_WQOS	Master 26 Write Quality of Service

Master 0 Read Quality of Service

The SCB_MST00_IB_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

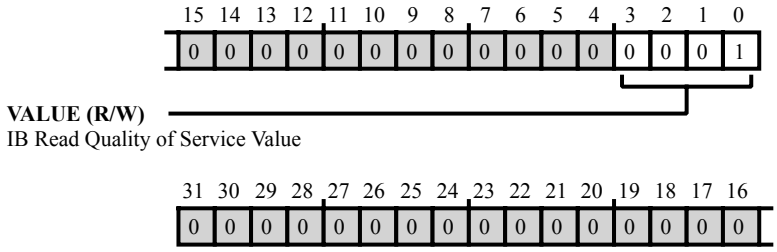

Figure 3-5: SCB_MST00_IB_RQOS Register Diagram

Table 3-6: SCB_MST00_IB_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	IB Read Quality of Service Value. The SCB_MST00_IB_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 0 IB Sync Mode

The SCB_MST00_IB_SYNC register changes the synchronization scheme used in the FIFO. By default, FIFO is a pure asynchronous FIFO. If the user wishes to reduce register access latency while CCLK:SCLK frequency ratio is n: 1 (n is an integer) or 1:1. The FIFO mode register can be programmed to the respective values.

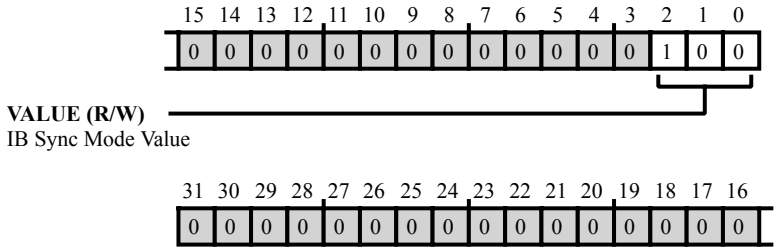

Figure 3-6: SCB_MST00_IB_SYNC Register Diagram

Table 3-7: SCB_MST00_IB_SYNC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2:0 (R/W)	VALUE	IB Sync Mode Value. The SCB_MST00_IB_SYNC.VALUE bits select the sync/async mode. All enumeration values not shown are reserved.	
		0	Sync 1:1
		1	sync n:1
		4	Async

Master 0 Write Quality of Service

The SCB_MST00_IB_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

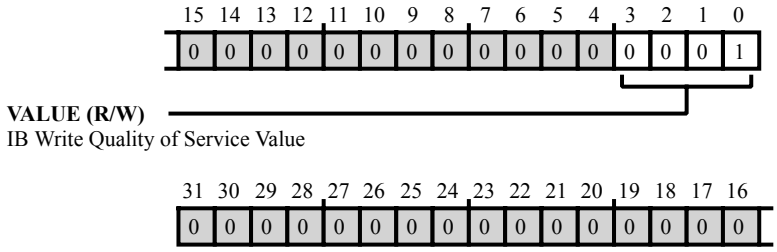

Figure 3-7: SCB_MST00_IB_WQOS Register Diagram

Table 3-8: SCB_MST00_IB_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	IB Write Quality of Service Value. The SCB_MST00_IB_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 1 Read Quality of Service

The SCB_MST01_IB_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

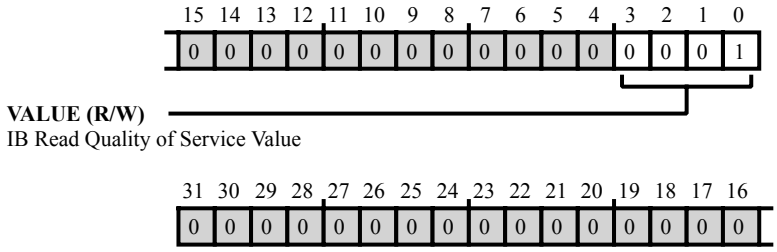

Figure 3-8: SCB_MST01_IB_RQOS Register Diagram

Table 3-9: SCB_MST01_IB_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	IB Read Quality of Service Value. The SCB_MST01_IB_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 1 IB Sync Mode

The SCB_MST01_IB_SYNC register changes the synchronization scheme used in the FIFO. By default, FIFO is a pure asynchronous FIFO. If the user wishes to reduce register access latency while CCLK:SCLK frequency ratio is n:1 (n is an integer) or 1:1. The FIFO mode register can be programmed to the respective values.

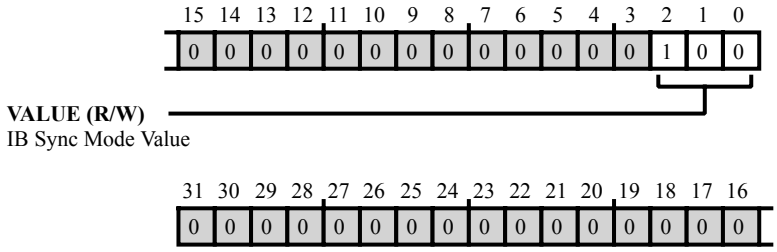

Figure 3-9: SCB_MST01_IB_SYNC Register Diagram

Table 3-10: SCB_MST01_IB_SYNC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2:0 (R/W)	VALUE	IB Sync Mode Value. The SCB_MST01_IB_SYNC.VALUE bits select the sync/async mode. All enumeration values not shown are reserved.	
		0	Sync 1:1
		1	Sync n:1
		4	Async

Master 1 Write Quality of Service

The SCB_MST01_IB_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

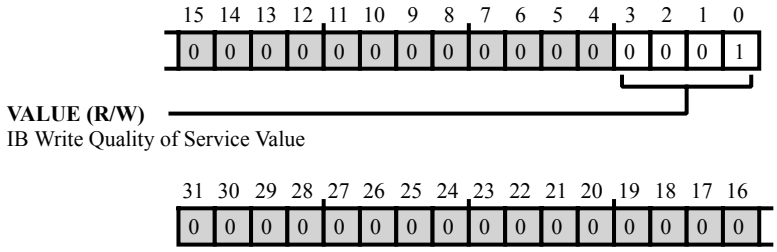

Figure 3-10: SCB_MST01_IB_WQOS Register Diagram

Table 3-11: SCB_MST01_IB_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	IB Write Quality of Service Value. The SCB_MST01_IB_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 2 Read Quality of Service

The SCB_MST02_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

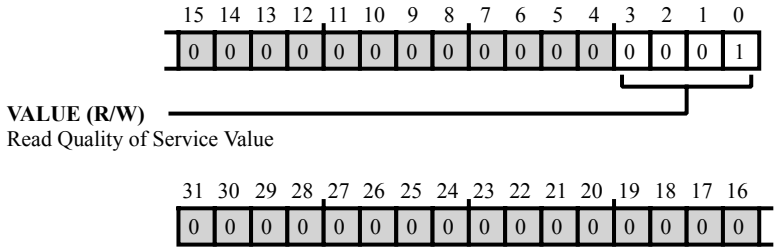

Figure 3-11: SCB_MST02_RQOS Register Diagram

Table 3-12: SCB_MST02_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST02_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 2 Write Quality of Service

The SCB_MST02_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

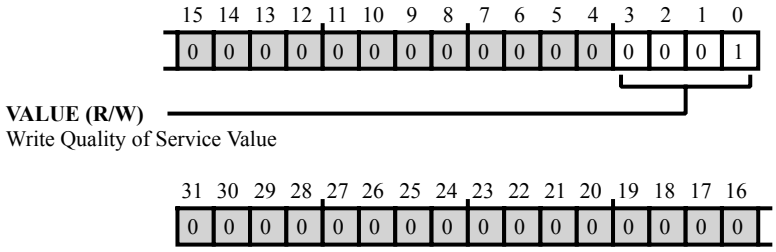

Figure 3-12: SCB_MST02_WQOS Register Diagram

Table 3-13: SCB_MST02_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST02_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 3 Read Quality of Service

The SCB_MST03_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

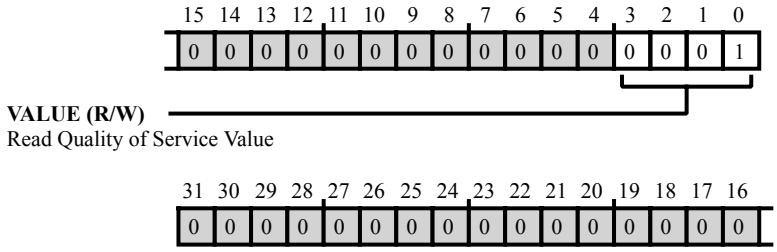

Figure 3-13: SCB_MST03_RQOS Register Diagram

Table 3-14: SCB_MST03_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST03_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 3 Write Quality of Service

The SCB_MST03_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

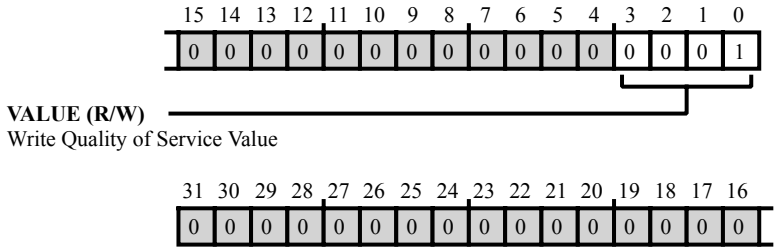

Figure 3-14: SCB_MST03_WQOS Register Diagram

Table 3-15: SCB_MST03_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST03_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 4 Read Quality of Service

The SCB_MST04_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

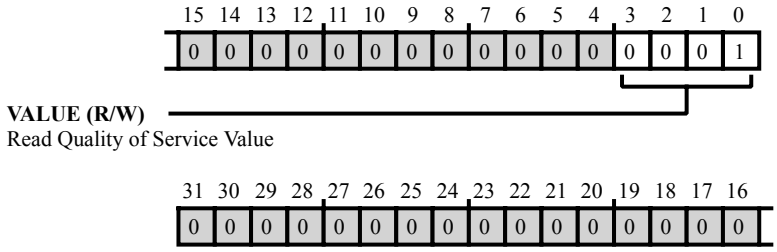

Figure 3-15: SCB_MST04_RQOS Register Diagram

Table 3-16: SCB_MST04_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST04_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 4 Write Quality of Service

The SCB_MST04_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

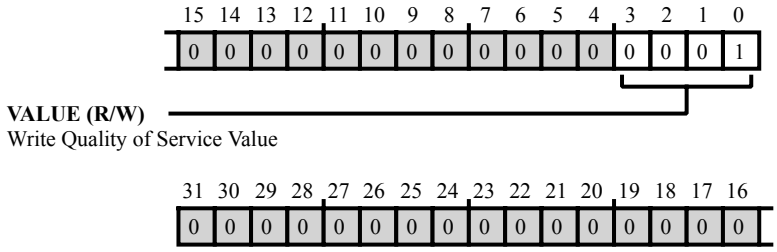

Figure 3-16: SCB_MST04_WQOS Register Diagram

Table 3-17: SCB_MST04_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST04_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 5 Read Quality of Service

The SCB_MST05_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

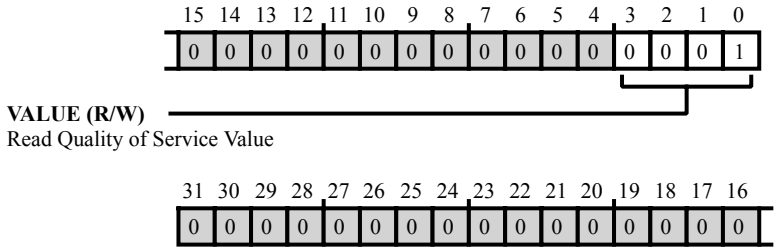

Figure 3-17: SCB_MST05_RQOS Register Diagram

Table 3-18: SCB_MST05_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST05_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 5 Write Quality of Service

The SCB_MST05_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

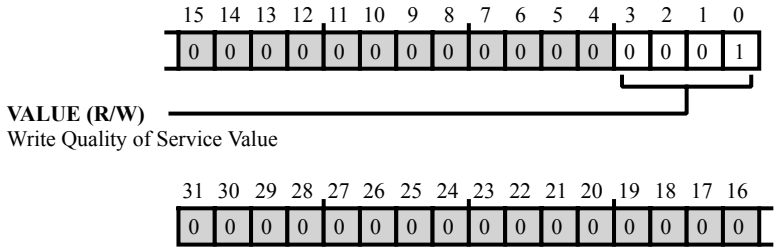

Figure 3-18: SCB_MST05_WQOS Register Diagram

Table 3-19: SCB_MST05_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST05_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 6 Read Quality of Service

The SCB_MST06_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

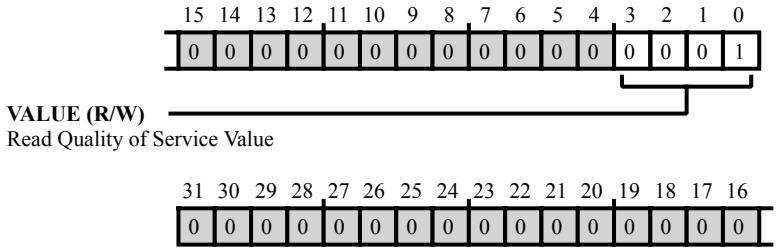

Figure 3-19: SCB_MST06_RQOS Register Diagram

Table 3-20: SCB_MST06_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST06_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 6 Write Quality of Service

The SCB_MST06_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

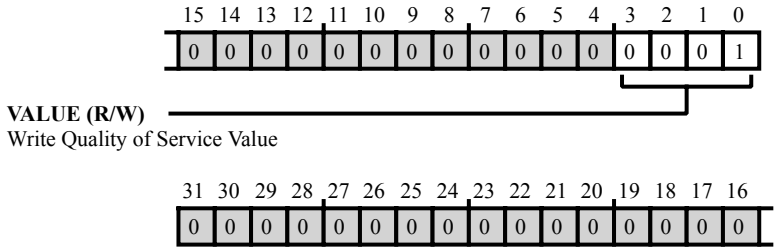

Figure 3-20: SCB_MST06_WQOS Register Diagram

Table 3-21: SCB_MST06_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST06_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 7 Read Quality of Service

The SCB_MST07_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

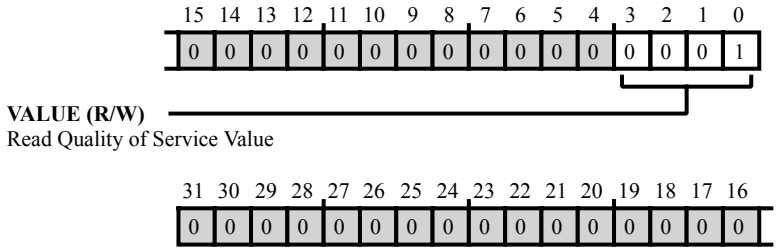

Figure 3-21: SCB_MST07_RQOS Register Diagram

Table 3-22: SCB_MST07_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST07_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 7 Write Quality of Service

The SCB_MST07_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

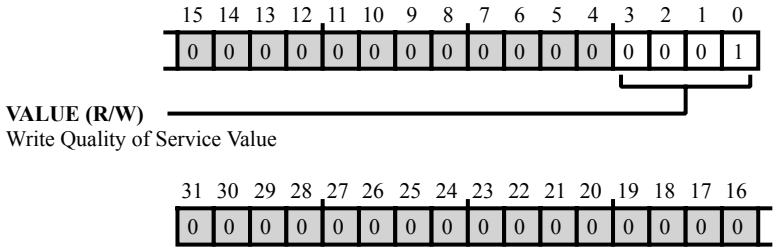

Figure 3-22: SCB_MST07_WQOS Register Diagram

Table 3-23: SCB_MST07_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST07_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 8 Read Quality of Service

The SCB_MST08_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

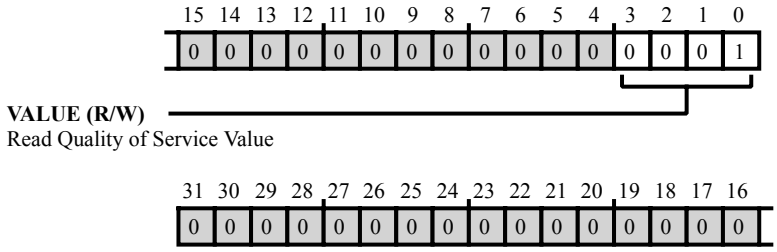

Figure 3-23: SCB_MST08_RQOS Register Diagram

Table 3-24: SCB_MST08_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST08_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 8 Write Quality of Service

The SCB_MST08_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

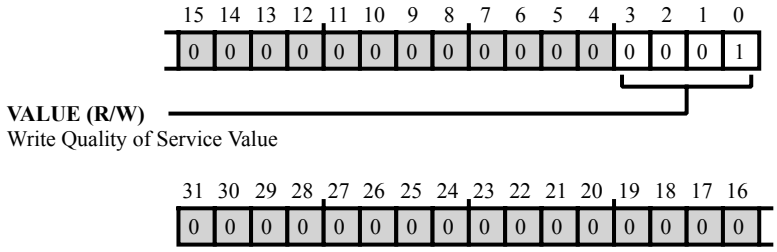

Figure 3-24: SCB_MST08_WQOS Register Diagram

Table 3-25: SCB_MST08_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST08_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 9 Read Quality of Service

The SCB_MST09_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

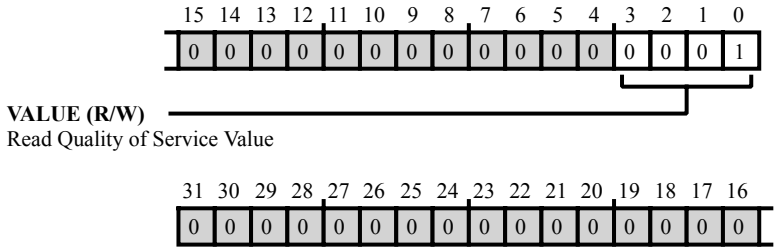

Figure 3-25: SCB_MST09_RQOS Register Diagram

Table 3-26: SCB_MST09_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST09_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 9 Write Quality of Service

The SCB_MST09_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

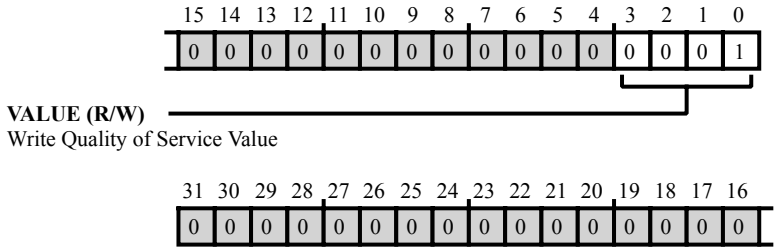

Figure 3-26: SCB_MST09_WQOS Register Diagram

Table 3-27: SCB_MST09_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST09_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 10 Read Quality of Service

The SCB_MST10_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

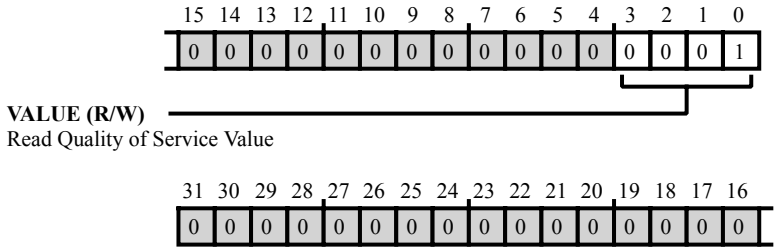

Figure 3-27: SCB_MST10_RQOS Register Diagram

Table 3-28: SCB_MST10_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST10_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 10 Write Quality of Service

The SCB_MST10_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

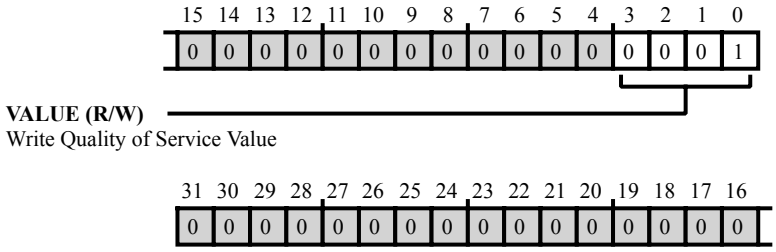

Figure 3-28: SCB_MST10_WQOS Register Diagram

Table 3-29: SCB_MST10_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST10_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 11 Read Quality of Service

The SCB_MST11_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

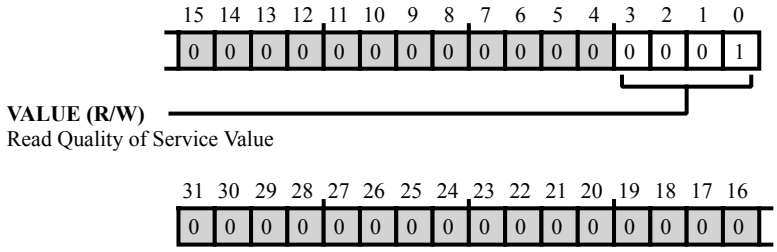

Figure 3-29: SCB_MST11_RQOS Register Diagram

Table 3-30: SCB_MST11_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST11_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 11 Write Quality of Service

The SCB_MST11_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

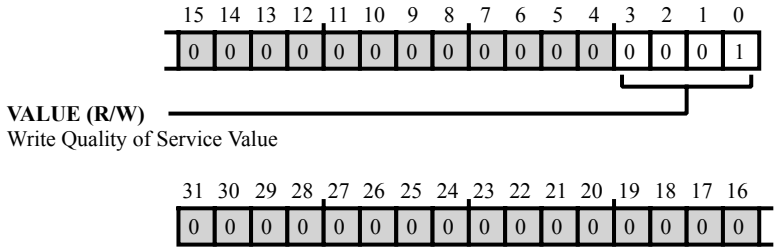

Figure 3-30: SCB_MST11_WQOS Register Diagram

Table 3-31: SCB_MST11_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST11_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 12 Read Quality of Service

The SCB_MST12_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

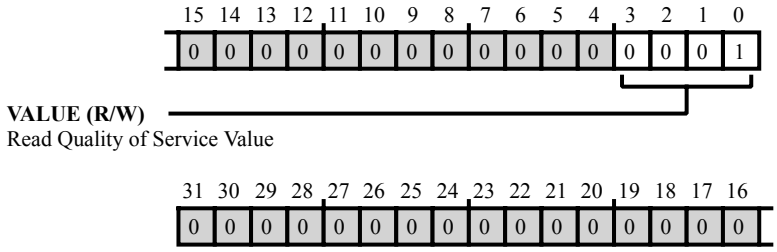

Figure 3-31: SCB_MST12_RQOS Register Diagram

Table 3-32: SCB_MST12_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST12_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 12 Write Quality of Service

The SCB_MST12_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

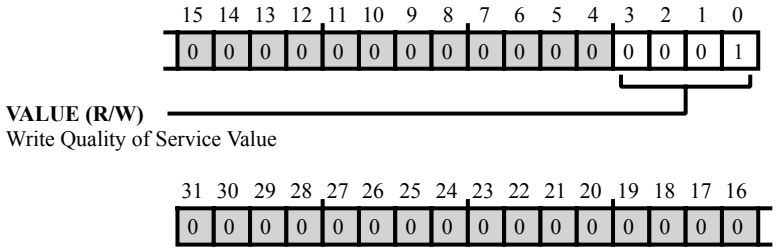

Figure 3-32: SCB_MST12_WQOS Register Diagram

Table 3-33: SCB_MST12_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST12_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 13 Read Quality of Service

The SCB_MST13_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

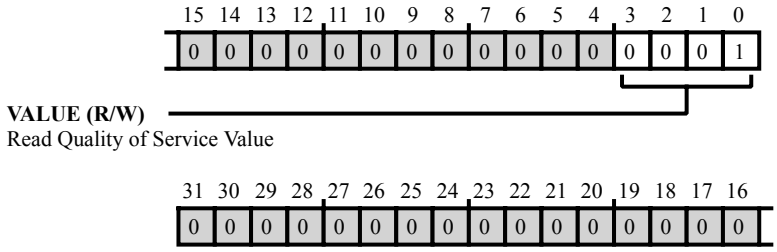

Figure 3-33: SCB_MST13_RQOS Register Diagram

Table 3-34: SCB_MST13_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST13_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 13 Write Quality of Service

The SCB_MST13_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

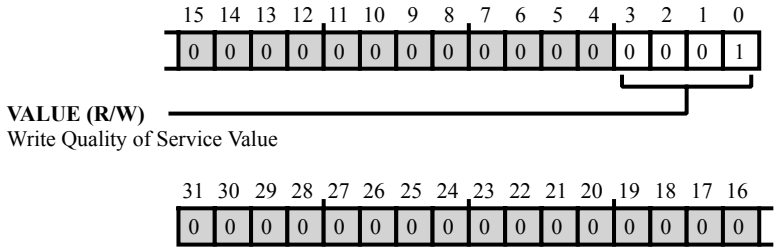

Figure 3-34: SCB_MST13_WQOS Register Diagram

Table 3-35: SCB_MST13_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST13_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 14 Read Quality of Service

The SCB_MST14_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

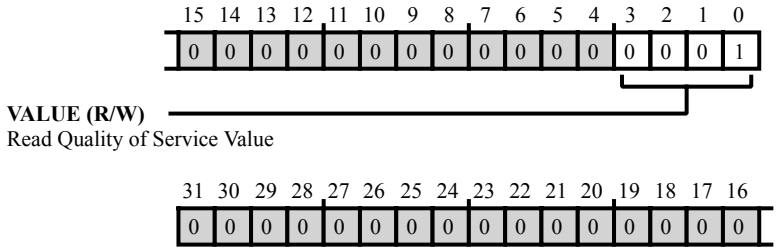

Figure 3-35: SCB_MST14_RQOS Register Diagram

Table 3-36: SCB_MST14_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST14_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 14 Write Quality of Service

The SCB_MST14_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

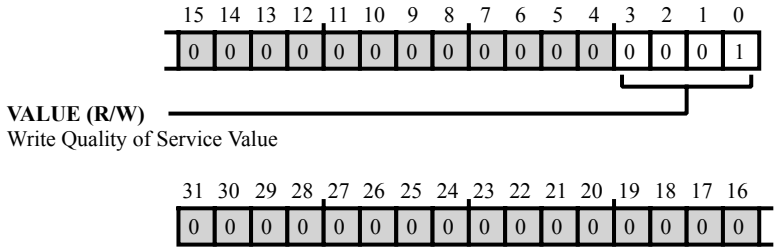

Figure 3-36: SCB_MST14_WQOS Register Diagram

Table 3-37: SCB_MST14_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST14_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 15 Read Quality of Service

The SCB_MST15_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

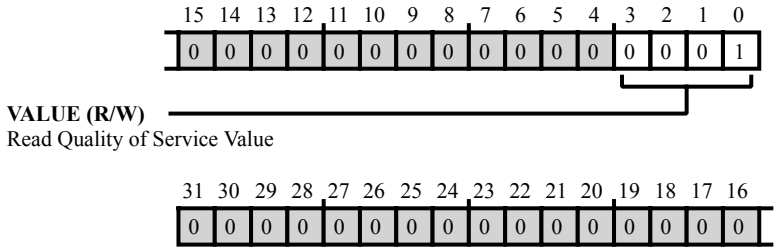

Figure 3-37: SCB_MST15_RQOS Register Diagram

Table 3-38: SCB_MST15_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST15_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 15 Write Quality of Service

The SCB_MST15_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

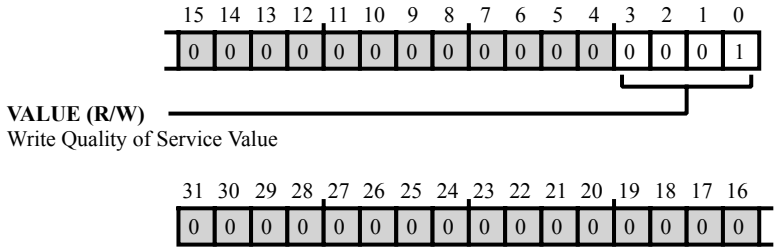

Figure 3-38: SCB_MST15_WQOS Register Diagram

Table 3-39: SCB_MST15_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST15_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 16 Read Quality of Service

The SCB_MST16_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

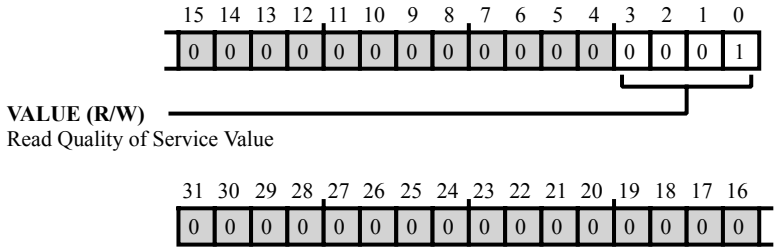

Figure 3-39: SCB_MST16_RQOS Register Diagram

Table 3-40: SCB_MST16_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST16_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 16 Write Quality of Service

The SCB_MST16_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

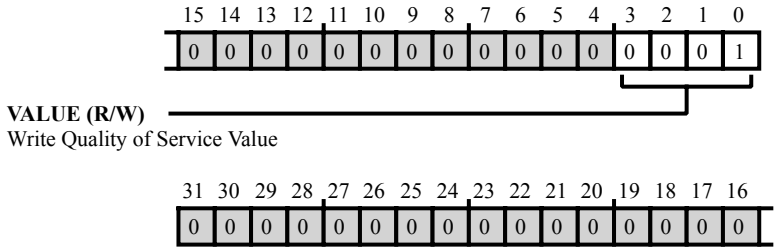

Figure 3-40: SCB_MST16_WQOS Register Diagram

Table 3-41: SCB_MST16_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST16_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 17 Read Quality of Service

The SCB_MST17_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

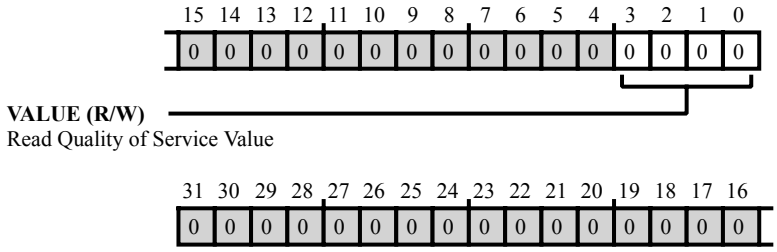

Figure 3-41: SCB_MST17_RQOS Register Diagram

Table 3-42: SCB_MST17_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST17_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 17 Write Quality of Service

The SCB_MST17_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

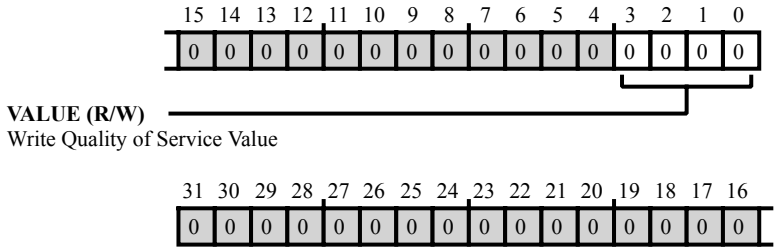

Figure 3-42: SCB_MST17_WQOS Register Diagram

Table 3-43: SCB_MST17_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST17_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 18 Read Quality of Service

The SCB_MST18_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

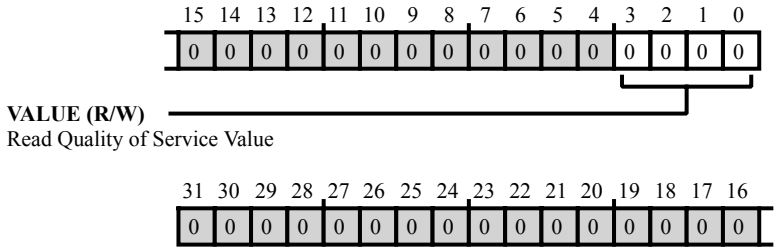

Figure 3-43: SCB_MST18_RQOS Register Diagram

Table 3-44: SCB_MST18_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST18_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 18 Write Quality of Service

The SCB_MST18_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

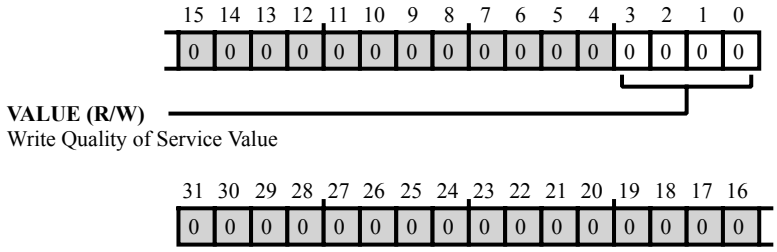

Figure 3-44: SCB_MST18_WQOS Register Diagram

Table 3-45: SCB_MST18_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST18_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 19 Read Quality of Service

The SCB_MST19_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

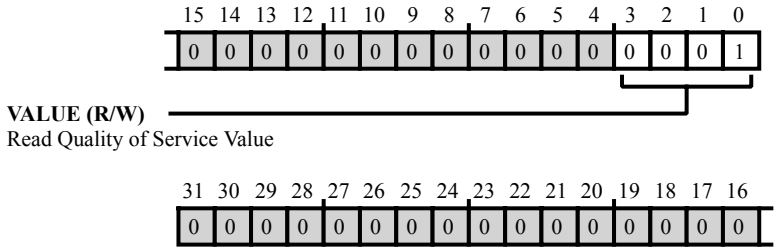

Figure 3-45: SCB_MST19_RQOS Register Diagram

Table 3-46: SCB_MST19_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST19_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 19 Write Quality of Service

The SCB_MST19_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

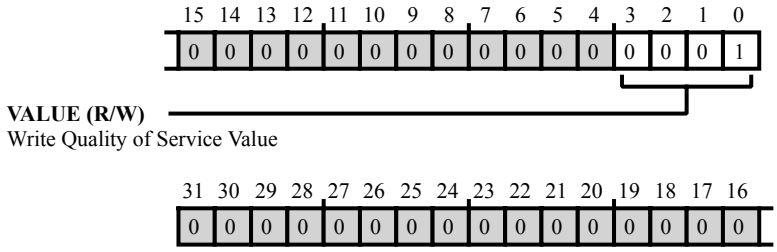

Figure 3-46: SCB_MST19_WQOS Register Diagram

Table 3-47: SCB_MST19_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST19_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master20 Read Quality of Service

The SCB_MST20_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

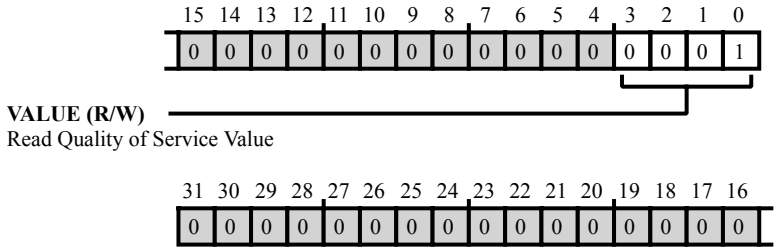

Figure 3-47: SCB_MST20_RQOS Register Diagram

Table 3-48: SCB_MST20_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST20_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 20 Write Quality of Service

The SCB_MST20_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

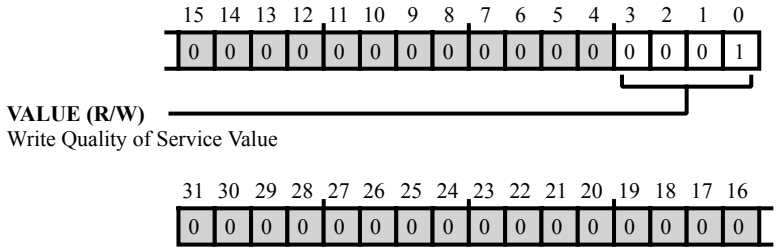

Figure 3-48: SCB_MST20_WQOS Register Diagram

Table 3-49: SCB_MST20_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST20_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 21 Read Quality of Service

The SCB_MST21_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

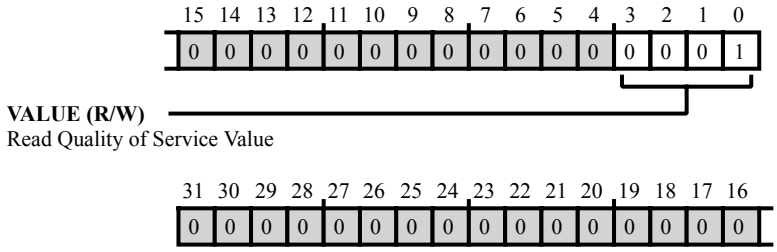

Figure 3-49: SCB_MST21_RQOS Register Diagram

Table 3-50: SCB_MST21_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST21_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 21 Write Quality of Service

The SCB_MST21_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

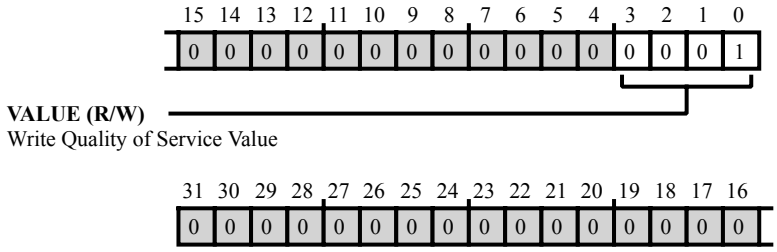

Figure 3-50: SCB_MST21_WQOS Register Diagram

Table 3-51: SCB_MST21_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST21_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 22 Read Quality of Service

The SCB_MST22_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

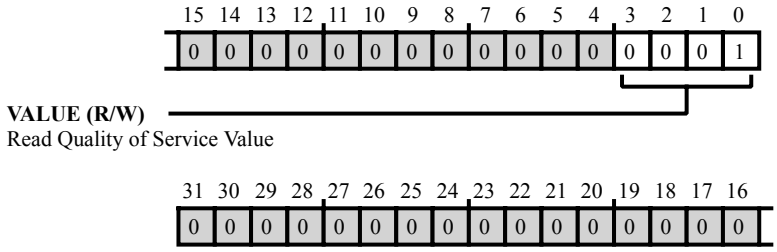

Figure 3-51: SCB_MST22_RQOS Register Diagram

Table 3-52: SCB_MST22_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST22_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 22 Write Quality of Service

The SCB_MST22_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

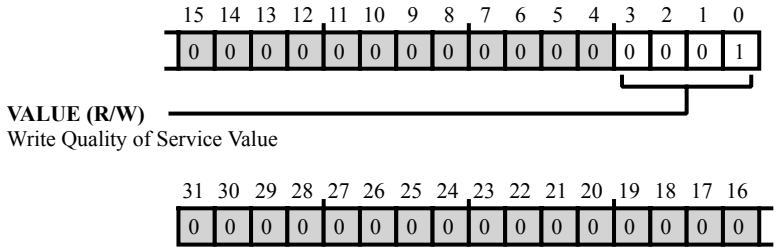

Figure 3-52: SCB_MST22_WQOS Register Diagram

Table 3-53: SCB_MST22_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST22_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 23 Read Quality of Service

The SCB_MST23_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

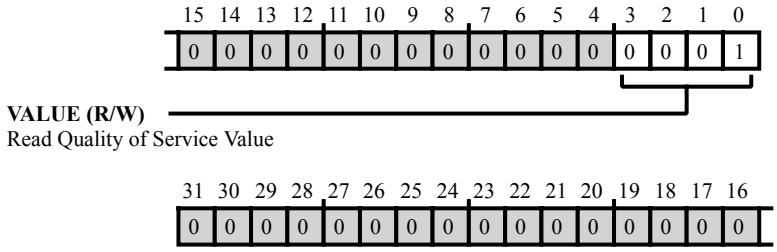

Figure 3-53: SCB_MST23_RQOS Register Diagram

Table 3-54: SCB_MST23_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST23_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 23 Write Quality of Service

The SCB_MST23_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

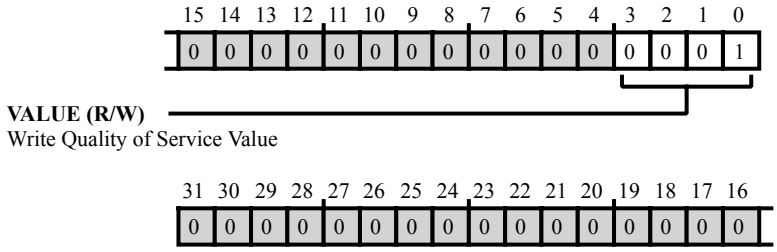

Figure 3-54: SCB_MST23_WQOS Register Diagram

Table 3-55: SCB_MST23_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST23_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 24 Read Quality of Service

The SCB_MST24_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

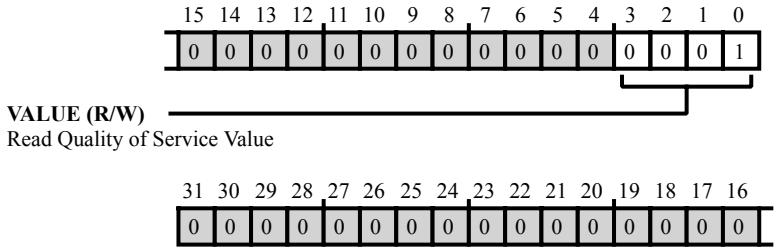

Figure 3-55: SCB_MST24_RQOS Register Diagram

Table 3-56: SCB_MST24_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST24_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 24 Write Quality of Service

The SCB_MST24_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

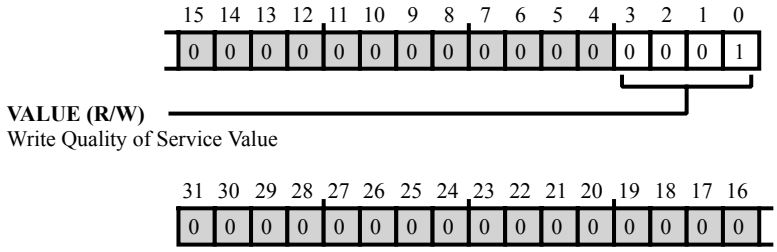

Figure 3-56: SCB_MST24_WQOS Register Diagram

Table 3-57: SCB_MST24_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST24_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 25 Read Quality of Service

The SCB_MST25_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

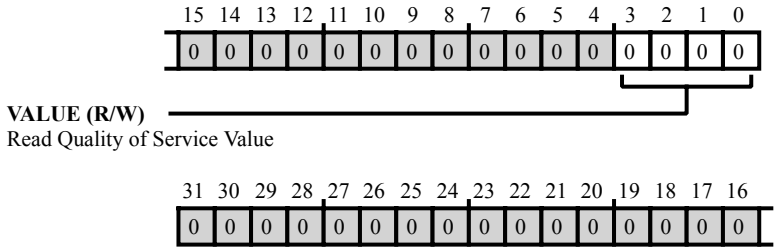

Figure 3-57: SCB_MST25_RQOS Register Diagram

Table 3-58: SCB_MST25_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST25_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 25 Write Quality of Service

The SCB_MST25_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

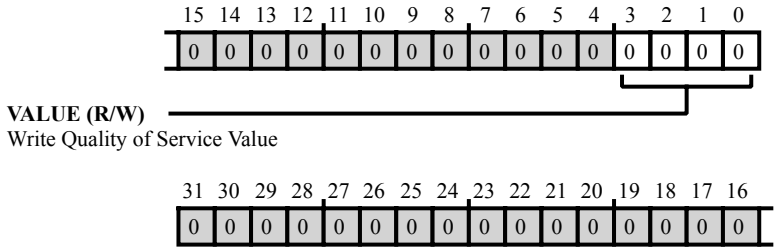

Figure 3-58: SCB_MST25_WQOS Register Diagram

Table 3-59: SCB_MST25_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST25_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

Master 26 Read Quality of Service

The SCB_MST26_RQOS register selects the QoS value for this master's read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

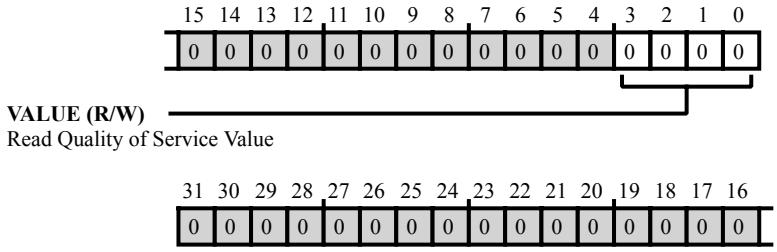

Figure 3-59: SCB_MST26_RQOS Register Diagram

Table 3-60: SCB_MST26_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service Value. The SCB_MST26_RQOS.VALUE bit field holds the programmable QoS value for this master's read channel.

Master 26 Write Quality of Service

The SCB_MST26_WQOS register selects the QoS value for this master's write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

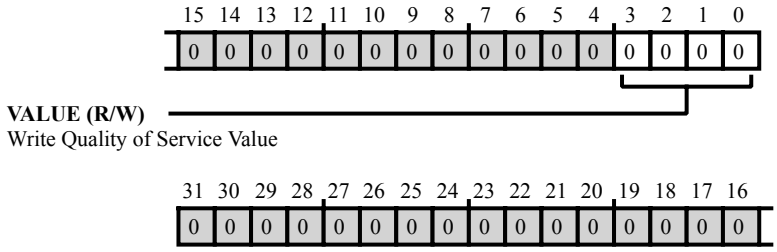

Figure 3-60: SCB_MST26_WQOS Register Diagram

Table 3-61: SCB_MST26_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service Value. The SCB_MST26_WQOS.VALUE bit field holds the programmable QoS value for this master's write channel.

4 Clock Generation Unit (CGU)

The clock generation unit (CGU) includes the phase locked loop (PLL) and the PLL control unit (PCU). The PLL generates a clock that runs at a frequency that is a multiple of the CLKIN input clock frequency. It also generates all on-chip clocks and synchronization signals. The PCU allows the application software to control the PLL module operation.

CGU Features

The CGU module supports the following features:

- Generates all on-chip clocks and synchronization signals; programmable values divide the PLL clock frequency.

  The CGU generates the core clock (*CCLK*), the system clocks (*SYSCLK* and *SCLK*), and the output clock (*OCLK*).
- Provides smooth transitions from current clock condition to new condition with PLL logic, executes the changes to clocks due to register programming
- Supports programmable options for the SYS_CLKOUT output, providing output divided-down versions of the on-chip clocks
- Provides PLL and clock domain status reporting for event management
- Manages power dynamically, allowing the dynamic control of the core clock frequency (f_{CCLK}) of the processor

NOTE: For more information about processor-specific CGU features, see the processor data sheet.

CGU Functional Description

The CGU (clock generation unit) generates all on-chip clocks and synchronization signals based on the programmed PLL multiplication factor and dividers. The CGU provides the following functionality.

Change the PLL clock frequency

The CGU allows programs to change the PLL clock frequency by writing new values to bits in the control register. Any time the PLL relocks, the CGU aligns all core and system clocks.

Change other clock frequencies

The CGU allows programs to change the *CCLKn*, *SYSCLK*, *SCLKn*, *DCLK*, and *OUTCLK* frequencies by writing values to the CGU_DIV register. Any time the clock frequency is changed, the *OUTCLK*, *CCLKn*, *SYSCLK*, *DCLK* and *SCLKn* clocks exit the frequency change sequence aligned.

Perform clock alignment

The CGU can align all clocks by writing to the CGU_DIV register. This function aligns all PLL-based clocks.

For more information on these functions, see the CGU Programming Model section.

ADSP-CM40x CGU Register List

The clock generation unit (CGU) includes the phase locked loop (PLL) and the PLL control unit (PCU). The PLL generates a clock, running at a frequency that is a multiple of the CLKIN input clock's frequency. The CGU also generates all on-chip clocks and synchronization signals. The PCU permits application software control of the PLL's operation. A set of registers govern CGU operations. For more information on CGU functionality, see the CGU register descriptions.

Table 4-1: ADSP-CM40x CGU Register List

Name	Description
CGU_CLKOUTSEL	CLKOUT Select Register
CGU_CTL	Control Register
CGU_DIV	Clocks Divisor Register
CGU_OSCWDCTL	Oscillator Watchdog Register
CGU_STAT	Status Register
CGU_TSCOUNT0	Time Stamp Counter 32 LSB Register
CGU_TSCOUNT1	Time Stamp Counter 32 MSB Register
CGU_TSCTL	Timestamp Control Register
CGU_TSVALUE0	Timestamp Counter Initial 32 LSB Value Register
CGU_TSVALUE1	Time Stamp Counter Initial MSB Value Register

ADSP-CM40x CGU Interrupt List

Table 4-2: ADSP-CM40x CGU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
1	CGU0_EVT	CGU0 PLL Lock Count Expired	Edge	

ADSP-CM40x CGU Trigger List

Table 4-3: ADSP-CM40x CGU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
1	CGU0_EVT	CGU0 Event	Edge

Table 4-4: ADSP-CM40x CGU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

CGU Definitions

DPM

The dynamic power management (DPM) works with the CGU to provide flexible power dissipation modes for the processor.

PCU

The PLL control unit (PCU) in the CGU controls PLL operations. All the MMR registers of the CGU are implemented in this unit.

PLL

The phase-locked loop (PLL) operates within the CGU.

RCU

The reset control unit (RCU) provides input to the CGU to manage clocks during processor reset.

CGU

The clock generation unit (CGU) is comprised of the PLL and PCU. The CGU generates the clocks listed in the ***Clock Descriptions*** table.

Table 4-5: Clock Descriptions

Clock	Description
PLLCLK	Phase-locked loop clock provides the source from which all clocks listed in this table are derived from unless the PLL is bypassed
CCLK	Core Clock
SYSCLK	Clock for system buses and peripherals

Table 4-5: Clock Descriptions (Continued)

Clock	Description
SCLK	Identical to SYSCLK
OCLK	Output clock is a possible source for SYS_CLKOUT

NOTE: On ADSP-CM40x processors, the CCLKn signal is equivalent to CCLK0, and the SYSCLK signal is equivalent to SCLKn.

CGU PLL Block Diagram

The ***CGU PLL Block Diagram*** provides a top-level block diagram of the phase locked loop (PLL). The main blocks of the PLL are the phase/frequency detector (PFD), the charge pump, the loop filter, and the voltage controlled oscillator (VCO). The VCO multiplies the `SYS_CLKIN` input to a higher frequency.

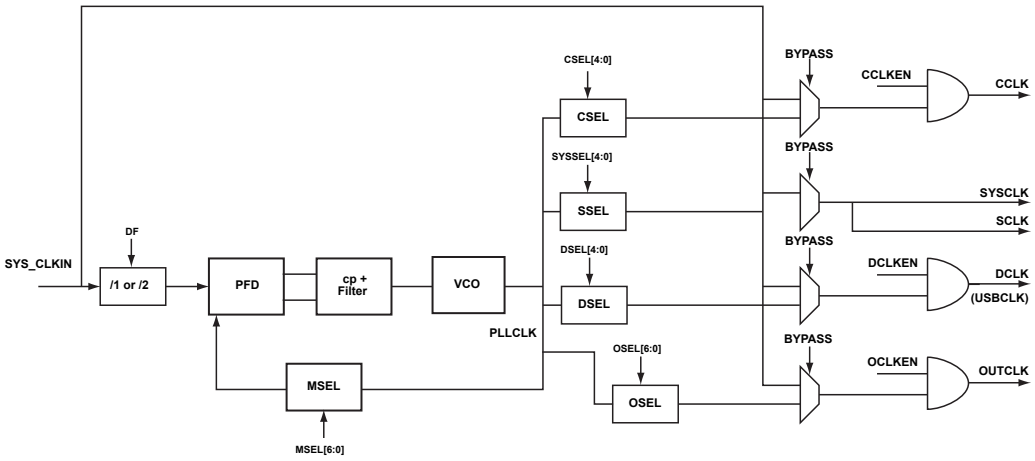

Figure 4-1: CGU PLL Block Diagram

The output of these blocks is called *PLLCLK*. The *PLLCLK* is divided to form *CCLK, SYSCLK*, and *OCLK*.

NOTE: On ADSP-CM40x processors, the *SYSCLK* signal is equivalent to the *SCLKx* signals mentioned throughout this book. The *CCLKn* signal is equivalent to the *CCLK0* signal mentioned throughout this book. Also note that for the ADSP-CM40x, the *DCLK* signal is equivalent to the *USBCLK* signal.

The *OCLK* (shown in the CGU PLL block diagram) is routed to the *CLKOUT* block (shown in the ***SYS_CLKOUT Generation*** figure). *OCLK* can be selected as one of the `CGU_DIV SYS_CLKOUT` sources.

The ***SYS_CLKOUT Generation*** figure is a conceptual representation of the CLKOUT module. As shown in the ***CGU PLL Block Diagram***, many clocks are available on the `SYS_CLKOUT` output pin. The `CGU_CLKOUTSEL` bit controls the clock outputs selection on the `SYS_CLKOUT` pin.

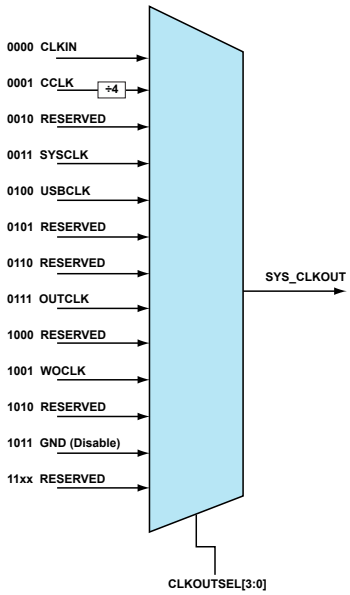

Figure 4-2: SYS_CLKOUT Generation

CGU Operating Modes

The CGU does not have configurable operating modes, but CGU operations affect the operating modes of other modules. Some CGU operation issues that affect operation of other modules include the following:

- The PLL of the CGU operates in either normal mode (CGU clock divisors applied) or bypass mode (CGU PLL is bypassed and clock divisors ignored).
- The SCB uses the CGU for clock synchronization across clock domains. For more information, see the System Crossbars (SCB) chapter.
- The DPM uses the CGU for clock management as power state transitions occur. For more information, see the Dynamic Power Management (DPM) chapter.
- Clock gating control to obtain flexible low-power modes.

CGU Power-up Sequence

The power-up scenario requires that all power supplies and PLL input clocks are valid before the system HW reset pin is deasserted. The deassertion starts to lock the PLL with default settings. For details, refer to the specific product data sheet.

CGU Event Control

The CGU generates an event or error for several different reasons.

CGU Events

After a frequency change, a CGU event indicates that the PLL has locked and clocks are synchronized. If a core was idled while changing frequencies, the CGU can use an event interrupt to break the core idle. While in active mode, a CGU event indicates that the PLL has locked.

CGU Errors

A CGU error occurs under following conditions:

- A write access to the CGU_DIV register triggers an alignment sequence while the PLL is locked and is still aligning the clocks.

  The CGU_STAT.WDIVERR bit state indicates this error. If this error occurs, clear the CGU_STAT.WDIVERR bit and rewrite the desired values to the CGU_DIV register.

- A change to the CGU_DIV register occurs while the PLL is locked and is still aligning the clocks

  The CGU_STAT.WDIVERR bit state indicates this error. If this error occurs, clear the CGU_STAT.WDIVERR bit and rewrite the desired values to the CGU_DIV register.

- A write access to the CGU_CTL.DF bit field occurs or a write access to the CGU_CTL.MSEL bit field occurs while the PLL is locking.

  The CGU_STAT.WDFMSERR bit state indicates this error. If this error occurs, wait until the PLL has finished locking, clear the error, and rewrite the desired value change.

- A clock divisor value error occurs when the *CCLK* divisor is greater than the *SYSCLK* divisor. For example, the CGU_DIV.CSEL greater than CGU_DIV.SYSSEL.

  The CGU_STAT.WDIVERR bit state indicates this error. If this error occurs, clear it. The CGU writes new values to the CGU_DIV.CSEL bit field, so the field is less than or equal to the CGU_DIV.SYSSEL bit field value.

The CGU monitors changes to the following fields:

- CCLK Divisor - CGU_DIV.CSEL
- SYSCLK Divisor - CGU_DIV.SYSSEL

CGU Generated Bus Errors

The CGU generates a bus error when a read or write transaction is attempted to an unused address within the CGU address range. It also generates a bus error when a misaligned access is made to a CGU register. In addition to the bus error, the CGU_STAT.ADDRERR bit is set. If a write to a write-protected CGU register is attempted, the CGU generates a bus error. In addition, the CGU_STAT.LWERR bit is set.

Oscillator Watchdog

The oscillator watchdog detects the absence of input clock transitions and provides a fault warning through the SYS_FAULT pin. To detect harmonic or subharmonic crystal oscillator behavior, the watchdog (under

programmable control) also detects and reports input oscillator frequencies above and below specified limits. Use an internal asynchronous, local 1-MHz oscillator combined with a series of programmable counters for this detection. Set the CGU_OSCWDCTL.MONDIS bit and clear the CGU_OSCWDCTL.FAULTEN bit in the control register to optionally disable all the input clock monitor and fault detection functions.

Set the CGU_OSCWDCTL.HODEN bit to enable harmonic oscillation detection. The CGU uses the CGU_OSCWDCTL.HODF bit field to indicate the desired lower fail limit for the harmonic oscillation detection in MHz. The upper limit is always twice the lower limit. The ***HODF Settings for Different Input Clock Frequencies*** table shows an example of the CGU_OSCWDCTL.HODF bit settings for different input clock frequencies.

Table 4-6: HODF Settings for Different Input Clock Frequencies

HODF[5:0]	Subharmonic Frequency (MHz)	Nominal Lower Fail Limit	Input Clock Frequency (MHz)	Nominal Upper Fail Limit (MHz)	Second Harmonic Frequency (MHz)
14	10	MHz	20	28	40
21	15	MHz	30	42	60

The CGU uses the CGU_OSCWDCTL.BOUF asynchronous control bit field to indicate the desired upper fail limit for the bad oscillation detection. set theCGU_OSCWDCTL.BOUEN bit to enable upper-limit bad oscillation detection. A bad oscillation detection condition signals a fault before any processor operations occur. This detection occurs (even in bypass mode) whenever a clock frequency exceeds its specifications.

CGU_OSCWDCTL.BOUF =0 starts with a target of 32 MHz and each additional LSB increases the frequency test limit by 2 MHz. For example:

Target Upper Frequency Limit = CGU_OSCWDCTL.BOUF × 2 MHz + 32 MHz

The CGU_STAT.OSCWDSTATFC status bits indicate the nature of the fault. The ***Fault Map*** table shows the fault values.

Table 4-7: Fault Map

FAULT_CODE[2:0]	Fault Type
0	No Fault
1	No Input Clock
2	Subharmonic CLKIN
3	Harmonic CLKIN
4	No AUX_CLK
5	CLKIN > Upper Freq Limit (BOUF)
6	Reserved
7	Multiple Limit Faults

There is a priority to the faults given in the case of multiple fault errors. The highest priority is given to No Input Clock followed by No AUX_CLK. The other three fault cases share the lowest priority. Multiple Limit Faults are asserted if more than one type of Subharmonic CLKIN, Harmonic CLKIN, or BOUF faults are observed.

NOTE: All the CGU_STAT.OSCWDSTATFC other than the absence of AUX_CLK (for example CGU_STAT.OSCWDSTATFC =4) is not reliable and is used for debug only.

The CGU uses the CGU_OSCWDCTL.CNGEN bit to enable the detection of the clock fault. An asynchronous reset is issued to the processor through reset control unit or dynamic power management module. By default, this bit is disabled.

CGU Programming Model

This section describes the programming concepts and mode configuration techniques for the CGU.

Configuring CGU Modes

This section provides procedures related to clock and PLL configuration.

Changing Clock Frequencies

Applications change clock frequencies in two ways. The first way is modifying the PLL multiplication value by writing to the CGU_CTL register and the second is modifying the clock dividers by writing to the CGU_DIV register. Both actions have different implications even if the frequencies of the final clock are the same. Write accesses to change the CGU_CTL.DF or CGU_CTL.MSEL bit fields, while the PLL is locking, set the CGU_STAT.WDFMSERR error bit. The CGU_STAT.WDIVERR error bit is set when one of following accesses is attempted while the PLL is locked but still aligning the clocks:

- A write access to the CGU_DIV register to trigger an alignment sequence
- A write access to the CGU_DIV register to change the CGU_DIV.CSEL, CGU_DIV.SYSSEL, CGU_DIV.S0SEL, CGU_DIV.S1SEL, or CGU_DIV.DSEL bits

Read-after-write accesses to these registers return the new value, even if the frequency of the clock change is still in-progress.

Modifying the PLL multiplier requires the PLL to relock. Once the PLL locks, synchronize the clocks. Changes to the CGU_CTL.DF or CGU_CTL.MSEL bit field result in bypassing the PLL. By setting the CGU_CTL.WFI bit, programs force the PLL to wait for all the cores to return to their idle or reset states before the frequency changes. If necessary, clear the CGU_DIV.UPDT bit to avoid multiple clock alignment sequences. If the CGU_DIV register is not updated, the CGU module uses the current values to determine the frequencies of the clock. It is the programs responsibility to guarantee that the new CGU_CTL.DF or CGU_CTL.MSEL and CGU_DIV combinations are legal.

Changing the PLL Clock Frequency

To change the phase-locked loop clock (*PLLCLK*) frequency, write new values to the CGU_CTL.MSEL field or CGU_CTL.DF field. Any time the PLL relocks, all core and system clocks are aligned.

1. Read CGU_STAT register and verify that:
    a. The CGU_STAT.PLLEN bit =1 (PLL enabled).
    b. The CGU_STAT.PLOCK bit =1 (PLL is not locking).
    c. The CGU_STAT.CLKSALGN bit =0 (clocks aligned).
2. Write the desired values to the clock divisor select fields of the CGU_DIV register with the CGU_DIV.UPDT bit =0.
3. Write the desired values to the CGU_CTL.DF and CGU_CTL.MSEL fields.
    a. To change the PLL frequency while the cores are idle, write to the CGU_CTL register with the CGU_CTL.WFI bit =1.
    b. To change the PLL frequency while the cores are active, write to the CGU_CTL register with the CGU_CTL.WFI bit =0.

This sequence:

1. Updates the corresponding CGU registers.
2. Bypasses the PLL.
3. Makes the PLL lock to the new values in the CGU_CTL.MSEL or CGU_CTL.DF fields.
4. Changes the clock frequencies.
5. Exits PLL bypass with all clocks aligned.

When exiting the PLL bypass state, a CGU event occurs.

The CGU_STAT register exits this sequence with the CGU_STAT.PLLEN bit =1, the CGU_STAT.PLOCK bit =1, the CGU_STAT.PLLBP bit =0, and the CGU_STAT.CLKSALGN bit =0. Poll the CGU_STAT.PLOCK bit, CGU_STAT.PLLBP bit, and CGU_STAT.CLKSALGN bit to discover when the PLL is locked and the clocks are aligned.

Changing the frequency of the PLL is allowed while the PLL is bypassed. But, the new PLLCLK frequency is not used until the PLL is no longer bypassed.

Changing the CCLKn or SYSCLK Frequency Without Modifying the PLLCLK Frequency

To change the clock frequencies, write new values to CGU_DIV.CSEL or CGU_DIV.SYSSEL bits. The frequency change occurs only when the PLL is not bypassed. Any time the *CCLKn* or *SYSCLK* clock frequencies are changed, they exit the frequency change sequence aligned.

1. Read the CGU_STAT register to verify that the CGU_STAT.CLKSALGN bit =0 (clocks aligned).

2. Write the desired CGU_DIV.CSEL, CGU_DIV.SYSSEL, and CGU_DIV.OSEL bit field values with the CGU_DIV.UPDT bit = 1.

   ADDITIONAL INFORMATION: This write updates the CGU_DIV register, changes the *SCLKn* and *SYSCLK* frequencies, and aligns the clocks. When the clocks are aligned, a CGU event occurs.

The CGU_STAT register exits this sequence with the CGU_STAT.CLKSALGN bit =0. Poll the CGU_STAT.CLKSALGN bit to discover when the clocks are aligned. Any write attempt to change the CGU_DIV.S0SEL or CGU_DIV.S1SEL bit fields while CGU_STAT.CLKSALGN bit =1 (clocks alignment in progress) triggers an MMR access bus error and the CGU_DIV register is not modified.

Programming the *SYSCLK* frequency to a higher value than *CCLKn* also triggers an MMR access bus error and the CGU_DIV register is not modified.

Writing to the CGU_DIV register is allowed while the processor is in active (PLL bypassed) mode. But, the effect of the write is visible only after the transition to full-on (PLL not bypassed) mode.

Changing the USBCLK Clock Frequency

To change the *USBCLK* clock frequency, write a new CGU_DIV.DSEL value. The frequency change occurs only when the PLL is not bypassed. Any time *USBCLK* clock frequency is changed, it exits the frequency change sequence aligned. The CGU_SYSUSBCLK_ALGN bit = 1 if the *SYSCLK* and the *USBCLK* frequencies are equal.

1. Read the CGU_STAT register to verify that the CGU_STAT.CLKSALGN bit =0 (clocks aligned).
2. Write desired CGU_DIV.DSEL value with the CGU_DIV.UPDT bit =1.

   ADDITIONAL INFORMATION: This write updates the CGU_DIV register, changes the *USBCLK* frequency, and aligns all clocks except *OUTCLK*.

The CGU_STAT register exits this sequence with the CGU_STAT.CLKSALGN bit =0. Poll the CGU_STAT.CLKSALGN bit to discover when the clocks are aligned. Any write attempt to change the CGU_DIV.DSEL field while CGU_STAT.CLKSALGN bit =1 (clocks alignment in progress) triggers an MMR access bus error and the CGU_DIV register is not modified. When clocks are aligned a CGU event occurs.

Writing to CGU_DIV.DSEL bits is allowed while the processor is in active (PLL bypassed) mode. But, the effect of the write is visible only after the transition to full-on (PLL not bypassed) mode.

Changing the OUTCLK Frequency

To change the *OUTCLK* clock frequency, write a new CGU_DIV.OSEL bit value. Any time the *OUTCLK* clock frequency is changed, the *OUTCLK*, *CCLKn*, *SYSCLK*, and *SCLKn* clocks exit the frequency change sequence aligned.

1. Read the CGU_STAT register to verify that the CGU_STAT.CLKSALGN bit =0 (clocks aligned).
2. Write desired CGU_DIV.OSEL value with the CGU_DIV.UPDT bit =1.

ADDITIONAL INFORMATION: This write updates the CGU_DIV register, changes the *OUTCLK* frequency, and aligns all clocks except *OUTCLK*.

The CGU_STAT register exits this sequence with the CGU_STAT.CLKSALGN bit =0. Poll the CGU_STAT.CLKSALGN bit to discover when the clocks are aligned. Any write attempt to change the CGU_DIV.DSEL field while the CGU_STAT.CLKSALGN bit =1 (clock alignment in progress) triggers an MMR access bus error and the CGU_DIV register is not modified. When clocks are aligned, a CGU event occurs.

Writing to the CGU_DIV.OSEL bit field is allowed while the processor is in active (PLL bypassed) mode. But, the effect of the write is visible only after the transition to full-on (PLL not bypassed) mode.

Aligning All Clocks

To align the clocks, write 1 to the CGU_DIV.ALGN bit. The frequency can be changed, if necessary. The clocks aligned include:

- *CCLKn*
- *SYSCLK*
- *SCLKn*
- *DCLK*
- *OUTCLK*

1. Read the CGU_STAT register to verify that CGU_STAT.CLKSALGN bit =0 (clocks aligned).
2. Write 1 to the CGU_DIV.ALGN bit. All other fields can change.

   ADDITIONAL INFORMATION: This write does not alter the CGU_DIV register unless one of the clock-select fields is modified. When clocks are aligned, a CGU event occurs.

The CGU_STAT register exits this sequence with the CGU_STAT.CLKSALGN bit =0. Poll the CGU_STAT.CLKSALGN bit to discover when the clocks are aligned. Any write to the CGU_DIV register intended to align clocks or to change a clock select field while the CGU_STAT.CLKSALGN bit =1 (clocks alignment in progress) triggers an MMR access bus error. And, the CGU_DIV register is not modified.

Writing 1 to the CGU_DIV.ALGN bit has no effect while the processor is in active (PLL bypassed) mode.

Valid Clock Multiplier Settings

Processor operations depend on valid settings in the CGU_CTL and CGU_DIV registers. These registers control the clock multiplier and divisor values. Set these registers such that the minimum and maximum clocks specified in the data sheet are not violated. All other clock specifications in the data sheet must also be adhered to for correct operation of the part.

ADSP-CM40x CGU Register Descriptions

Clock Generation Unit (CGU) contains the following registers.

Table 4-8: ADSP-CM40x CGU Register List

Name	Description
CGU_CLKOUTSEL	CLKOUT Select Register
CGU_CTL	Control Register
CGU_DIV	Clocks Divisor Register
CGU_OSCWDCTL	Oscillator Watchdog Register
CGU_STAT	Status Register
CGU_TSCOUNT0	Time Stamp Counter 32 LSB Register
CGU_TSCOUNT1	Time Stamp Counter 32 MSB Register
CGU_TSCTL	Timestamp Control Register
CGU_TSVALUE0	Timestamp Counter Initial 32 LSB Value Register
CGU_TSVALUE1	Time Stamp Counter Initial MSB Value Register

CLKOUT Select Register

The CGU_CLKOUTSEL selects the signal that the CGU drives through the CLKOUT multiplexer. Also, this register selects the divisor for the USBCLK output.

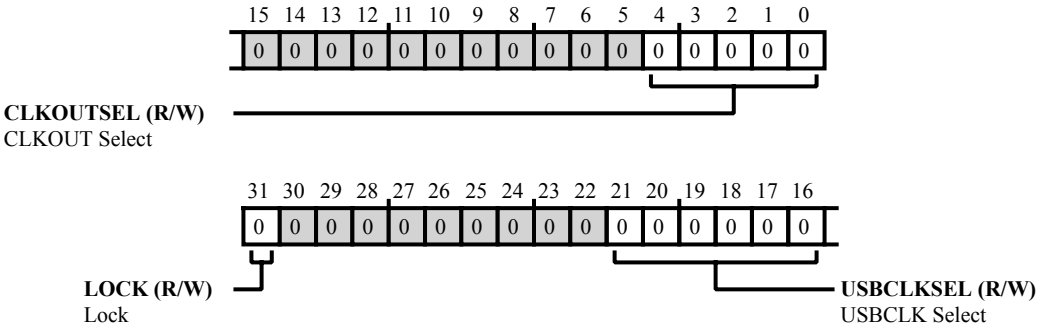

Figure 4-3: CGU_CLKOUTSEL Register Diagram

Table 4-9: CGU_CLKOUTSEL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the CGU_CLKOUTSEL.LOCK bit is set, the CGU_CLKOUTSEL register is read only (locked).	
		0	Unlock
		1	Lock
21:16 (R/W)	USBCLKSEL	USBCLK Select. The CGU_CLKOUTSEL.USBCLKSEL selects the divisor in the USBCLK equation: USBCLK frequency = (USB PLL frequency) / (CGU_CLKOUTSEL.USBCLKSEL +1) Where the value of CGU_CLKOUTSEL.USBCLKSEL is between 0 and 63.	
		0	USBCLKSEL = 0 to 63
		63	USBCLKSEL = 63
4:0 (R/W)	CLKOUTSEL	CLKOUT Select. The CGU_CLKOUTSEL.CLKOUTSEL selects the signal that the CGU drives through the CLKOUT pin multiplexer.	
		0	CLKBUF (Buffered version of SYS_CLKIN)
		1	CCLK0/4
		2	Reserved
		3	SYSCLK (SCLK)
		4	DCLK (USBCLK)

Table 4-9: CGU_CLKOUTSEL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		5	Reserved
		6	Reserved
		7	OCLK
		8	Reserved
		9	WOCLK (Osc Watchdog)
		10	Reserved
		11	GND (Disable CLKOUT)
		15-31	Reserved

Control Register

The CGU_CTL controls the clock generation divisors for SYS_CLKIN and the PLL. Read after write accesses to the CGU_CTL register returns the new value even if the clock's frequency change is still in progress.

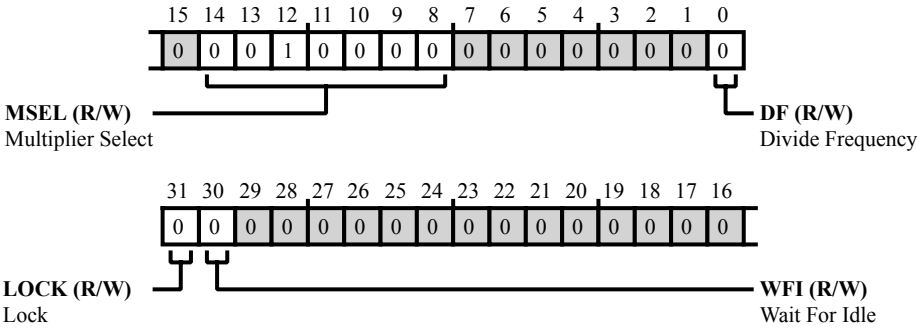

Figure 4-4: CGU_CTL Register Diagram

Table 4-10: CGU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the CGU_CTL.LOCK bit is set, the CGU_CTL register is read only (locked).	
		0	Unlock
		1	Lock
30 (R/W)	WFI	Wait For Idle. Modifying the PLL multiplier requires the PLL to re-lock and once the PLL locks, clocks have to be synchronized. Changes to the CGU_CTL.MSEL and the CGU_CTL.DF result in bypassing the PLL. The CGU_CTL.WFI force the PLL to wait for all processor cores to be in an idle or reset state before changing frequencies as a result of change to the CGU_CTL.MSEL or CGU_CTL.DF fields. Write accesses to CGU_CTL to change CGU_CTL.DF or CGU_CTL.MSEL while the PLL is locking sets the CGU_STAT.WDFMSERR bit.	
		0	Update Immediately
		1	Wait for Idle
14:8 (R/W)	MSEL	Multiplier Select. The CGU_CTL.MSEL selects the multiplier in the PLLCLK equation: PLLCLK frequency = (SYS_CLKIN frequency / (DF+1)) * MSEL Where the value of MSEL is between 1 and 127.	
		0	MSEL = 128
		1-127	MSEL = 1 to 127

Table 4-10: CGU_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	DF	Divide Frequency. The `CGU_CTL.DF` selects whether or not the SYS_CLKIN input is divided by two before being passed to the PLL.	
		0	Pass OSC_CLKIN to PLL
		1	Pass OSC_CLKIN/2 to PLL

Clocks Divisor Register

The CGU_DIV register controls clock divisors for core clocks, system clocks, external (off core) memory clocks, and output clock. Read after write accesses to the CGU_DIV register returns the new value even if the clock's frequency change is still in progress.

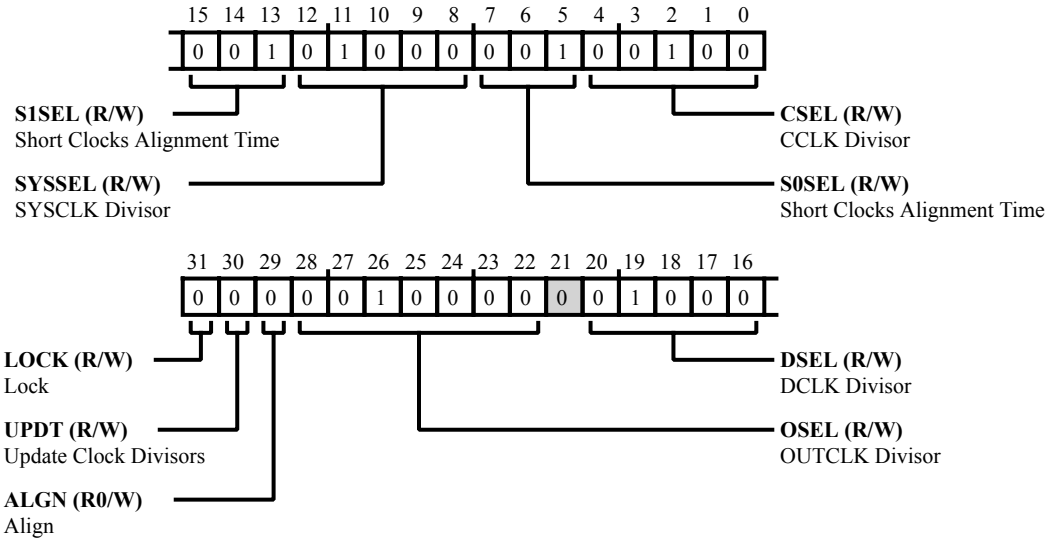

Figure 4-5: CGU_DIV Register Diagram

Table 4-11: CGU_DIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the CGU_DIV.LOCK bit is set, the CGU_DIV register is read only (locked).	
		0	Unlock
		1	Lock
30 (R/W)	UPDT	Update Clock Divisors. The CGU_DIV.UPDT controls whether the CGU drives new CGU_DIV.CSEL, CGU_DIV.SYSSEL, CGU_DIV.S0SEL, CGU_DIV.S1SEL, CGU_DIV.DSEL, and CGU_DIV.OSEL values to PLL after CGU_DIV register update.	
		0	No PLL Update
		1	Drive Updated SEL Values to PLL

Table 4-11: CGU_DIV Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R0/W)	ALGN	Align. The CGU_DIV.ALGN directs the CGU to align the PLL-based clocks. The divisor selections (CGU_DIV.CSEL, CGU_DIV.SYSSEL, CGU_DIV.S0SEL, CGU_DIV.S1SEL, CGU_DIV.DSEL, and/or CGU_DIV.OSEL) do not have to change.	
		0	No Action
		1	Align PLL Clocks
28:22 (R/W)	OSEL	OUTCLK Divisor. The CGU_DIV.OSEL selects the divisor in the OUTCLK equation: OUTCLK frequency = (SYS_CLKIN frequency / (DF+1)) * MSEL / CGU_DIV.OSEL Where the value of CGU_DIV.OSEL is between 1 and 127.	
		0	OSEL = 128
		1-127	OSEL = 1 to 127
20:16 (R/W)	DSEL	DCLK Divisor. The CGU_DIV.DSEL selects the divisor in the DCLK equation: DCLK frequency = (SYS_CLKIN frequency/(DF+1)) MSEL/CGU_DIV.DSEL Where the value of CGU_DIV.DSEL is between 1 and 31.	
		0	DSEL = 32
		1-31	DSEL = 1 to 31
15:13 (R/W)	S1SEL	Short Clocks Alignment Time. The CGU_DIV.S1SEL Determines if the time it takes clocks to align is short or long.	
		0	Long Clocks Alignment Time
		1	Short Clocks Alignment Time
		2-7	Reserved
12:8 (R/W)	SYSSEL	SYSCLK Divisor. The CGU_DIV.SYSSEL selects the divisor in the SYSCLK equation: SYSCLK frequency = (SYS_CLKIN frequency/(DF+1)) MSEL/CGU_DIV.SYSSEL Where the value of CGU_DIV.SYSSEL is between 1 and 31.	
		0	SYSSEL = 32
		1-31	SYSSEL = 1 to 31

Table 4-11: CGU_DIV Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7:5 (R/W)	S0SEL	Short Clocks Alignment Time. The CGU_DIV.S0SEL Determines if the time it takes clocks to align is short or long.	
		0	Long Clocks Alignment Time
		1	Short Clocks Alignment Time
		2-7	S0SEL = 8
4:0 (R/W)	CSEL	CCLK Divisor. The CGU_DIV.CSEL selects the divisor in the CCLK equation: CCLK frequency = (SYS_CLKIN frequency / (DF+1)) * MSEL / CGU_DIV.CSEL Where the value of CGU_DIV.CSEL is between 1 and 31.	
		0	CSEL = 32
		1-31	CSEL= 1 to 31

Oscillator Watchdog Register

The CGU_OSCWDCTL register configures the CGU to allow the detection of the absence of input clock transitions and provides a fault warning via the SYS_FAULT pin. The CGU_OSCWDCTL register also detects and reports input oscillator frequencies above and below specified limits, in order to specifically detect harmonic or sub-harmonic crystal oscillator behavior. This detection is achieved by using an internal asynchronous, local 1 MHz oscillator combined with a series of programmable counters.

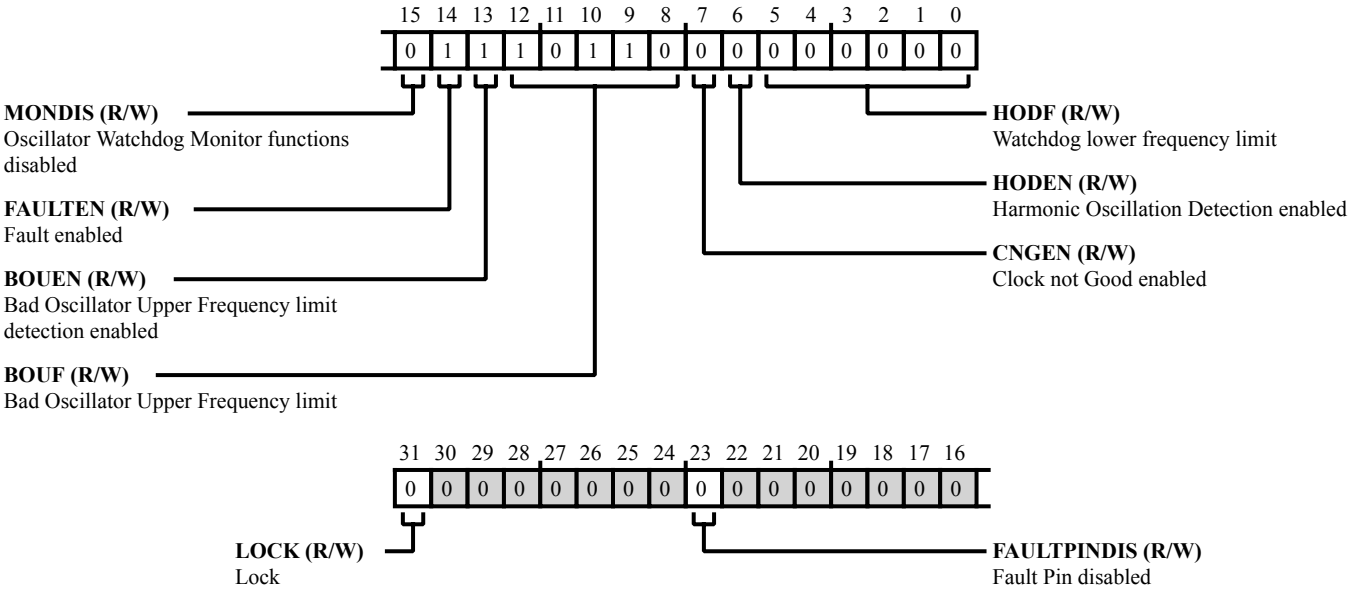

Figure 4-6: CGU_OSCWDCTL Register Diagram

Table 4-12: CGU_OSCWDCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock. If the global lock bit is set and theCGU_OSCWDCTL.LOCK bit is set, the CGU_OSCWDCTL register is read only (locked).
23 (R/W)	FAULTPINDIS	Fault Pin disabled. The CGU_OSCWDCTL.FAULTPINDIS bit disables pin fault detection.
15 (R/W)	MONDIS	Oscillator Watchdog Monitor functions disabled. The CGU_OSCWDCTL.MONDIS bit disables all the input clock monitor and fault detection functions.
14 (R/W)	FAULTEN	Fault enabled. The CGU_OSCWDCTL.FAULTEN bit enables fault detection.
13 (R/W)	BOUEN	Bad Oscillator Upper Frequency limit detection enabled. The CGU_OSCWDCTL.BOUEN bit enables upper limit bad oscillation detection.

Table 4-12: CGU_OSCWDCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12:8 (R/W)	BOUF	Bad Oscillator Upper Frequency limit. The `CGU_OSCWDCTL.BOUF` bits indicate the desired upper fail limit for the bad oscillation detection.
7 (R/W)	CNGEN	Clock not Good enabled. The `CGU_OSCWDCTL.CNGEN` bit enables the detection of an oscillator watchdog clock fault.
6 (R/W)	HODEN	Harmonic Oscillation Detection enabled. The `CGU_OSCWDCTL.HODEN` bit enables harmonic oscillation detection.
5:0 (R/W)	HODF	Watchdog lower frequency limit. The `CGU_OSCWDCTL.HODF` bit field is used to indicate the desired lower fail limit for the harmonic oscillation detection in MHz.

Status Register

The CGU_STAT register reflects the PLL status and errors detected during the PLL configuration. This register may be cleared asynchronously by a reset signal from the RCU module. All bits---except those defined as W1C (write-1-to-clear)---are read only.

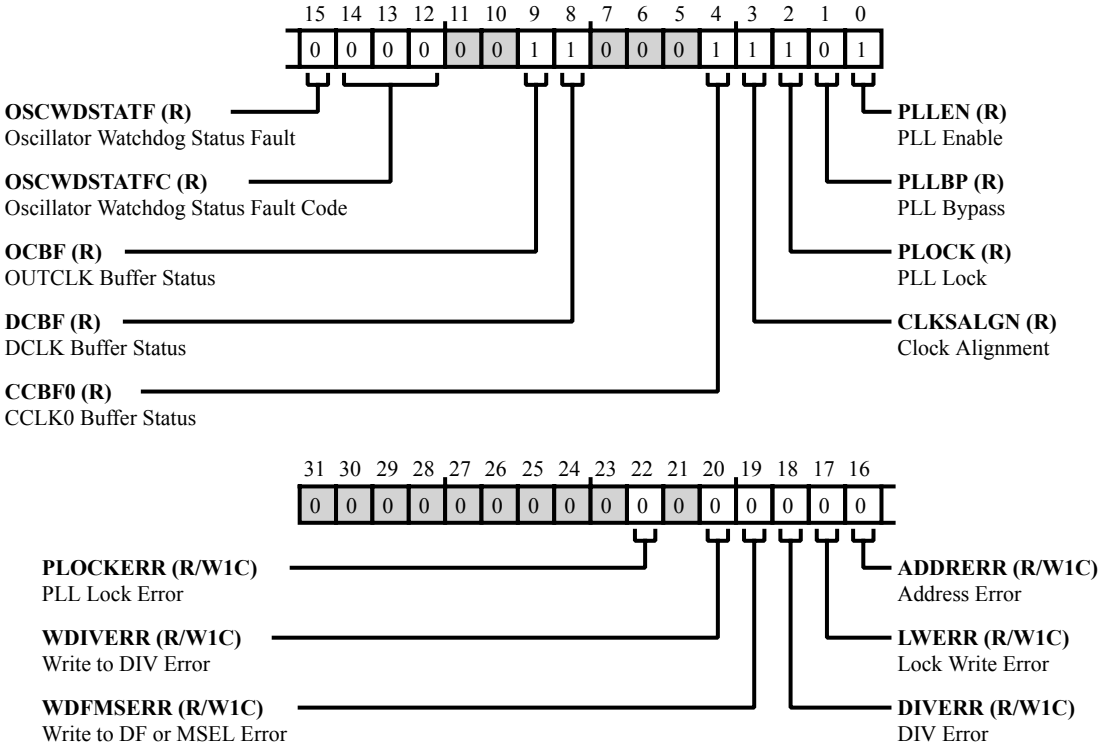

Figure 4-7: CGU_STAT Register Diagram

Table 4-13: CGU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
22 (R/W1C)	PLOCKERR	PLL Lock Error. The CGU_STAT.PLOCKERR bit indicates that the PLL does not start locking. This triggers a CGU_ERROR and bypasses the PLL.	
		0	No Error
		1	PLL Lock Error

Table 4-13: CGU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/W1C)	WDIVERR	Write to DIV Error. The CGU_STAT.WDIVERR indicates a write access to the CGU_DIV register (to trigger an alignment sequence or to change CGU_DIV.CSEL, CGU_DIV.SYSSEL, CGU_DIV.S0SEL, CGU_DIV.S1SEL, or CGU_DIV.DSEL) while the PLL is locked, but still aligning the clocks. Read after write accesses to the CGU_STAT and CGU_DIV registers return the new value even if the clock frequency change is still in progress.	
		0	No Error
		1	Write DIV Error
19 (R/W1C)	WDFMSERR	Write to DF or MSEL Error. The CGU_STAT.WDFMSERR indicates a write access to the CGU_CTL register to change CGU_CTL.DF or CGU_CTL.MSEL while the PLL is locking.	
		0	No Error
		1	Write DF/MSEL Error
18 (R/W1C)	DIVERR	DIV Error. The CGU_STAT.DIVERR indicates a clock divisor value error, occurring when the CCLK clock divisor is greater than the SYSCLK clock divisor, as in: CGU_DIV.CSEL > CGU_DIV.SYSSEL The CGU issues a CGU error for this condition.	
		0	No Error
		1	DIV Error
17 (R/W1C)	LWERR	Lock Write Error. The CGU_STAT.LWERR indicates an attempt to write to write-protected (locked) CGU registers. The CGU issues a bus error for this condition.	
		0	No Error
		1	Lock Write Error
16 (R/W1C)	ADDRERR	Address Error. The CGU_STAT.ADDRERR indicates an attempt to make a read or write access to un-implemented addresses or accesses are non-aligned. The CGU issues a bus error for this condition.	
		0	No Error
		1	Address Error

Table 4-13: CGU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/NW)	OSCWDSTATF	Oscillator Watchdog Status Fault. The CGU_STAT.OSCWDSTATF bit indicates a fault in the oscillator watchdog (CGU's OSC_WDSTAT[1:0] input pins.	
		0	Reserved
		1	Reserved
14:12 (R/NW)	OSCWDSTATFC	Oscillator Watchdog Status Fault Code. The CGU_STAT.OSCWDSTATFC bits indicate the nature of the fault in the oscillator watchdog (CGU's OSC_WDSTAT[1:0] input pins.	
		0-7	Reserved
9 (R/NW)	OCBF	OUTCLK Buffer Status. The CGU_STAT.OCBF indicates whether the OUTCLK buffer is enabled.	
		0	Disabled
		1	Enabled
8 (R/NW)	DCBF	DCLK Buffer Status. The CGU_STAT.DCBF indicates whether the DCLK buffer is enabled.	
		0	Disabled
		1	Enabled
4 (R/NW)	CCBF0	CCLK0 Buffer Status. The CGU_STAT.CCBF0 indicates whether the CCLK0 buffer is enabled.	
		0	Disabled
		1	Enabled
3 (R/NW)	CLKSALGN	Clock Alignment. The CGU_STAT.CLKSALGN indicates whether a clock alignment sequence is in progress. This bit is set when clocks alignment is required by changes to CGU_DIV.CSEL, CGU_DIV.S0SEL, CGU_DIV.S1SEL, CGU_DIV.DSEL, or CGU_DIV.OSEL. The CGU_STAT.CLKSALGN bit is cleared when clocks are aligned. Note that (after a PLL frequency change in active state) the CGU_STAT.CLKSALGN bit may indicate that clocks are not aligned even though the clocks are aligned (all clocks are aligned and running at CLKIN frequency).	
		0	Clocks are Aligned
		1	Clocks not Aligned (alignment in progress)

Table 4-13: CGU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/NW)	PLOCK	PLL Lock. The CGU_STAT.PLOCK indicates whether the PLL is locked. This bit is set when the PLL locks (PLL lock counter end-of-count). The CGU_STAT.PLOCK bit is cleared when requested PLL frequency change (for PLL reset, PLL disable-to-enable transition, or change to CGU_CTL.MSEL or CGU_CTL.DF) is in progress.	
		0	PLL not Locked (PLL frequency change in progress)
		1	PLL Locked
1 (R/NW)	PLLBP	PLL Bypass. The CGU_STAT.PLLBP indicates whether the PLL is bypassed. The default value for CGU_STAT.PLLBP is determined by the bypass strap pin.	
		0	PLL not Bypassed
		1	PLL Bypassed
0 (R/NW)	PLLEN	PLL Enable. The CGU_STAT.PLLEN indicates whether the PLL is enabled.	
		0	Disabled
		1	Enabled

Time Stamp Counter 32 LSB Register

The CGU_TSCOUNT0 register address is used to read the CoreSight Timestamp counter LSB 32-bit (bits [31:0]) value.

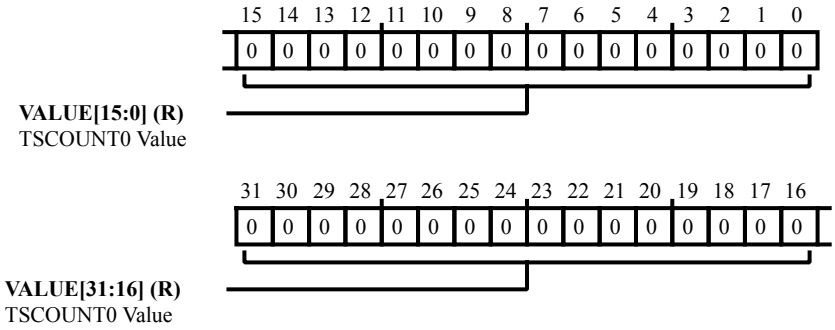

Figure 4-8: CGU_TSCOUNT0 Register Diagram

Table 4-14: CGU_TSCOUNT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	TSCOUNT0 Value. The CGU_TSCOUNT0.VALUE bit field holds the time stamp counter 32 LSBs.

Time Stamp Counter 32 MSB Register

The CGU_TSCOUNT1 register address is used to read the CoreSight Timestamp counter MSB 32-bit (bits [63:32]) value.

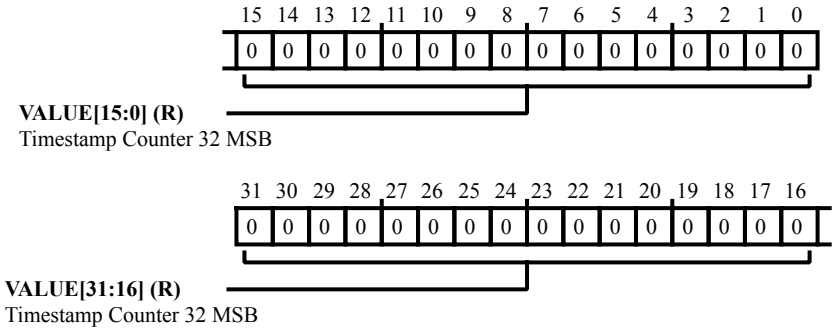

Figure 4-9: CGU_TSCOUNT1 Register Diagram

Table 4-15: CGU_TSCOUNT1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Timestamp Counter 32 MSB. The CGU_TSCOUNT1.VALUE bit field holds the time stamp counter 32 MSBs.

Timestamp Control Register

The CGU_TSCTL register controls the operation of the CoreSight Timestamp counter.

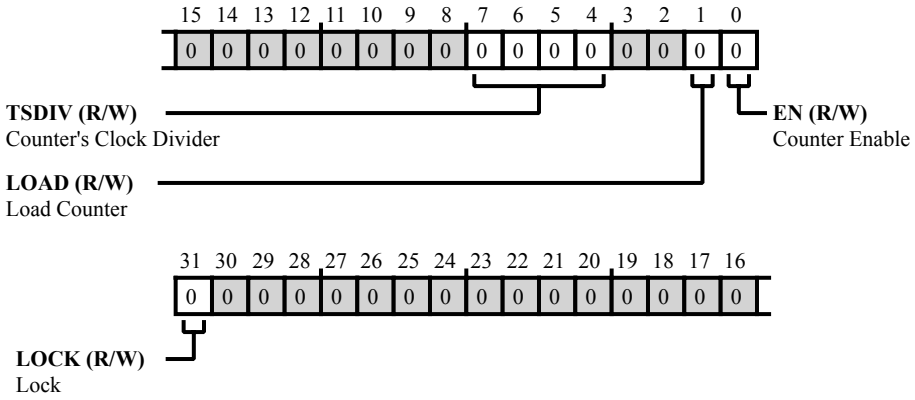

Figure 4-10: CGU_TSCTL Register Diagram

Table 4-16: CGU_TSCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. Setting the CGU_TSCTL.LOCK bit locks this register.	
		0	Unlock
		1	Lock
7:4 (R/W)	TSDIV	Counter's Clock Divider. The CGU_TSCTL.TSDIV bit field divides SYSCLK by 2^{TSDIV}.	
		0-15	Divides SYSCLK by 2^TSDIV
1 (R/W)	LOAD	Load Counter. Writing one to the CGU_TSCTL.LOAD bit causes CoreSight Timestamp Counter to be loaded from the CGU_TSVALUE0 and CGU_TSVALUE1 registers.	
		0	Always read as "0"
0 (R/W)	EN	Counter Enable. The CGU_TSCTL.EN bit enables or disables the CoreSight Timestamp counter.	
		0	Counter Disabled
		1	Counter Enabled

Timestamp Counter Initial 32 LSB Value Register

The CGU_TSVALUE0 register holds the least significant bits (bits [31:0]) value that is initially loaded to the CoreSight timestamp counter.

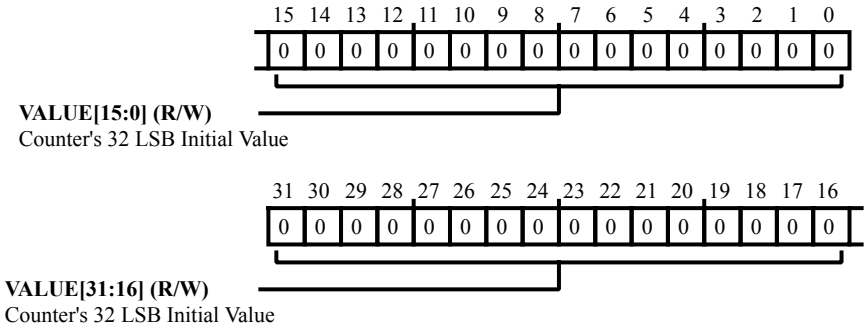

Figure 4-11: CGU_TSVALUE0 Register Diagram

Table 4-17: CGU_TSVALUE0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Counter's 32 LSB Initial Value. The CGU_TSVALUE0.VALUE bit field holds the LSBs value that is initially loaded to the CoreSight time stamp counter.

Time Stamp Counter Initial MSB Value Register

The CGU_TSVALUE1 register holds the most significant bits (bits [63:32]) value that is initially loaded to the CoreSight time stamp counter.

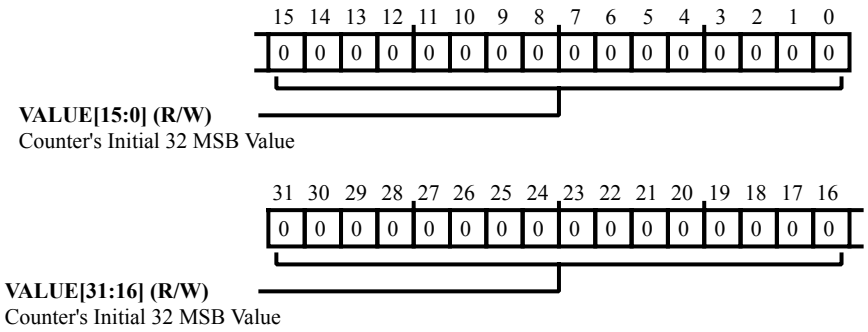

Figure 4-12: CGU_TSVALUE1 Register Diagram

Table 4-18: CGU_TSVALUE1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Counter's Initial 32 MSB Value. The CGU_TSVALUE1.VALUE bit field holds the MSBs value that is initially loaded to the CoreSight time stamp counter.

5 System Protection Unit (SPU)

The system protection unit (SPU) provides features for protecting system resources from errant writes. A number of protection categories are available. Each category contains a set of registers for protection.

In a system with multiple system MMR masters, configurations of peripherals can be changed unintentionally leading to bad data or even system malfunctions. The peripherals are shared resources in the system. The SPU restricts access to certain MMRs, similar to the functionality of a semaphore.

SPU Features

The SPU has the following features.

- Write-protect system MMR from certain system masters and core masters.
- Simultaneously lock multiple peripheral configuration registers through a global lock mechanism.
- Write-protect and block access to its own write-protection registers from other system masters.

SPU Functional Description

The following sections provide information on the function of the SPU.

ADSP-CM40x SPU Register List

The System Protection Unit (SPU) provides a set of registers that can protect system resources from errant writes. The protection categories are global lock (protects configuration registers) and write protect register lock (protects the write protect register). For more information on SPU functionality, see the SPU register descriptions.

Table 5-1: ADSP-CM40x SPU Register List

Name	Description
SPU_CTL	Control Register
SPU_STAT	Status Register
SPU_WP[n]	Write Protect Register n

Peripheral Register Write Protection

The SPU has a write-protection register (SPU_WP[n]) associated with each peripheral. Each of these write-protection registers has the exact same bits that correspond to a particular SMMR master (Core 0, Core 1, MDMA, for example). When the bits are set, the SPU locks the corresponding SMMR masters from accessing the register address space of the associated peripheral. The bits in the register can be cleared to allow access to the registers of the peripheral again. When the SPU intiates the write-protection register, any writes that are in-progress complete, before the SPU blocks subsequent writes.

In the ***SPU Write Protect Registers*** figure, each write-protect register in the SPU is associated with a particular peripheral.

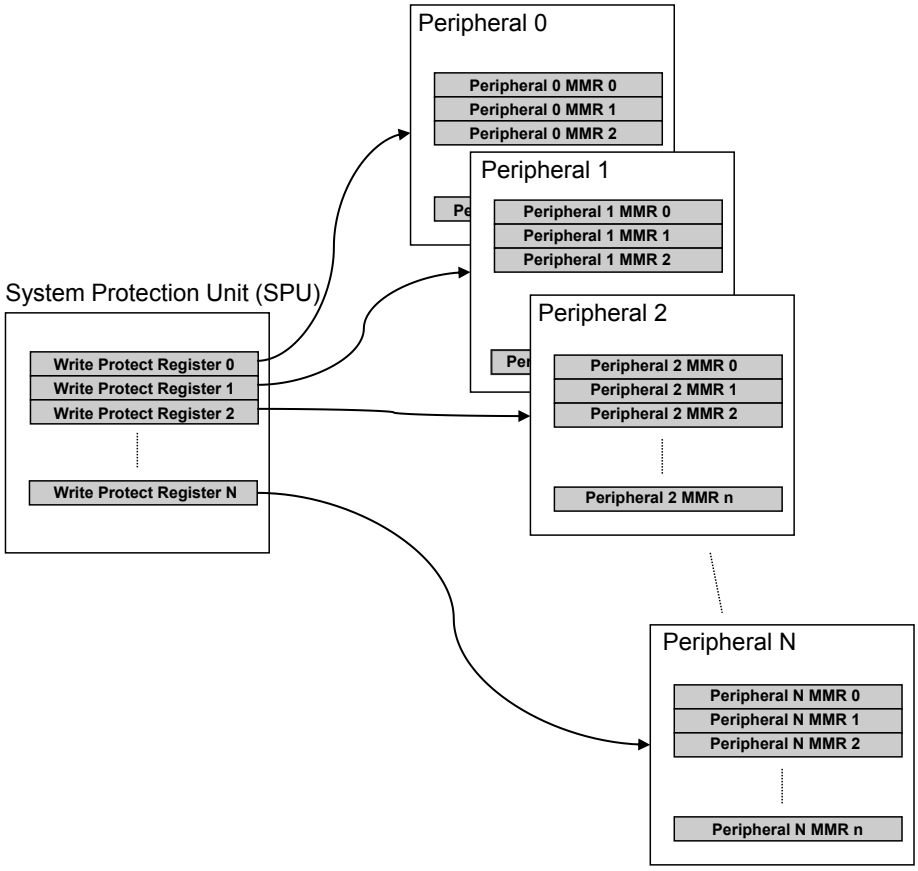

Figure 5-1: SPU Write Protect Registers

In the figure, a write-protect register in the SPU blocks write-attempts to the MMR space of the associated peripheral. The bits in the write-protect register specify from which masters to block write-access.

NOTE: A SPU write protection register (SPU_WP[n]) exists for the SPU alone. If all defined bits are set in this register for the SPU, any configurations in the SPU are locked and cannot be changed. Only a system reset can restore access to the SPU.

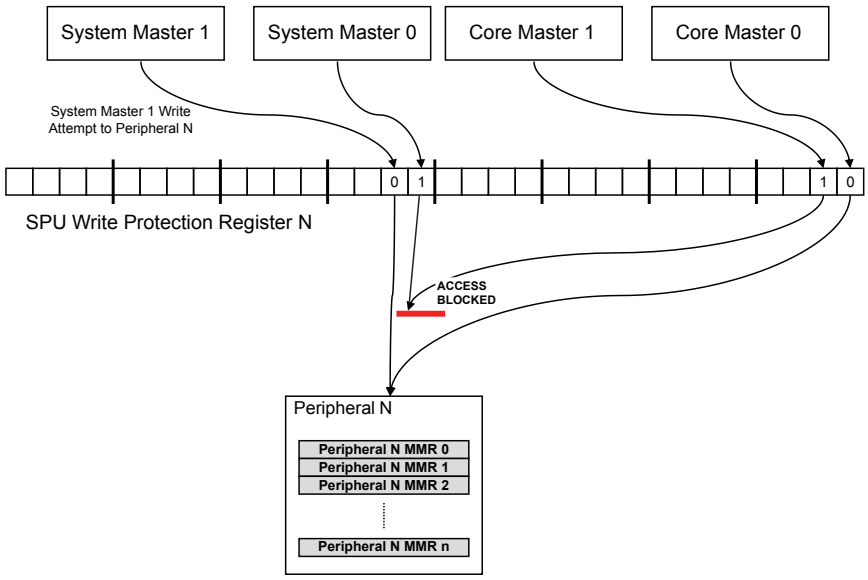

Figure 5-2: SPU Write-Protect Register Blocking Access from System Master 0 and Core Master 1

Global Locking

The SPU also has global locking capability. When enabled by setting `SPU_CTL.GLCK` bit field to a value other than 0xAD, a system-wide global lock signal is active. Some peripherals have a lock enable bit in their control register. When this bit is set, the peripheral recognizes the global lock signal and blocks further write-accesses to its own control register. Access to the configuration register of the peripheral is enabled when global lock is turned off in the SPU.

The ***Global Locking*** figure is a conceptual diagram. The diagram shows how the SPU (or any peripheral) blocks any write attempts to its control register when:

- The global lock signal from the SPU is active, and
- The global lock enable bit is set in the control register of the peripheral.

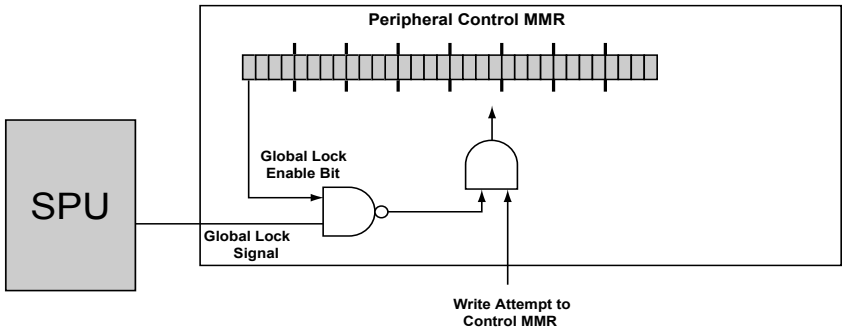

Figure 5-3: Global Locking

The SPU can write-protect its own registers. When the `SPU_CTL.WPLCK` bit is set and global locking is enabled, the SPU blocks accesses to the SPU write-protection registers. To enable write access to the write-protection registers in the SPU, disable the global locking.

SPU Block Diagram

The *SPU System-Level Block Diagram* shows a system-level block diagram of where the SPU is located in the system. It resides between the SMMR interface and the system crossbar. Depending on the configuration of the SPU write-protect registers, it can block access to some peripherals from certain SMMR masters.

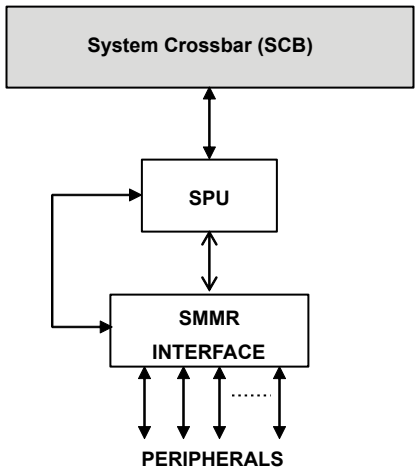

Figure 5-4: SPU System-Level Block Diagram

SPU Architectural Concepts

As shown in the block diagram, the SPU sits between the system crossbar (SCB) and the SMMR interface to the peripherals. The SPU gates any MMR access to any peripheral from any master that comes through the SCB. Depending on the configuration of the write-protection registers in the SPU, the SPU does or does not allow the MMR write to go through.

SPU Event Control

The system protection unit provides write-protection against MMRs peripherals and its own write-protect registers. If a write attempt is made to any locked MMR peripheral the SPU has write-protected, it blocks the write. The SPU generates a bus error to the master that attempted the write. That master does or does not generate an event, based on the returned error.

The SPU does not generate an event for blocked write attempts.

SPU Programming Model

The system protection unit (SPU) consists of write-protect registers. Each one corresponds to a different peripheral instance. Bits in the write-protect registers correspond to system masters that can modify the MMR contents of the peripherals. By writing to these write-protect registers, the corresponding memory-mapped registers of the peripheral are write-protected against masters whose bits in the write-protect register are set.

The SPU globally locks the control register of the peripheral. Peripherals that support this feature have a lock enable bit in their control register. The peripheral blocks any additional write attempts to its control register from any master when:

- The global lock signal is active from the SPU, and
- The lock enable bit of the peripheral is set

If the lock enable bit of a peripheral is not set and the global lock signal is active, access to that control register of the peripheral is still allowed. To grant access again, disable the global lock signal from the SPU by writing the value 0xAD into the `SPU_CTL.GLCK` bit field.

Another protection mechanism that the SPU offers is write-protection against the write-protection registers. If the write protect register lock bit (`SPU_CTL.WPLCK`) is set and the global lock signal is active, writes to the write-protect registers of the SPU are blocked. To reenable access to the write-protect registers in the SPU, deactivate the global lock signal by writing 0xAD into the `SPU_CTL.GLCK` bit field.

Enabling and Disabling the SPU

The SPU is always operating. There are no bits to enable or disable the SPU. The SPU configuration can be updated at any time. Any ongoing transactions finish before a new configuration is in effect. By default, the SPU does not write-protect any of the MMRs.

Write-Protecting the SPU

The SPU is treated like any other peripheral in the system. As such, the SPU also has an associated write-protection register. If this write-protection register is configured to block all writes from all masters, any SPU configuration remains the same until the next system reset.

SPU Mode Configuration

The SPU can provide address range-wide protection by write-protecting the peripherals MMR address range from system MMR masters. It can also provide register wide protection using global locking. Peripherals that support this feature can enable it in their respective configuration register. When the SPU enables the global lock signal, all subsequent writes to the configuration register of the peripheral are blocked until the global lock signal is deasserted. Similarly, the write-protection registers of the SPU can be write-protected using the global lock signal as well. The SPU uses all these modes of operation together.

Locking Write-Protect Registers

Use the following steps to lock (write-protect) a register.

1. Set the `SPU_CTL.WPLCK` bit and configure the `SPU_CTL.GLCK` field to something other than 0xAD.

The SPU write-protect registers are blocked from further write accesses.

Protecting a Peripheral

Use the following procedure to protect a peripheral.

1. Determine which peripheral needs protection and locate the corresponding write-protect register (SPU_WP[n]) in the SPU. See the "Write-Protect and Secure Peripheral Registers" section.
2. Determine the SMMR masters from which the peripheral needs protection. Then, set the corresponding bit or bits in the write-protect register (SPU_WP[n]) for the peripheral. See the "Write-Protect and Secure Peripheral Registers" section.

After setting the write-protect register for the particular peripheral, the identified SMMR masters are blocked from writing to any MMR in the address space of the peripheral. This block remains in place until the bits in the write-protect register are cleared.

ADSP-CM40x SPU Register Descriptions

System Protection Unit (SPU) contains the following registers.

Table 5-2: ADSP-CM40x SPU Register List

Name	Description
SPU_CTL	Control Register
SPU_STAT	Status Register
SPU_WP[n]	Write Protect Register n

Control Register

The SPU Control Register (SPU_CTL) provides a global lock for configuration registers as well as control for locking the Write Protect (SPU_WP[n]) registers. It also controls the generation of an interrupt to report blocked accesses.

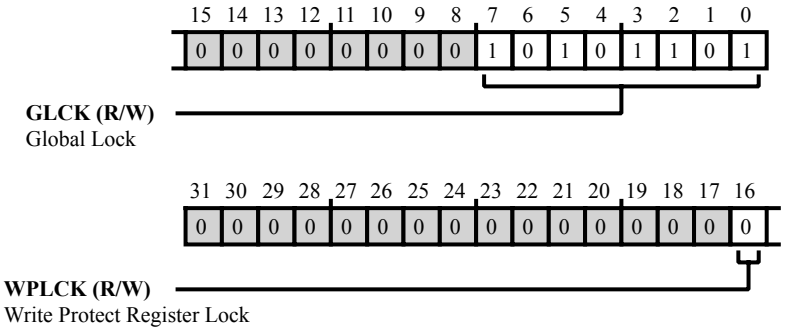

Figure 5-5: SPU_CTL Register Diagram

Table 5-3: SPU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	WPLCK	Write Protect Register Lock. When the SPU_CTL.WPLCK bit is set in combination with the SPU_CTL.GLCK bit, writes to the SPU's Write Protect Registers are blocked and return an error.	
		0	Disable
		1	Enable
7:0 (R/W)	GLCK	Global Lock. The SPU_CTL.GLCK controls the Global Lock signal. The Global Lock signal provides register-based write protection. Writing 0xAD to this field disables the lock, and writing any other value enables the lock.	

Status Register

The SPU_STAT register indicates if there have been any errors, active interrupts and Global Lock status.

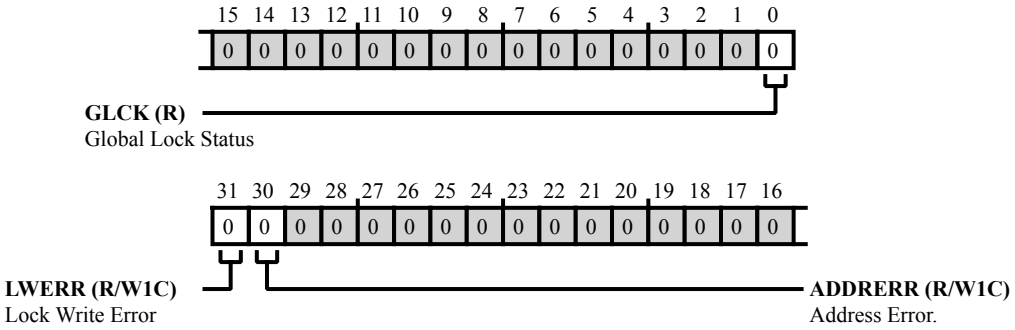

Figure 5-6: SPU_STAT Register Diagram

Table 5-4: SPU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W1C)	LWERR	Lock Write Error. The SPU_STAT.LWERR indicates whether there has been an attempted write to a register in the SPU with its lock bit (WPLCK or SCRLCK) set while SPU_CTL.GLCK was asserted. This bit can be cleared by writing a '1' to it.	
		0	Inactive
		1	Active
30 (R/W1C)	ADDRERR	Address Error.. The SPU_STAT.ADDRERR indicates whether there has been an attempted write to a read-only register or an access an invalid address in the SPU MMR address range. This bit can be cleared by writing a '1' to it.	
		0	Inactive
		1	Active
0 (R/NW)	GLCK	Global Lock Status. The SPU_STAT.GLCK indicates whether the Global Lock is enabled or disabled.	
		0	Disabled (global_lock=0)
		1	Enabled (global_lock=1)

Write Protect Register n

In the system, each SPU_WP[n] register is assigned to a specific MMR address range associated with one peripheral. When the appropriate bits are set, writes to the peripheral from a specific master are blocked and an error is returned to the master. For more information, see the processor specific additional information for the SPU_WP[n] register.

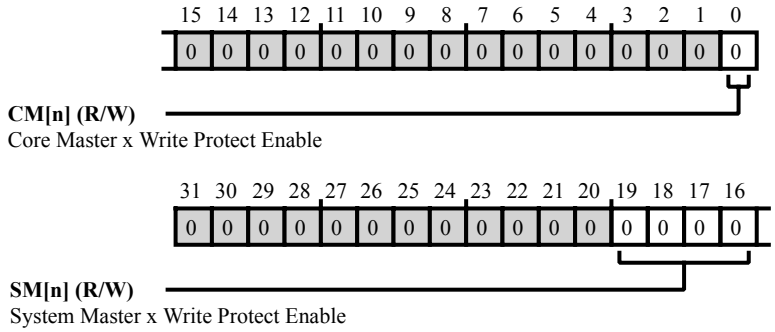

Figure 5-7: SPU_WP[n] Register Diagram

Table 5-5: SPU_WP[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:16 (R/W)	SM[n]	System Master x Write Protect Enable. The SPU_WP[n].SM[n] bits correspond to different system masters in the system. When a particular bit is set in this field, the corresponding system master cannot write to the corresponding peripheral's MMR address space. The write attempt will be blocked by the SPU.
0 (R/W)	CM[n]	Core Master x Write Protect Enable. The SPU_WP[n].CM[n] bits correspond to different cores in the system. When a particular bit is set in this field, the corresponding core cannot write to the corresponding peripheral's MMR address space. The write attempt will be blocked by the SPU.

ADSP-CM40x Write-Protect Registers

The SPU consists of a collection of write-protect registers each of which are associated with a specific peripheral or slave. The SPU for ADSP-CM40xx is configured with 59 write-protect registers. The ***SPU_WPn Registers and Related Peripherals*** table provides the write-protect register number for each peripheral.

Table 5-6: SPU_WPn Registers and Related Peripherals

Write-Protect Register Number (n)	Peripheral
0	TIMER0
1	TWI0
2	SPORT0 A

Table 5-6: SPU_WPn Registers and Related Peripherals (Continued)

Write-Protect Register Number (n)	Peripheral
3	SPORT0 B
4	SPORT1 A
5	SPORT1 B
6	CRC0
7	CAN0
8	CAN1
9	UART0
10	UART1
11	UART2
12	PORTA
13	PORTB
14	PORTC
15	PORTD
16	PORTE
17	PORTF
18	PADS0
19	PINT0
20	PINT1
21	PINT2
22	PINT3
23	PINT4
24	SMC0
25	EPWM0
26	EPWM1
27	EPWM2
28	CNT0
29	CNT1
30	CNT2
31	CNT3
32	SINC0
33	ADCC0

Table 5-6: SPU_WPn Registers and Related Peripherals (Continued)

Write-Protect Register Number (n)	Peripheral
34	DACC0
35	DMA1
36	DMA2
37	DMA3
38	DMA4
39	RCU0
40	TRU0
41	CGU0
42	DPM0
43	HAE0
44	EMAC0
45	SWU0 (SMC)
46	SWU1 (SPI2)
47	SWU2 (SPI0)
48	SWU3 (SRAM)
49	SWU4 (MMR)
50	SPU0
51	SEC0
52	JTAG
53	SPI0
54	SPI1
55	SPI2
56	WDT0
57	USB0
58	Reserved

For each processor, there are a different number of masters that can access the SMMR space. The ***SPU_WPn.CMn and SPU_WPn.SMn Bit Descriptions*** table shows which bits enable protection against a specific master.

Table 5-7: SPU_WPn.CMn and SPU_WPn.SMn Bit Descriptions

Bit No.	Bit Name	Description
0	CM0_WP (Core Master 0)	Core
16	SM0_WP (System Master 0)	MDMA0 Source

Table 5-7: SPU_WPn.CMn and SPU_WPn.SMn Bit Descriptions (Continued)

Bit No.	Bit Name	Description
17	SM1_WP (System Master 1))	MDMA0 Destination
18	SM2_WP (System Master 2)	MDMA1 Source
19	SM3_WP (System Master 3))	MDMA1 Destination

If global locking is enabled in the SPU control register, MMRs of certain peripherals can be write-protected by locking them in their control registers. Global lock is enabled by writing any value other than 0xAD to the `GLCK` field in SPU control register. The ***Slave Numbers, Protected Modules, and Global Locks*** table shows which peripheral can be write protected with this feature.

Table 5-8: Slave Numbers, Protected Modules, and Global Locks

Slave Number	Protected Module	Global Lock Enable Register
1	GPIO (PORTs A-F)	Port Lock Register
2	SEC0	Fault Control Register
3	TRU0	TRU Slave Select Register
4	CGU0	CGU Control Register
5	DPM0	DPM Control Register
6	RCU0	RCU Control Register

6 Dynamic Power Management (DPM)

The dynamic power management (DPM) unit of the processor controls transitions between different power-saving modes.

DPM Features

The DPM allows programs to control the power mode of the processor as follows.

- Permits operation of multiple, external wake-up sources.

DPM Functional Description

The DPM can be programmed to transition between power modes.

ADSP-CM40x DPM Interrupt List

Table 6-1: ADSP-CM40x DPM Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
113	DPM0_EVT	DPM0 Event	Level	

ADSP-CM40x DPM Register List

A set of registers govern DPM operations. For more information on DPM functionality, see the DPM register descriptions.

Table 6-2: ADSP-CM40x DPM Register List

Name	Description
DPM_CCBF_DIS	Core Clock Buffer Disable Register
DPM_CCBF_EN	Core Clock Buffer Enable Register
DPM_CCBF_STAT	Core Clock Buffer Status Register
DPM_CCBF_STAT_STKY	Core Clock Buffer Status Sticky Register

Table 6-2: ADSP-CM40x DPM Register List (Continued)

Name	Description
DPM_CTL	Control Register
DPM_SCBF_DIS	System Clock Buffer Disable Register
DPM_STAT	Status Register
DPM_WAKE_EN	Wakeup Enable Register
DPM_WAKE_POL	Wakeup Polarity Register
DPM_WAKE_STAT	Wakeup Status Register

DPM Definitions

To make the best use of the DPM, it is useful to understand the following terms.

Active mode

A power-saving mode in which the PLL is bypassed but still enabled.

Active mode with PLL disabled

A power-saving mode in which the PLL is bypassed and disabled.

CGU

Acronym for the clock generation unit (CGU), which is comprised of the PLL and PCU

Deep sleep mode

A power-saving mode in which all CCLKs are gated.

DPM

Acronym for the dynamic power management (DPM) controller.

Full-on mode

The normal operating mode in which all clock domains are derived from the PLL.

PCU

Acronym for the PLL control unit (PCU).

PLL

Acronym for the phase-locked loop (PLL).

RCU

Acronym for the reset control unit (RCU).

DPM Operating Modes

The DPM includes several operating modes. The modes are:

- RESET
- FULL-ON
- ACTIVE
- ACTIVE with PLL Disabled
- DEEP SLEEP

The ***Operating Modes and Transitions*** figure shows the relationships between DPM modes for the ADSP-CM40x processor.

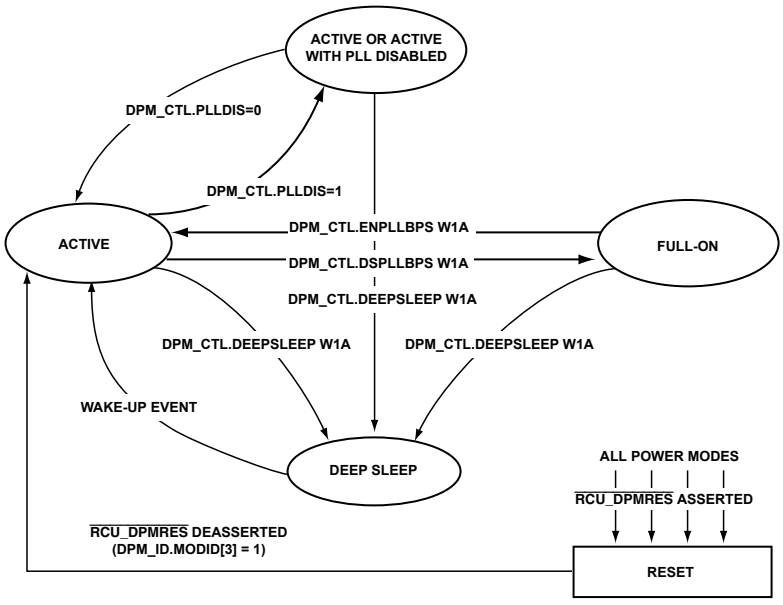

Figure 6-1: Operating Modes and Transitions

Reset State

Reset is the initial state of the processor and is the result of a hardware or software triggered event. The DPM itself does not trigger entering reset. The external SYS_HWRST pin or the RCU triggers entering reset. The DPM responds to reset by transitioning to its default state.

From RESET, the DPM always transitions to ACTIVE state.

Full-on Mode

In full-on mode, the processor can reach its maximum clock rate and power dissipation can be at its highest. The DPM transitions from full-on mode to:

- Active mode if the DPM_CTL.PLLBPST bit is set
- Deep sleep mode if the DPM_CTL.DEEPSLEEP bit is set

Active Mode

Active mode is the default state of the DPM after RESET.

In active mode, power dissipation is reduced on the VDD_INT power domain (compared to full-on mode) by bypassing the PLL and running clock domains at the SYS_CLKIN pin frequency. The processor is fully functional. The DPM transitions from active mode to:

- Full-on mode if the DPM_CTL.PLLBPCL bit is set
- Active with PLL disabled mode if the DPM_CTL.PLLDIS bit is set
- Deep sleep mode if the DPM_CTL.DEEPSLEEP bit is set

ACTIVE with PLL Disabled

In active with PLL disabled mode, power dissipation is reduced on the VDD_INT power domain (compared to active mode) by disabling the PLL. Running all units at the SYS_CLKIN pin frequency also reduces the power domain. The processor is fully functional. The DPM transitions from active with PLL disabled mode to:

- Active mode if the DPM_CTL.PLLDIS bit is cleared
- Deep sleep mode if the DPM_CTL.DEEPSLEEP bit is set

Deep Sleep Mode

To enter deep sleep mode, the processor sets the DPM_CTL.DEEPSLEEP bit. All processor cores are in idle state. The program software must guarantee that system transfers, including DMA, are stopped before each processor core goes into idle state and the processor enters deep sleep mode. In this state, power dissipation on the VDD_INT0 power domain is reduced by gating all the core and system clocks and disabling the PLL.

The enabled hardware wake-up signals or a hardware reset signal make the processor exit deep sleep mode. The DPM_WAKE_EN.WS[n] bits and DPM_WAKE_POL.WS[n] bits work together to determine which hardware wake-up signals are enabled and the polarity of the signals. Wake-up signal assertion is latched only when the signal is enabled. The enabled wake-up signal assertion occurring first is recorded in the DPM_WAKE_STAT register.

NOTE: To see which wake-up sources the processor reports, see ADSP-CM40x Wake-Up Sources.

When a wake-up occurs, the DPM does the following:

- Signals a DPM event interrupt to the SEC

- Transitions to ACTIVE mode
- Enables all clock domains that are not disabled in the DPM_SCBF_DIS register

The DPM event interrupt stays active until the program clears any bits that are set in the DPM_WAKE_STAT register. The DPM event interrupt is the first indication that the processor has exited DEEP SLEEP.

One option for waking up the core is to enable the CGU event interrupt, which asserts after the PLL locks.

Another option is to use the DPM event interrupt to make a core exit idle and to enable the corresponding core clock buffer.

DPM Event Control

The DPM event is triggered when an enabled wake-up is asserted. The DPM generates bus errors when a misaligned access to a register occurs. It also generates errors when an attempt is made to access unused DPM address space or a write-protected register.

DPM Events

The DPM event interrupt is triggered when any bit in the DPM_WAKE_STAT register is set, indicating that an enabled wake-up was asserted. The DPM event interrupt stays active until the user clears any bits that are set in the DPM_WAKE_STAT register.

DPM Errors

The DPM generates a bus error when a read or write transaction is attempted to an unused address within the DPM address range. It also generates a bus error when a misaligned access is made to a DPM register. In addition to the bus error, the DPM sets the DPM_STAT.ADDRERR bit.

If a write to a write-protected DPM register is attempted, the DPM generates a bus error. In addition, the DPM sets the DPM_STAT.LWERR bit.

DPM Programming Model

The following sections describe programming techniques, including verifying restoration of power supplies, managing power modes, and selecting wake-up sources.

Configuring Deep Sleep Mode

The mode for deep sleep gates all core and system clocks to save power.

The processor enters deep sleep mode from any state in which software can run. Reading the DPM_STAT.CURMODE field reveals the current power mode. Clocks do not stop immediately after entry to deep sleep mode is requested, but no further action is required to guarantee that the mode transition occurs.

The processor cores must be idle before the clocks are shut down.

1. If the DPM_STAT.CURMODE bit indicates full-on mode, wait for the CGU_STAT.PLLBP bit =0, the CGU_STAT.PLOCK bit =1, and the CGU_STAT.CLKSALGN bit =0.
2. If the DPM_STAT.CURMODE bit indicates active mode with PLL disabled, wait for the CGU_STAT.PLLEN bit =0.
3. Enable the DPM event interrupt to wake up the desired core, directing exit from idle after exit from deep sleep mode.
4. Set the polarity of wake-up sources as needed with the DPM_WAKE_POL.WS[n] bits.
5. Enable the wake-up sources as needed with the DPM_WAKE_EN.WS[n] bits.
6. Set the DPM_CTL.DEEPSLEEP bit.
7. Clear all pending core transactions, DMA transactions, and interrupts. If applicable, use a system synchronization instruction.
8. Place all processor cores in idle state.

The processor is now in deep sleep mode. To wake the processor, assert any of the enabled wake-up sources.

ADSP-CM40x Wake-Up Sources

The *ADSP-CM40x DEEP SLEEP Wake-up Sources* table shows the DEEP SLEEP wake-up sources for the ADSP-CM40x. The first column shows the wake-up source bit (*WSn*) used in the DPM_WAKE_EN, DPM_WAKE_POL, and the DPM_WAKE_STAT registers. The *Assigned Source* column shows which peripheral or pin source is assigned to the (*WSn*) bit. Peripherals in parentheses mean that the source can either be used as a GPIO wake-up or as the specific peripheral wake-up listed. The *Deep Sleep* column indicates whether the wake-up source can wake the processor from deep sleep mode.

Table 6-3: ADSP-CM40x DEEP SLEEP Wake-up Sources

DPM_WAKE_EN, DPM_WAKE_POL, and DPM_WAKE_STAT Bit	Assigned Source	DEEP SLEEP
WS0	PC_06	Yes
WS1	PC_07	Yes
WS2	PB_14	Yes
WS3	PB_13	Yes
WS4	JTAG_WK_UP	Yes
WS30: WS5	Reserved	N/A

ADSP-CM40x Clock Buffer Disable Bit Assignments

The *ADSP-CM40x Bit Assignments for DPM_SCBF_DIS* table shows the clock buffers that can be disabled on the ADSP-CM40x processor.

The first column shows the system clock buffer bit (DPM_SCBF_DIS.SCBF[n]) used in the DPM_SCBF_DIS register. The *Assigned Clock* column shows the clock buffer assigned to the DPM_SCBF_DIS.SCBF[n] bit.

The *ADSP-CM40x Bit Assignments for DPM_SCBF_DIS* table shows the clock buffers that can be disabled on the ADSP-CM40x processor. The first column shows the system clock buffer disable bit (DPM_SCBF_DIS.SCBF[n]) used in the DPM_SCBF_DIS register. The *Assigned Clock* column shows the clock buffer assigned to the DPM_SCBF_DIS.SCBF[n] bit.

Table 6-4: ADSP-CM40x Bit Assignments for DPM_SCBF_DIS

DPM_SCBF_DIS Bit	Assigned Clock
SCBF0	USBCLK
SCBF1	OUTCLK

ADSP-CM40x DPM Register Descriptions

Dynamic Power Management (DPM) contains the following registers.

Table 6-5: ADSP-CM40x DPM Register List

Name	Description
DPM_CCBF_DIS	Core Clock Buffer Disable Register
DPM_CCBF_EN	Core Clock Buffer Enable Register
DPM_CCBF_STAT	Core Clock Buffer Status Register
DPM_CCBF_STAT_STKY	Core Clock Buffer Status Sticky Register
DPM_CTL	Control Register
DPM_SCBF_DIS	System Clock Buffer Disable Register
DPM_STAT	Status Register
DPM_WAKE_EN	Wakeup Enable Register
DPM_WAKE_POL	Wakeup Polarity Register
DPM_WAKE_STAT	Wakeup Status Register

Core Clock Buffer Disable Register

The DPM_CCBF_DIS register controls the core n clock buffers. The number of clocks varies with the processor design, with bit 0 corresponding to CCLK0, bit 1 corresponding to CCLK1, and so on. This register includes a write protection lock.

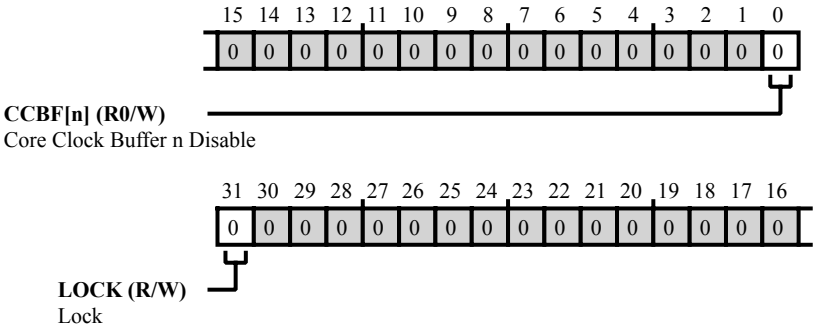

Figure 6-2: DPM_CCBF_DIS Register Diagram

Table 6-6: DPM_CCBF_DIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock.	
		0	Unlock
		1	Lock
0 (R0/W)	CCBF[n]	Core Clock Buffer n Disable. The DPM_CCBF_DIS.CCBF[n] bits provide a core clock buffer disable for each core on the processor with bit 0 corresponding to CCLK0, bit 1 corresponding to CCLK1, and so on.	
		0	No Action
		1	Disable Buffer

Core Clock Buffer Enable Register

The DPM_CCBF_EN register controls the core n clock buffers. The number of clocks varies with the processor design, with bit 0 corresponding to CCLK0, bit 1 corresponding to CCLK1, and so on. This register includes a write protection lock.

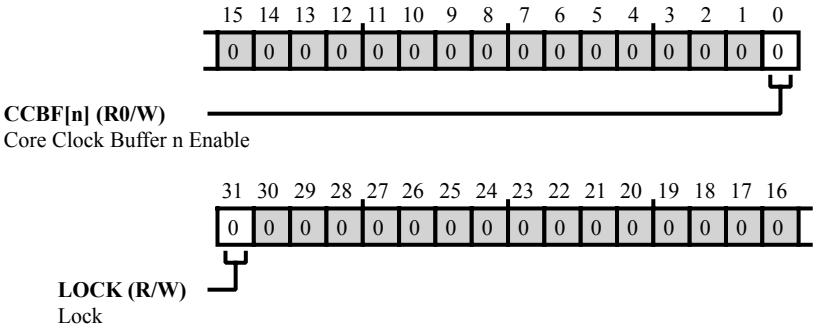

Figure 6-3: DPM_CCBF_EN Register Diagram

Table 6-7: DPM_CCBF_EN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the DPM_CCBF_EN.LOCK bit is set, the DPM_CCBF_EN register is read only (locked).	
		0	Unlock
		1	Lock
0 (R0/W)	CCBF[n]	Core Clock Buffer n Enable. The DPM_CCBF_EN.CCBF[n] bits provide a core clock buffer enable for each core on the processor with bit 0 corresponding to CCLK0, bit 1 corresponding to CCLK1, and so on.	
		0	No Action
		1	Enable Buffer

Core Clock Buffer Status Register

The DPM_CCBF_STAT register indicates core clock buffer enable or disable status for each core on the processor, with bit 0 corresponding to CCLK0.

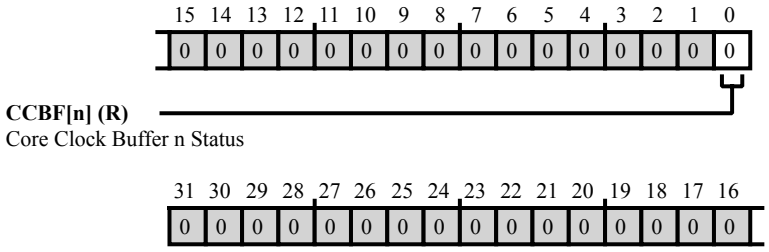

Figure 6-4: DPM_CCBF_STAT Register Diagram

Table 6-8: DPM_CCBF_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/NW)	CCBF[n]	Core Clock Buffer n Status. The DPM_CCBF_STAT.CCBF[n] bits indicates core clock buffer enabled or disabled status for each core on the processor with bit 0 corresponding to CCLK0.	
		0	Buffer Enabled
		1	Buffer Disabled

Core Clock Buffer Status Sticky Register

The DPM_CCBF_STAT_STKY register indicates core n clock buffer enable or disable sticky status for each core on the processor, with bit 0 corresponding to CCLK0, bit 1 corresponding to CCLK1, and so on.

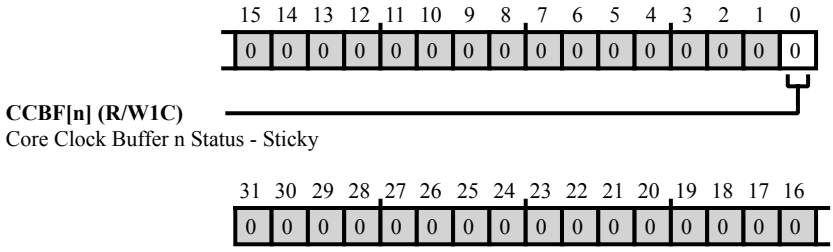

Figure 6-5: DPM_CCBF_STAT_STKY Register Diagram

Table 6-9: DPM_CCBF_STAT_STKY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W1C)	CCBF[n]	Core Clock Buffer n Status - Sticky. The DPM_CCBF_STAT_STKY.CCBF[n] bits indicates core clock buffer enabled or disabled sticky status for each core on the processor with bit 0 corresponding to CCLK0, bit 1 corresponding to CCLK1, and so on. The sticky status shows that the status was set since the last time the bit was cleared with a W1A or reset.	
		0	Buffer Enabled - Sticky
		1	Buffer Disabled - Sticky

Control Register

The DPM_CTL register controls sleep modes selections and PLL operations of the DPM. A write protect feature permits locking out changes to this register.

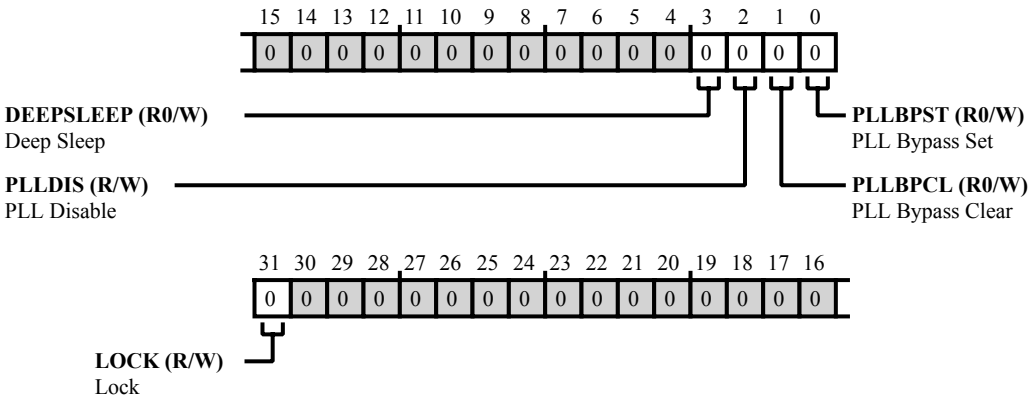

Figure 6-6: DPM_CTL Register Diagram

Table 6-10: DPM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the DPM_CTL.LOCK bit is set, the DPM_CTL register is read only (locked).	
		0	Unlock
		1	Lock
3 (R0/W)	DEEPSLEEP	Deep Sleep. The DPM_CTL.DEEPSLEEP bit puts the processor into deep sleep mode. The processor stays in this mode until a wakeup event occurs. For more information about DPM modes, see the functional description.	
		0	No Action
		1	Deep Sleep
2 (R/W)	PLLDIS	PLL Disable. While the DPM is in active mode, it is possible to disable the PLL with the DPM_CTL.PLLDIS bit, keeping the DPM active and running with lower power consumption. For more information about DPM modes, see the operating modes.	
		0	Enable
		1	Disable

Table 6-10: DPM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R0/W)	PLLBPCL	PLL Bypass Clear. While the DPM is in active mode, it is possible to disable the PLL bypass with the `DPM_CTL.PLLBPCL` bit, transitioning to DPM to full-on mode. For more information about DPM modes, see the operating modes.	
		0	No action
		1	Disable PLL Bypass
0 (R0/W)	PLLBPST	PLL Bypass Set. While the DPM is in full on mode, it is possible to enable the PLL bypass with the `DPM_CTL.PLLBPST` bit, transitioning the DPM to active mode. For more information about DPM modes, see the operating modes.	
		0	No action
		1	Enable PLL Bypass

System Clock Buffer Disable Register

The DPM_SCBF_DIS register controls the system n clock buffers. The number of clocks varies with the processor design. See the Clock Buffer Disable Bit Assignments topic for the bit assignments of this processor.

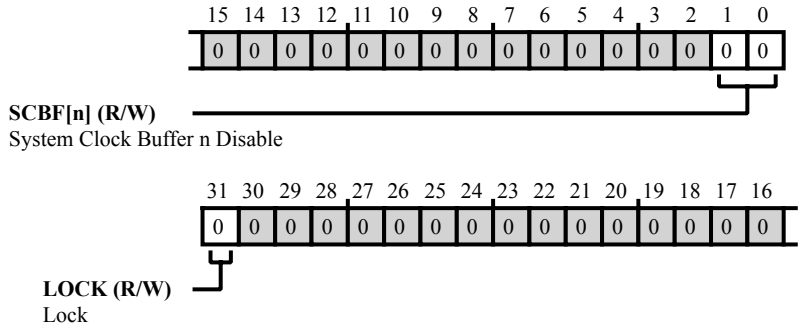

Figure 6-7: DPM_SCBF_DIS Register Diagram

Table 6-11: DPM_SCBF_DIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the DPM_SCBF_DIS.LOCK bit is set, the DPM_SCBF_DIS register is read only (locked).	
		0	Unlock
		1	Lock
1:0 (R/W)	SCBF[n]	System Clock Buffer n Disable. The DPM_SCBF_DIS.SCBF[n] bits provide a system clock buffer enable for each system clock domain on the processor. See the Clock Buffer Disable Bit Assignments topic for the bit assignments of this processor.	
		0	Enable Buffer
		3	Disable Buffer

Status Register

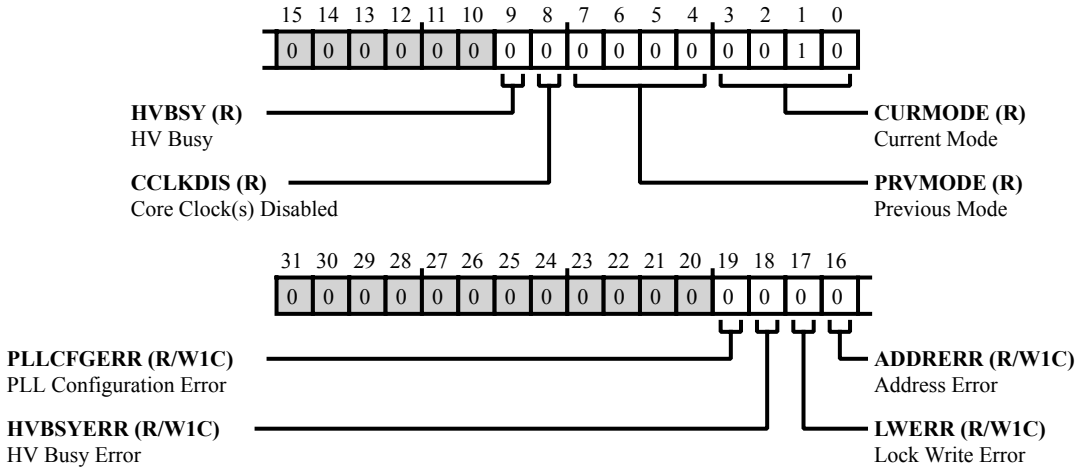

Figure 6-8: DPM_STAT Register Diagram

Table 6-12: DPM_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W1C)	PLLCFGERR	PLL Configuration Error.	
		0	Inactive
		1	Active
18 (R/W1C)	HVBSYERR	HV Busy Error. Reading registers during restore of DPM-LV from DPM-HV.	
		0	Inactive
		1	Active
17 (R/W1C)	LWERR	Lock Write Error.	
		0	Inactive
		1	Active
16 (R/W1C)	ADDRERR	Address Error.	
		0	Inactive
		1	Active
9 (R/NW)	HVBSY	HV Busy.	
		0	Not Busy (ready)
		1	Busy

Table 6-12: DPM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	CCLKDIS	Core Clock(s) Disabled. One or more of the core clocks disabled.	
		0	Inactive
		1	Active
7:4 (R/NW)	PRVMODE	Previous Mode.	
		0	Reset
		1	Full-On
		2	Active
		3	Active with PLL disabled
		4	Deep Sleep
		5-15	Reserved
3:0 (R/NW)	CURMODE	Current Mode.	
		0	Reserved
		1	Full-On
		2	Active
		3	Active with PLL disabled
		4-15	Reserved

Wakeup Enable Register

The DPM_WAKE_EN register enables the wakeup event sources for exiting deep sleep mode. The number of wakeup sources varies with the processor design, with bit 0 corresponding to wakeup source 0, bit 1 corresponding to wakeup source 1, and so on. This register includes a write protection lock. For information about wakeup source assignments, see the DPM Wakeup Sources topic.

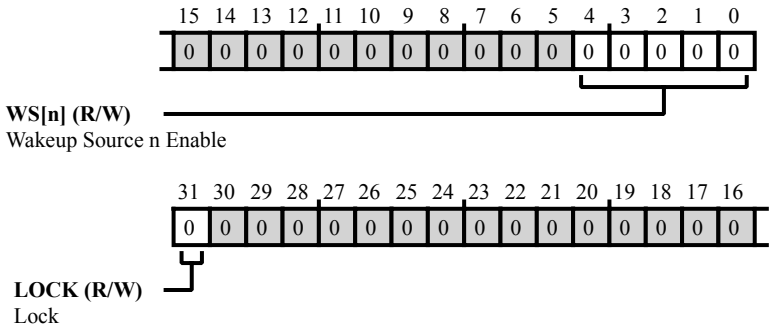

Figure 6-9: DPM_WAKE_EN Register Diagram

Table 6-13: DPM_WAKE_EN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the DPM_WAKE_EN.LOCK bit is set, the DPM_WAKE_EN register is read only (locked).	
		0	Unlock
		1	Lock
4:0 (R/W)	WS[n]	Wakeup Source n Enable. The DPM_WAKE_EN.WS[n] bits enable wakeup sources for exiting deep sleep mode, with bit 0 corresponding to wakeup source 0, bit 1 corresponding to wakeup source 1, and so on. For information about wakeup source assignments, see the DPM Wakeup Sources topic.	

Wakeup Polarity Register

The DPM_WAKE_POL register select polarity (active high or low) of the wakeup event sources for exiting deep sleep mode. The number of wakeup sources varies with the processor design, with bit 0 corresponding to wakeup source 0, bit 1 corresponding to wakeup source 1, and so on. This register includes a write protection lock. For information about wakeup source assignments, see the DPM Wakeup Sources topic.

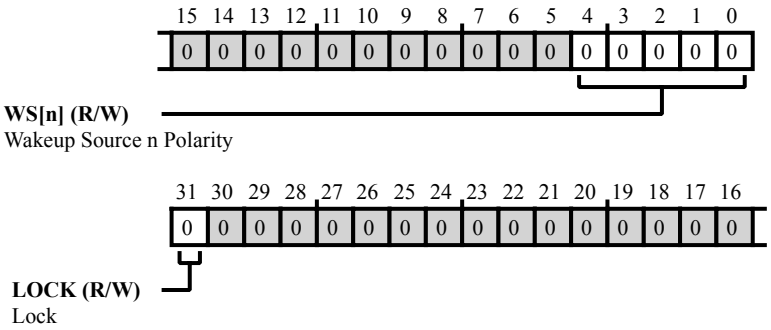

Figure 6-10: DPM_WAKE_POL Register Diagram

Table 6-14: DPM_WAKE_POL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the DPM_WAKE_POL.LOCK bit is set, the DPM_WAKE_POL register is read only (locked).	
		0	Unlock
		1	Lock
4:0 (R/W)	WS[n]	Wakeup Source n Polarity. The DPM_WAKE_POL.WS[n] bits select polarity (active high or low) of wakeup sources for exiting deep sleep mode, with bit 0 corresponding to wakeup source 0, bit 1 corresponding to wakeup source 1, and so on. For information about wakeup source assignments, see the DPM Wakeup Sources topic.	
		0	Low Active Wakeup
		31	High Active Wakeup

Wakeup Status Register

The DPM_WAKE_STAT register indicates the enabled and active status of the wakeup event sources for exiting deep sleep mode. The number of wakeup sources varies with the processor design, with bit 0 corresponding to wakeup source 0, bit 1 corresponding to wakeup source 1, and so on. For information about wakeup source assignments, see the DPM Wakeup Sources topic.

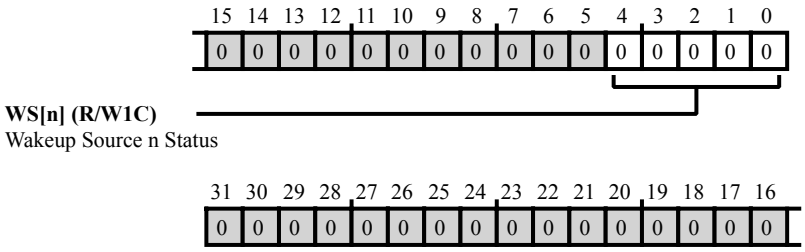

Figure 6-11: DPM_WAKE_STAT Register Diagram

Table 6-15: DPM_WAKE_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4:0 (R/W1C)	WS[n]	Wakeup Source n Status. The DPM_WAKE_STAT.WS[n] bits indicate the enabled and active status of wakeup sources for exiting deep sleep mode, with bit 0 corresponding to wakeup source 0, bit 1 corresponding to wakeup source 1, and so on. For information about wakeup source assignments, see the DPM Wakeup Sources topic.	
		0	No Status
		31	Enabled and Active

7 System Event Controller (SEC)

System event management is the responsibility of the system event controller (SEC).

System event management is the responsibility of the SEC. It is comprised of a Cortex M4F Nested Vectored Interrupt Controller (NVIC) and the System Fault Interface (SFI) to perform efficient event management.

SEC Features

The following list describes the system event controller features.

- NVIC supports
  - A programmable priority level of 0-16 for each interrupt
  - Level and pulse detection of interrupt signals.
  - Dynamic reprioritizing of interrupts. Grouping of priority values into group priority and subpriority fields.
  - Interrupt tail-chaining.
- System Fault Interface Supports
  - Fault action configuration, timeout, external indication, and system reset.
  - Fault from external event through GPIO

SEC Functional Description

The following sections provide a functional description of the SEC.

ADSP-CM40x SEC Register List

The System Event Controller (SEC) manages the system interrupt and/or system fault sources, including control features such as enable/disable, prioritization, and active/pending source status. For more information on SEC functionality, see the SEC register descriptions.

Table 7-1: ADSP-CM40x SEC Register List

Name	Description
SEC_FCOPP	Fault COP Period Register
SEC_FCOPP_CUR	Fault COP Period Current Register
SEC_FCTL	Fault Control Register
SEC_FDLY	Fault Delay Register
SEC_FDLY_CUR	Fault Delay Current Register
SEC_FEND	Fault End Register
SEC_FSID	Fault Source ID Register
SEC_FSRDLY	Fault System Reset Delay Register
SEC_FSRDLY_CUR	Fault System Reset Delay Current Register
SEC_FSTAT	Fault Status Register
SEC_GCTL	Global Control Register
SEC_GSTAT	Global Status Register
SEC_RAISE	Global Raise Register
SEC_SCTL[n]	Source Control Register n
SEC_SSTAT[n]	Source Status Register n

ADSP-CM40x Interrupt List

Table 7-2: ADSP-CM40x Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
-15	M4_SCS0_RESET	M4_SCS0 Reset		
-14	M4_SCS0_NonMaskableint	M4_SCS0 Non-maskable Interrupt		
-13	M4_SCS0_HardFault	M4_SCS0 Unmanaged Fault		
-12	M4_SCS0_MemoryManage-ment	M4_SCS0 MPU Fault		
-11	M4_SCS0_BusFault	M4_SCS0 Crossbar Fault	Edge	
-10	M4_SCS0_UsageFault	M4_SCS0 Instruction/Privilege Fault		
-5	M4_SCS0_SVCall	M4_SCS0 Service Call		
-4	M4_SCS0_DebugMonitor	M4_SCS0 Debug Monitor		
-2	M4_SCS0_PendSV	M4_SCS0 Pending Service		
-1	M4_SCS0_SysTick	M4_SCS0 System Time Tick		
0	OSCW_EVT	Oscillator Watchdog Error		

Table 7-2: ADSP-CM40x Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
1	CGU0_EVT	CGU0 PLL Lock Count Expired	Edge	
2	M4P0_L1CC_PERR	M4P0 L1 Cache Code Parity Error		
3	M4P0_CORE_SRAM_PERR	M4P0 SRAM Core Parity Error	Level	
4	M4P0_DMA_SRAM_PERR	M4P0 SRAM DMA Parity Error		
5	M4P0_BUS_FAULT	M4P0 Bus Fault	Edge	
6	M4P0_LOCKUP	M4P0 Lockup Error (Fault only; not an interrupt)	Level	
7	M4P0_SRAM_PERR_FLT	M4P0 SRAM Parity Error (Fault only; not an interrupt)		
8	WDOG0_EXP	WDOG0 Expiration	Level	
9	SEC0_ERR	SEC0 Fault Interrupt	Level	
10	SYS_ECT_EVT0	Embedded Cross Trigger Event 0		
11	SYS_ECT_EVT1	Embedded Cross Trigger Event 1		
12	PWM0_TRIP	PWM0 Trip Occurred	Level	
13	PWM1_TRIP	PWM1 Trip Occurred	Level	
14	PWM2_TRIP	PWM2 Trip Occurred	Level	
15	PWM0_SYNC	PWM0 PWMTMR Group Interrupt	Edge	
16	PWM1_SYNC	PWM1 PWMTMR Group Interrupt	Edge	
17	PWM2_SYNC	PWM2 PWMTMR Group Interrupt	Edge	
18	PINT0_BLOCK	PINT0 Block Interrupt Generated	Level	
19	PINT1_BLOCK	PINT1 Block Interrupt Generated	Level	
20	PINT2_BLOCK	PINT2 Block Interrupt Generated	Level	
21	PINT3_BLOCK	PINT3 Block Interrupt Generated	Level	
22	PINT4_BLOCK	PINT4 Block Interrupt Generated	Level	
23	SPORT0_A_DMA_ERR	SPORT0 DMA Channel 0 Error	Level	
24	SPORT0_B_DMA_ERR	SPORT0 DMA Channel 1 Error	Level	
25	SPI0_TXDMA_ERR	SPI0 DMA Channel 2 Error	Level	
26	SPI0_RXDMA_ERR	SPI0 DMA Channel 3 Error	Level	
27	UART0_TXDMA_ERR	UART0 DMA Channel 4 Error	Level	
28	UART0_RXDMA_ERR	UART0 DMA Channel 5 Error	Level	
29	SPORT1_A_DMA_ERR	SPORT1 DMA Channel 6 Error	Level	

Table 7-2: ADSP-CM40x Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
30	SPORT1_B_DMA_ERR	SPORT1 DMA Channel 7 Error	Level	
31	SPI1_TXDMA_ERR	SPI1 DMA Channel 8 Error	Level	
32	SPI1_RXDMA_ERR	SPI1 DMA Channel 9 Error	Level	
33	UART1_TXDMA_ERR	UART1 DMA Channel 10 Error	Level	
34	UART1_RXDMA_ERR	UART1 DMA Channel 11 Error	Level	
35	UART2_TXDMA_ERR	UART2 DMA Channel 12 Error	Level	
36	UART2_RXDMA_ERR	UART2 DMA Channel 13 Error	Level	
37	HAE0_RXDMA_CH0_ERR	HAE0 DMA Channel 14 Error	Level	
38	HAE0_RXDMA_CH1_ERR	HAE0 DMA Channel 15 Error	Level	
39	HAE0_TXDMA_ERR	HAE0 DMA Channel 16 Error	Level	
40	Reserved	Reserved	Reserved	Reserved
41	SPI0_ERR	SPI0 Error	Level	
42	SPI1_ERR	SPI1 Error	Level	
43	SPI2_ERR	SPI2 Error	Level	
44	ADCC0_ERR	ADCC0 Error	Level	
45	DACC0_ERR	DACC0 DAC Error	Level	
46	SYS_MDMA0_SRC_ERR	DMA Channel 17 Error	Level	
47	SYS_MDMA0_DST_ERR	DMA Channel 18 Error	Level	
48	SYS_MDMA1_SRC_ERR	DMA Channel 19 Error	Level	
49	SYS_MDMA1_DST_ERR	DMA Channel 20 Error	Level	
50	TIMER0_STAT	TIMER0 Status	Level	
51	TIMER0_TMR0	TIMER0 Timer 0 Expiration or Event	Level	
52	TIMER0_TMR1	TIMER0 Timer 1 Expiration or Event	Level	
53	TIMER0_TMR2	TIMER0 Timer 2 Expiration or Event	Level	
54	TIMER0_TMR3	TIMER0 Timer 3 Expiration or Event	Level	
55	TIMER0_TMR4	TIMER0 Timer 4 Expiration or Event	Level	
56	TIMER0_TMR5	TIMER0 Timer 5 Expiration or Event	Level	
57	TIMER0_TMR6	TIMER0 Timer 6 Expiration or Event	Level	
58	TIMER0_TMR7	TIMER0 Timer 7 Expiration or Event	Level	
59	CNT0_STAT	CNT0 Count Status	Level	
60	CNT1_STAT	CNT1 Count Status	Level	

Table 7-2: ADSP-CM40x Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
61	CNT2_STAT	CNT2 Count Status	Level	
62	CNT3_STAT	CNT3 Count Status	Level	
63	SPORT0_A_STAT	SPORT0 Channel A Status	Level	
64	SPORT0_A_DMA	SPORT0 Channel A DMA Transfer Complete	Level	0
65	SPORT0_B_STAT	SPORT0 Channel B Status	Level	
66	SPORT0_B_DMA	SPORT0 Channel B DMA Transfer Complete	Level	1
67	UART0_STAT	UART0 Status	Level	
68	UART0_TXDMA	UART0 Transmit DMA Transfer Complete	Level	4
69	UART0_RXDMA	UART0 Receive DMA Transfer Complete	Level	5
70	SINC0_STAT	SINC0 Status	Level	
71	SPI0_STAT	SPI0 Status	Level	
72	SPI0_TXDMA	SPI0 TX DMA Channel Transfer Complete	Level	2
73	SPI0_RXDMA	SPI0 RX DMA Channel Transfer Complete	Level	3
74	SPORT1_A_STAT	SPORT1 Channel A Status	Level	
75	SPORT1_A_DMA	SPORT1 Channel A DMA Transfer Complete	Level	6
76	SPORT1_B_STAT	SPORT1 Channel B Status	Level	
77	SPORT1_B_DMA	SPORT1 Channel B DMA Transfer Complete	Level	7
78	UART1_STAT	UART1 Status	Level	
79	UART1_TXDMA	UART1 Transmit DMA Transfer Complete	Level	10
80	UART1_RXDMA	UART1 Receive DMA Transfer Complete	Level	11
81	SPI1_STAT	SPI1 Status	Level	
82	SPI1_TXDMA	SPI1 TX DMA Channel Transfer Complete	Level	8
83	SPI1_RXDMA	SPI1 RX DMA Channel Transfer Complete	Level	9

Table 7-2: ADSP-CM40x Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
84	SPI2_TXDMA	SPI2 TX Channel (non-DMA) Transfer Complete	Level	
85	SPI2_RXDMA	SPI2 RX Channel (non-DMA) Transfer Complete	Level	
86	EMAC0_STAT	EMAC0 Status		
87	UART2_STAT	UART2 Status	Level	
88	UART2_TXDMA	UART2 Transmit DMA Transfer Complete	Level	12
89	UART2_RXDMA	UART2 Receive DMA Operation Complete	Level	13
90	HAE0_STAT	HAE0 Status	Level	
91	HAE0_RXDMA_CH0	HAE0 RX DMA Channel 0 Transfer Complete	Level	14
92	HAE0_RXDMA_CH1	HAE0 RX DMA Channel 1 Transfer Complete	Level	15
93	HAE0_TXDMA	HAE0 TX DMA Channel 0 Transfer Complete	Level	16
94	ADCC0_TMR0_EVT	ADCC0 Timer 0 Event Complete	Level	
95	ADCC0_TMR1_EVT	ADCC0 Timer 1 Event Complete	Level	
96	DACC0_DAC0	DACC0 DAC Interrupt 0 Generated	Level	
97	DACC0_DAC1	DACC0 DAC Interrupt 1 Generated	Level	
98	SPI2_STAT	SPI2 Status	Level	
99	TWI0_DATA	TWI0 Data Interrupt	Level	
100	CRC0_DCNTEXP	CRC0 Datacount expiration	Level	
101	CRC0_ERR	CRC0 Error	Level	
102	MDMA0_SRC	Memory DMA Stream 0 Source / CRC0 In Channel Transfer Complete	Level	17
103	MDMA0_DST	Memory DMA Stream 0 Destination / CRC0 Out Channel Transfer Complete	Level	18
104	MDMA1_SRC	Memory DMA Stream 1 Source Channel Transfer Complete	Level	19
105	MDMA1_DST	Memory DMA Stream 1 Destination Channel Transfer Complete	Level	20
106	USB0_STAT	USB0 Status/FIFO Data Ready	Level	

Table 7-2: ADSP-CM40x Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
107	USB0_DATA	USB0 DMA Status/Transfer Complete	Level	
108	TRU0_INT0	TRU0 Interrupt 0 Generated	Edge	
109	TRU0_INT1	TRU0 Interrupt 1 Generated	Edge	
110	TRU0_INT2	TRU0 Interrupt 2 Generated	Edge	
111	TRU0_INT3	TRU0 Interrupt 3 Generated	Edge	
113	DPM0_EVT	DPM0 Event	Level	
114	SOFT0	Software-Driven Interrupt 0 Generated		
115	SOFT1	Software-Driven Interrupt 1 Generated		
116	SOFT2	Software-Driven Interrupt 2 Generated		
117	SOFT3	Software-Driven Interrupt 3 Generated		
118	SWU0_EVT	SWU0 Event		
119	SWU1_EVT	SWU1 Event		
120	SWU2_EVT	SWU2 Event		
121	SWU3_EVT	SWU3 Event		
122	SWU4_EVT	SWU4 Event		
123	CAN0_RX	CAN0 Receive Transfer Complete	Level	
124	CAN0_TX	CAN0 Transmit Transfer Complete	Level	
125	CAN0_STAT	CAN0 Status	Level	
126	CAN1_RX	CAN1 Receive Transfer Complete	Level	
127	CAN1_TX	CAN1 Transmit Transfer Complete	Level	
128	CAN1_STAT	CAN1 Status	Level	
129	CPTMR0_CPT0_ERR	CPTMR0 Status	Level	
130	CPTMR0_CPT1_ERR	CPTMR0 Status	Level	
131	CPTMR0_CPT2_ERR	CPTMR0 Status	Level	
132	CPTMR0_CPT0_MEAS	CPTMR0 Status	Level	
133	CPTMR0_CPT1_MEAS	CPTMR0 Status	Level	
134	CPTMR0_CPT2_MEAS	CPTMR0 Status	Level	
135	TAPC0_KEYFAIL	TAPC0 User Key Match Fail	Edge	

ADSP-CM40x SEC Trigger List

Table 7-3: ADSP-CM40x SEC Trigger List Masters

Trigger ID	Name	Description	Sensitivity
53	SEC0_FAULT	SEC0 Fault Indication Received	Edge

Table 7-4: ADSP-CM40x SEC Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

SEC Definitions

The event controller uses the following definitions.

System Events

System source indications including interrupts and faults

System Source

Point of origin of system event

SID (Identification, unique)

Source numeric identifier for each system source connected to the SEC

SSI

SEC source interface, system event source control, and status subblock of the SEC

SFI

SEC Fault Interface, fault management subblock of the SEC

NVIC

Nested Vectored Interrupt Controller

SEC Block Diagram

The ***SEC Block Diagram*** shows how event management architecture.

As shown in the figure, the SEC has two blocks for event management. NVIC deals with interrupt from various system sources while SFI monitors and manages any fault event triggered from various fault input sources. System interrupt sources are routed to the SFI through SEC source interface (SSI).

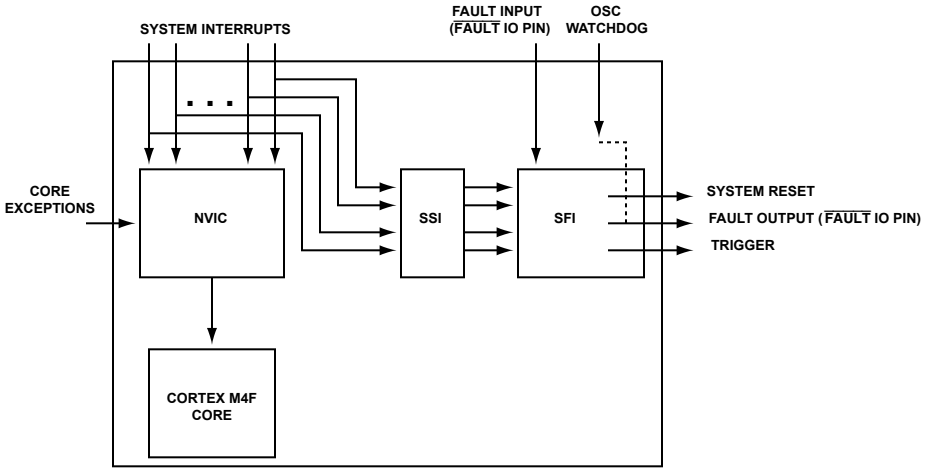

Figure 7-1: SEC Block Diagram

NOTE: NVIC is an independent unit inside the SEC closely tied to the Cortex core. Therefore its programming is the same as mentioned in the ARM Cortex standard documentation. In contrast, the SFI is mostly integrated to the SEC such that it uses all SEC-specific registers typically for fault management. For dealing with system interrupts, there is virtually no need for accessing any of the SEC registers.

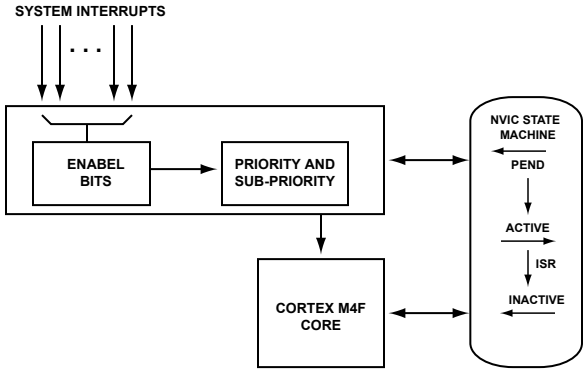

Figure 7-2: NVIC Block Diagram

SFI Block Diagram

The SFI manages fault events and associated actions. The fault management support provided in the SEC helps satisfy the safety requirements of various applications. The SSI provides the highest priority pending source that is enabled as a fault. The SFI captures this value and enables a countdown and, once the countdown expires, takes the prescribed action.

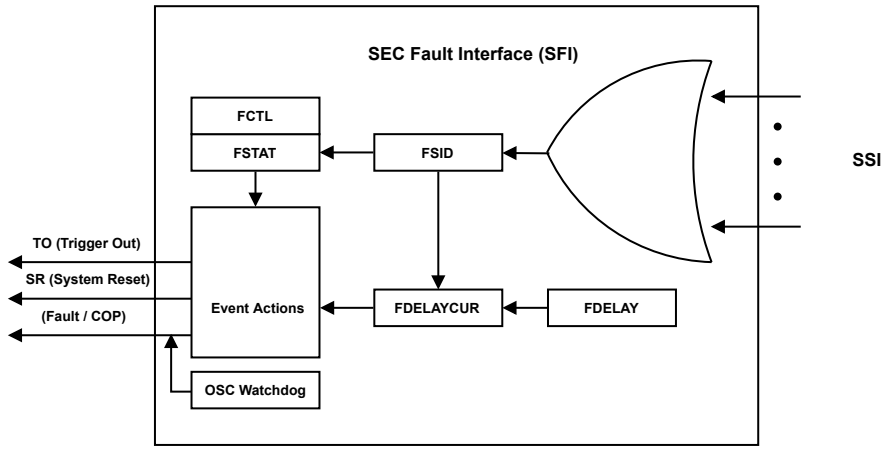

Figure 7-3: SFI Block Diagram

SEC Architectural Concepts

The following sections describe SEC architectural features supporting:

- Interrupt acknowledge
- Priority levels
- Grouping
- Flow
- Error management

System Interrupt Acknowledge

A system interrupt acknowledge occurs when the core provides an indication that it has acquired the SID of the interrupt last issued by the SEC. The SEC core interface option allows generation by:

- The assertion of an input acknowledge signal (the connected core generates the signal).

Nested Vectored Interrupt Controller (NVIC)

The Cortex-M4 processor closely integrates a configurable Nested Vectored Interrupt Controller (NVIC). The NVIC includes a non-maskable interrupt (NMI) that can provide up to 16 interrupt pre-emptive priority levels. The tight integration of the processor core and NVIC provides fast execution of interrupt service routines (ISRs), dramatically reducing the interrupt latency. This performance is achieved through stacking hardware registers and suspending load-multiple and store-multiple operations. Additionally, tail-chain optimization significantly reduces the overhead when switching from one ISR to another. The following are descriptions of the interrupt types used in the system.

Cortex system exceptions

The exceptions triggered from within the ARM Cortex-M4 processor core have negative interrupt IDs and are identified in the interrupt list with an M4_SCS0_ prefix. The Reset, Hard Fault, and NMI exceptions have fixed negative priority values, and these have higher priority than any other exception.

External interrupts

There are a total of 129 external interrupts in the processor. Almost all of these interrupts are triggered from several peripheral interrupt sources. A maximum of 16 levels of pre-emptive priority are possible. Interrupts can be enabled or disabled individually through the interrupt set or clear register and the priority level (priority + subpriority) can be defined by programming the interrupt priority registers.

Reset interrupt

When reset signal is de-asserted, execution restarts from the address provided by the reset entry in the vector table. Execution restarts as privileged execution in thread mode. Refer to the Reset Control Unit chapter for more details on reset implementation and usage in the processor. Refer to the Boot ROM chapter for information about how the ROM handles the reset event, before jumping to application.

Non-Maskable Interrupt (NMI)

NMI can be asserted with the following sources:

- Through the non-maskable interrupt pin, $\overline{\texttt{SYS\_NMI}}$.
- Through trigger outputs from TRU unit.

Vector Table

The vector table contains the reset value of the stack pointer, and the start addresses, also called exception vectors, for all exception handlers. On system reset, the vector table is fixed at address 0x00000000, which is inside the Boot ROM, for the processor. The address offset is aligned to the vector table size, extended to the next larger power of 2.

See the ADSP-CM40x Interrupt List for the ADSP-CM40x processor exception vectors.

Priority Grouping

ADSP-CM40x NVIC is configured with 4 bits of priority. The lower bits of the register are always 0; ***PRI_N[7:4]*** sets the priority, and ***PRI_N[3:0]*** is 4'b0000.

Table 7-5: PRIGROUP Implementation

PRIGROUP	Binary Point	Group Priority		Subpriority	
		Bits	Levels	Bits	Levels
011	4.0	4	16	0	0
100	3.1	3	8	1	2
101	2.2	2	4	2	4
110	1.3	1	2	3	8
111	0.4	0	0	4	16

NVIC Registers with ADSP-CM40x Specifications

The *Cortex-M4 Generic User Guide* documents a total of:

- Eight set or clear enable registers
- Eight set or clear pending registers
- Eight active bit registers
- 60 priority registers

For the ADSP-CM40x processor, only those bits and registers related to the 129 external interrupts of the processor are valid. Interrupt number assignment starts from 0 for external system interrupts, which the NVIC manages.

System Fault Interface (SFI) and NVIC

The system fault interface (SFI) operates independently from the NVIC. All system sources that can generate interrupts can also be routed to generate faults through the SEC source control registers. A fault can also be generated from the SYS_FAULT pin or from the oscillator watchdog. For more information on a fault generated from the oscillator watchdog, see the clock generation unit (CGU) chapter.

NOTE: For the ADSP-CM40x, the SYS_FAULT input pin is same as the output pin. The SYS_FAULT pin always operates in an open drain configuration and requires an external pull-up for correct operation.

Fault Input Options

A number of sources can generate a fault:

- Through the SYS_FAULT pin (external source)
- The oscillator watchdog
- System interrupt sources

Externally generated through SYS_FAULT Pin

When the SFI detects a falling edge at the SYS_FAULT pin, the SFI can sense this condition as an external fault and take necessary action. The SFI must be configured with the SEC_FCTL.FIEN bit to sample the pin for a fault. One usage case of this feature is to control the processor from another system or to permit another processor to communicate a fault occurrence.

From Oscillator Watchdog

If there is an oscillator watchdog fault input, the SEC could not be functional, because the SEC needs a clock for its operation. The direct path to the SYS_FAULT pin is provided in that case. The oscillator watchdog and its fault generation are explained in the clock generation unit (CGU) chapter.

Faults from System Interrupt Sources

System interrupts from various peripherals and other sources can be routed to the SFI through the SSI. This routing is enabled by setting the source signal enable (SEC_SCTL[n].SEN) and fault enable (SEC_SCTL[n].FEN) bits.

When the SSI detects the interrupt assertion and if the fault is enabled for that interrupt, the fault is passed to the SFI for further fault management actions.

Because the NVIC is an independent block outside of the SSI and SFI, the NVIC would not know how SFI manages the fault. Managing interrupts is dedicated to NVIC and managing faults is dedicated to SFI. So, configure the NVIC appropriately to service the interrupt using an interrupt service routine (ISR).

After the interrupt line is asserted, the signal is sent to both the NVIC and the SFI. Because the source for interrupt and fault generation is the same, a program can manage the fault inside the NVIC handler. Fault events are triggered when the delay registers count down to zero. In some cases, a program can perform a disable and reenable of the `SEC_SCTL[n].FEN` bit inside the handler to halt any fault event that is not desired when interrupt is running. For more information, see the examples in Programming Examples.

Managing Fault

A fault can be monitored and managed with a number of options:

- Trigger output
- System reset
- Indication on the $\overline{\texttt{SYS\_FAULT}}$ pin

Trigger Output

The fault interface does not have an interrupt line registered for dedicated management of fault actions. A program must assign a TRU slave (for example, NMI) to the SEC fault TRU master in order to manage the fault. This approach is not typically required when:

- a system interrupt is routed to the SFI through the SSI, and
- the NVIC interrupt service routine of that particular interrupt manages the fault event.

System Reset

It is possible to issue a system reset optionally. It is also possible to delay the assertion, such that the application can perform some critical housekeeping operations before the reset is generated.

Indication on the $\overline{\texttt{SYS\_FAULT}}$ *Pin*

SFI can drive the $\overline{\texttt{SYS\_FAULT}}$ pin to indicate fault to the external world. In a development system, this signal can be connected to an LED. In COP toggle mode, the SFI continuously sends out a series of pulses, and it stops when a fault is asserted.

Managing Fault Assertion

Pre-configure the SFI with input and output options, early during system initialization. The fault control register (`SEC_FCTL`) contains the bits for configuring these options. When COP toggle mode is selected, program the width value for the high and low phase of the toggled output on the COP pin . Note the following:

- The width value count is in SYSCLK cycles.

- The COP pin is an alternate function of the $\overline{\texttt{SYS\_FAULT}}$ pin.

For routing system interrupts to the SFI through the SSI, program the SEC_SCTL[n].SEN and SEC_SCTL[n].FEN bits.

The SEC fault delay register (SEC_FDLY) contains the number (SEC_FDLY.COUNT field) of SYSCLK periods to delay from fault pending to fault active, when actions are enabled. If enabled, the SEC fault system reset delay register (SEC_FSRDLY) contains the number (SEC_FSRDLY.COUNT field) of SYSCLK periods for the delay from a fault becoming active to system reset request assertion. Program these registers with values sufficient to manage critical tasks in the system, before the fault action can occur.

For SFI purposes, a fault can occur from a system interrupt source or from input on the $\overline{\texttt{SYS\_FAULT}}$ pin. The SEC_FSTAT.ACT bit indicates the status of a fault assertion. If this bit is set, it can be read from the fault source ID register (SEC_FSID). This register contains information about whether the last active fault was intesvnrnally (SSI) or externally triggered (fault pin). If it was triggered internally, the source ID field (SEC_FSID.SID) indicates the interrupt number of the system interrupt source.

SEC Error

The processor includes an SEC error (SEC_GSTAT.ERR) as a source input to the SEC to allow for handling the error as an interrupt or fault.

SEC Programming Model

Implementing a system interrupt service model using the SEC with NVIC requires, at a minimum:

- Proper configuration of a system interrupt source (for example a peripheral or DMA)
- A core interrupt, fault or event service model
- Proper configuration of the NVIC and the SEC

Programming Concepts

The following list provides the basic programming concepts necessary for configuring the NVIC and SEC.

- Configuring a System Interrupt with NVIC
- Configuring FMU as Fault Pin
- Managing Faults Inside a Triggered ISR
- Configuring and Managing Faults (that are also interrupts)

Programming Examples

This section provides example programming tasks that are typical for SEC usage.

Configuring a System Interrupt with NVIC

The NVIC supports configuring system interrupts.

1. Set the NVIC priority and subpriority levels for the interrupt.
2. Enable the system interrupt at peripheral source.
3. Enable the interrupt with the NVIC.
4. Inside the interrupt service routine (ISR), the NVIC pushes and pops the C program ABI registers.
5. The program must clear the source for the interrupt inside the ISR.

Configuring FMU as Fault Pin

The NVIC supports configuring the FMU as a fault pin.

1. Enable the `SEC_GCTL.RESET` bit.
2. Enable the `SEC_GCTL.EN` bit.
3. Program the `SEC_FCTL.FIEN` bit.
4. Program the `SEC_FCTL` register with the `SEC_FDLY` delay time from fault pending to fault active.
5. Program the `SEC_FCTL` register with `SEC_FCTL.SREN` or `SEC_FCTL.TOEN`, depending on requirements. If using trigger mode, route the fault trigger to an interrupt handler such as an NMI interrupt service routine (ISR).
6. Enable the fault unit by setting the `SEC_FCTL.EN` bit.
7. If using trigger mode, the fault is dealt with inside the interrupt handler of the ISR to which the fault was routed.

Managing Faults Inside a Triggered ISR

The SEC supports fault management within an interrupt service routine (ISR).

1. Check whether the `SEC_FSTAT.ACT` bit is set.
2. Check whether the `SEC_FSID.FEXT` bit is set.
    a. If set, clear the external fault pin source by writing the `SEC_FEND.FEXT` bit.
    b. If not set, clear the system interrupt source by writing the `SEC_FSID.SID` bit.

Configuring and Managing Faults (that are also interrupts)

The SEC permits simultaneously registering an interrupt with NVIC and configuring the interrupt as fault (configuring and managing a fault that is also an interrupt).

1. Enable the SEC_GCTL.RESET bit.
2. Enable the SEC_GCTL.EN bit.
3. Program the SEC_SCTL[n].SEN and SEC_SCTL[n].FEN bits for the SSI.
4. Program the SEC_FCTL register with options: fault delay, COP toggle mode, system reset, and others.
5. Enable the fault unit by setting the SEC_FCTL.EN bit.

   ADDITIONAL INFORMATION: The NVIC handles interrupts. Inside the interrupt service routine (ISR), the software must first clear the system interrupt source. If software does not handle the fault soon enough, the fault events are generated. So, prioritize interrupt handling first (finish it first).
6. To halt fault event, perform the following:
   a. Clear the SEC_SCTL[n].FEN and SEC_SCTL[n].SEN bits.
   b. Write the SEC_SSTAT[n].PND bits.
   c. Handle the fault as described in triggered ISR case.
   d. Reenable the SEC_SCTL[n].FEN and SEC_SCTL[n].SEN bits.
   e. Return from the ISR.

SEC Programming Restrictions

Setting the SEC_FCTL.EN bit while the SEC_FSTAT.ACT bit is high can result in unpredictable behavior. To avoid this issue, set the SEC_FCTL.EN bit while the SEC_FSTAT.ACT bit is low. The SEC_FSTAT.ACT bit is only set when the SEC_FCTL.EN bit is high. Therefore, the problem can only occur if the SEC_FCTL.EN bit transitions from 1 to 0 and then to 1 again.

Writing to SEC_FEND to end a fault with both the SEC_FCTL.FOEN bit and the SEC_FCTL.FIEN bit set can result in erroneous external fault detection. If this operation (ending a fault) and configuration (fault input and fault output enabled) are required by the application, clear the SEC_FCTL.FOEN bit prior to writing to SEC_FEND. The recommended sequence for ending a fault with the SEC_FCTL.FIEN or SEC_FCTL.FOEN==1 is as follows:

1. Clear SEC_FCTL.FOEN.
2. Write SEC_FEND.
3. Set SEC_FCTL.FOEN.

ADSP-CM40x SEC Register Descriptions

System Event Controller (SEC) contains the following registers.

Table 7-6: ADSP-CM40x SEC Register List

Name	Description
SEC_FCOPP	Fault COP Period Register
SEC_FCOPP_CUR	Fault COP Period Current Register
SEC_FCTL	Fault Control Register
SEC_FDLY	Fault Delay Register
SEC_FDLY_CUR	Fault Delay Current Register
SEC_FEND	Fault End Register
SEC_FSID	Fault Source ID Register
SEC_FSRDLY	Fault System Reset Delay Register
SEC_FSRDLY_CUR	Fault System Reset Delay Current Register
SEC_FSTAT	Fault Status Register
SEC_GCTL	Global Control Register
SEC_GSTAT	Global Status Register
SEC_RAISE	Global Raise Register
SEC_SCTL[n]	Source Control Register n
SEC_SSTAT[n]	Source Status Register n

Fault COP Period Register

The SEC fault COP period register (SEC_FCOPP) contains the width value (count in (SEC) clock cycles) for the high and low phase of the computer operating properly (COP) toggled output on the COP pin. Note that the actual high/low phase is value is the SEC_FCOPP.COUNT programmed value plus 1.

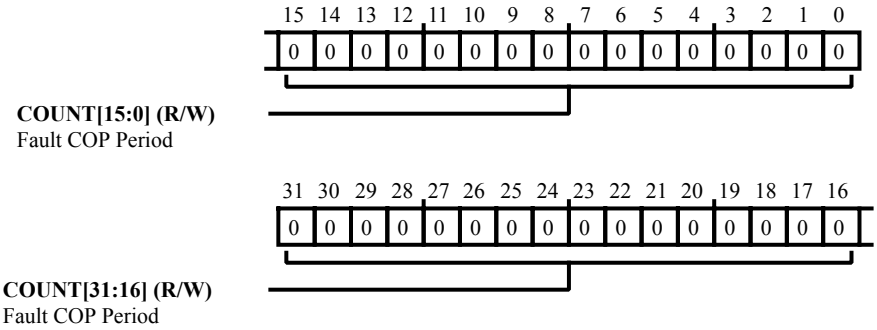

Figure 7-4: SEC_FCOPP Register Diagram

Table 7-7: SEC_FCOPP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	COUNT	Fault COP Period. The SEC_FCOPP.COUNT bit field is the width value for the high and low phase of the computer operating properly (COP) toggled output on the COP pin.

Fault COP Period Current Register

The SEC fault COP period current register (SEC_FCOPP_CUR) contains the active count (in (SEC) clock periods) for the current phase (high or low) of the computer operating properly (COP) toggled output on the COP pin. The SEC loads the SEC_FCOPP_CUR register from the SEC_FCOPP register when the SEC_FCOPP_CUR.COUNT field is cleared and the SEC is in COP mode (SEC_FCTL.CMS bit =1). The SEC decrements the SEC_FCOPP_CUR count each (SEC) clock cycle while SEC_FCTL.CMS is set and the SEC_FSTAT.ACT bit is not set.

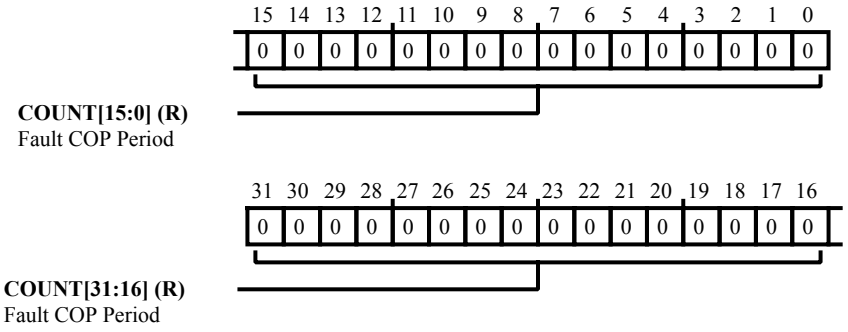

Figure 7-5: SEC_FCOPP_CUR Register Diagram

Table 7-8: SEC_FCOPP_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	COUNT	Fault COP Period. The SEC_FCOPP_CUR.COUNT bit field is the active count for the current phase (high or low) of the computer operating properly (COP) toggled output on the COP pin.

Fault Control Register

The SEC fault control register (SEC_FCTL) contains fault control bits for all SEC channels. This register controls the operation of the System Fault Management Interface (SFI).

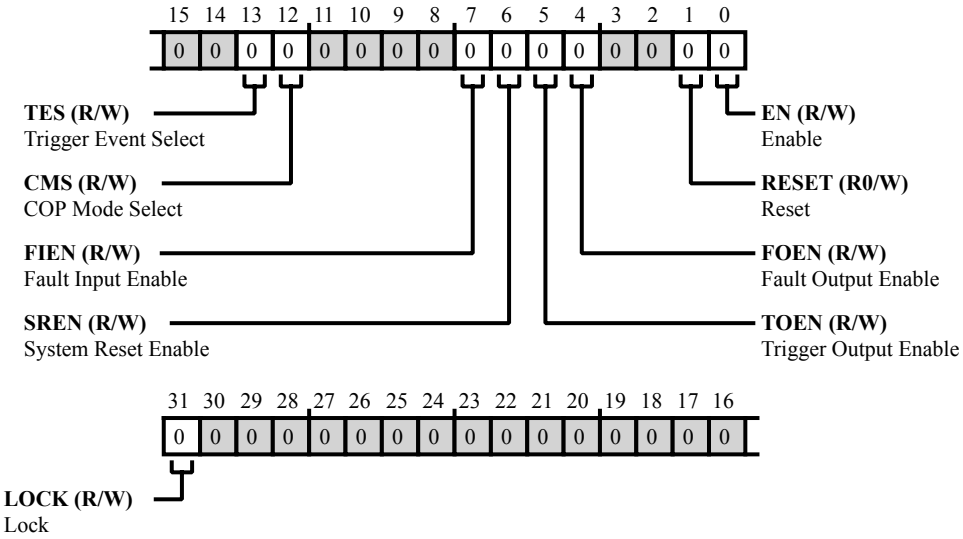

Figure 7-6: SEC_FCTL Register Diagram

Table 7-9: SEC_FCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (SPU_CTL.GLCK bit =1) and the SEC_FCTL.LOCK bit is enabled, the SEC_FCTL register is read only.	
		0	UnLock
		1	Lock
13 (R/W)	TES	Trigger Event Select. The SEC_FCTL.TES bit selects the event that directs the SEC to assert trigger output. In fault pending mode, the SEC asserts trigger output when a fault is pending. In fault active mode, the SEC asserts trigger output when a fault is active.	
		0	Fault Active Mode
		1	Fault Pending Mode

Table 7-9: SEC_FCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	CMS	COP Mode Select. The SEC_FCTL.CMS selects the SEC mode for handling fault input. In COP mode, the SEC toggles the COP pin to indicate that no fault is active and ceases toggling the pin to indicate that a fault is active. In fault mode, the SEC deasserts the fault pin (=0) and fault_b pin (=1) when no fault is active and asserts the fault pin (=1) and fault_b pin (=0) when a fault is active. Not all processors feature both the fault and fault_b pins. Refer to the product data sheet for details.	
		0	Fault Mode
		1	COP Mode
7 (R/W)	FIEN	Fault Input Enable. The SEC_FCTL.FIEN bit enables the SEC the to sample fault input. If SEC_FCTL.FIEN is set (=1), a fault indication from an external device sets the SEC_FSTAT.ACT bit and SEC_FSID.FEXT bit.	
		0	Disable
		1	Enable
6 (R/W)	SREN	System Reset Enable. The SEC_FCTL.SREN bit enables the SEC to issue a system reset request when a fault becomes active.	
		0	Disable
		1	Enable
5 (R/W)	TOEN	Trigger Output Enable. The SEC_FCTL.TOEN bit enables the SEC to produce trigger output when a fault becomes active.	
		0	Disable
		1	Enable
4 (R/W)	FOEN	Fault Output Enable. The SEC_FCTL.FOEN bit enables the SEC to indicate fault status, according to the SEC_FCTL.CMS bit configuration.	
		0	Disable
		1	Enable
1 (R0/W)	RESET	Reset. Setting the SEC_FCTL.RESET bit resets ALL SEC registers to their default values.	
		0	No Action
		1	Reset

Table 7-9: SEC_FCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Enable. The SEC_FCTL.EN bit controls the operational state of the SEC. Clearing the SEC_FCTL.EN bit halts the execution of the SEC without resetting status registers. Setting the SEC_FCTL.EN bit enables the SEC to begin or resume operation with the current configuration and status.	
		0	Disable
		1	Enable

Fault Delay Register

The SEC fault delay register (SEC_FDLY) contains the number (SEC_FDLY.COUNT field) of (SEC) clock periods to delay from fault pending to fault active, when actions are enabled.

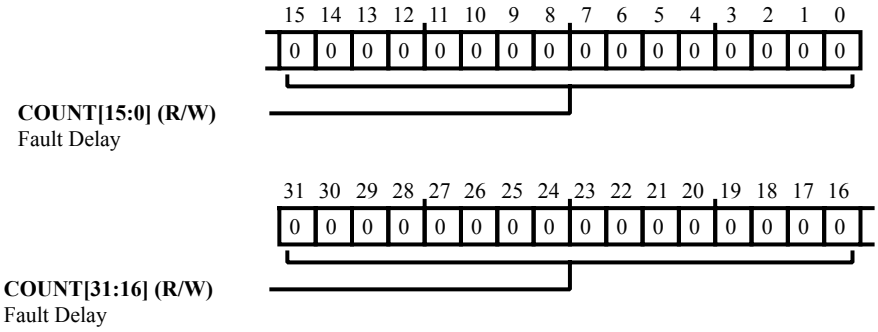

Figure 7-7: SEC_FDLY Register Diagram

Table 7-10: SEC_FDLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	COUNT	Fault Delay. The SEC_FDLY.COUNT bit field is the number of (SEC) clock periods to delay from fault pending to fault active, when actions are enabled.

Fault Delay Current Register

The SEC fault delay current register (SEC_FDLY_CUR) contains the active count (SEC_FDLY_CUR.COUNT field) in (SEC) clock periods for the delay from fault pending to fault active, when actions are enabled. The count is loaded from the SEC_FDLY register when a fault becomes pending (SEC_FSTAT.PND bit is set). The SEC decrements the value in SEC_FDLY_CUR each (SEC) clock cycle while the SEC_FSTAT.PND bit is set.

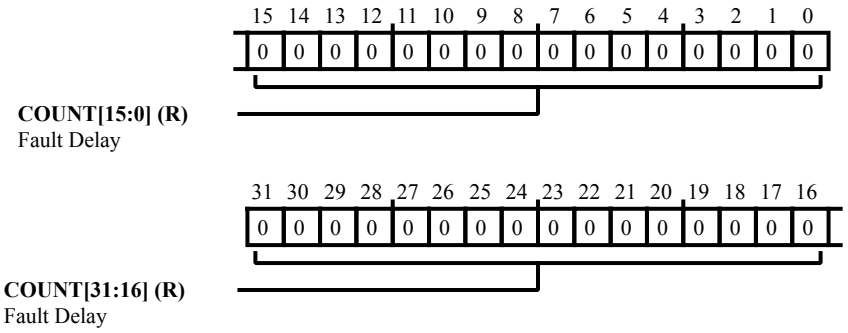

Figure 7-8: SEC_FDLY_CUR Register Diagram

Table 7-11: SEC_FDLY_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	COUNT	Fault Delay. The SEC_FDLY_CUR.COUNT bit field is the active count in (SEC) clock periods for the delay from fault pending to fault active, when actions are enabled.

Fault End Register

The SEC fault end register (SEC_FEND) contains fault source ID and internal/external fields. This register receives fault end indication from a core.

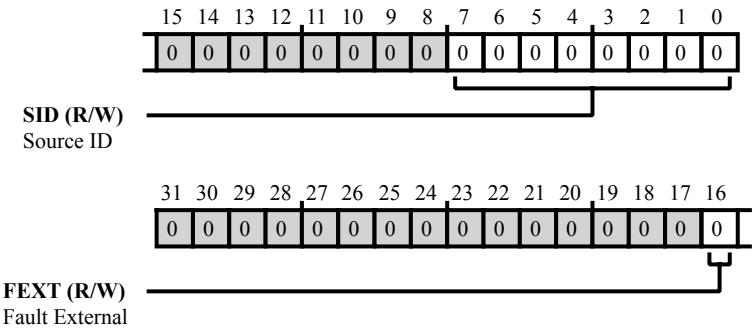

Figure 7-9: SEC_FEND Register Diagram

Table 7-12: SEC_FEND Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	FEXT	Fault External. Setting the SEC_FEND.FEXT bit, when the SEC_FEND.SID field is cleared, clears an active fault from an external source.	
		0	Fault Internal
		1	Fault External
7:0 (R/W)	SID	Source ID. The SEC_FEND.SID identifies a fault to be ended as indicated to the SEC by the core. The core loads the SEC_FEND.SID field value. If the SEC_FEND.SID value matches the SEC_FSID.SID value, the SEC_FSTAT.PND bit and SEC_FSTAT.ACT bit are cleared.	

Fault Source ID Register

The SEC fault source ID register (SEC_FSID) contains a fault source ID and internal/external fields.

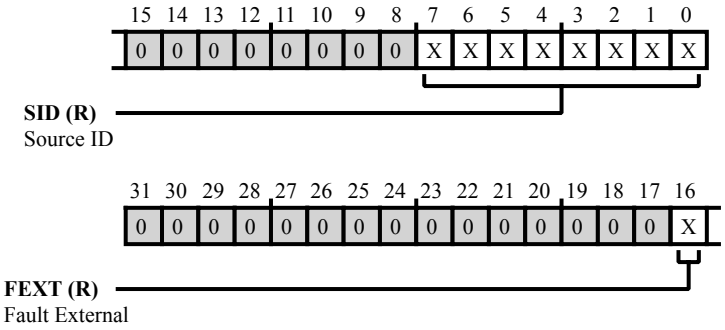

Figure 7-10: SEC_FSID Register Diagram

Table 7-13: SEC_FSID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/NW)	FEXT	Fault External. The SEC_FSID.FEXT bit indicates that the last active fault was asserted by an external device. The SEC sets the SEC_FSID.FEXT bit when the SEC_FSTAT register's SEC_FSTAT.ACT bit is set by the fault input pins. The SEC_FSID.FEXT bit is cleared when the SEC_FSTAT.ACT bit is set by an internal fault or when the external fault is ended. When the SEC_FSID.FEXT bit is set, the SEC_FSID.SID is cleared.	
		0	Fault Internal
		1	Fault External
7:0 (R/NW)	SID	Source ID. The SEC_FSID.SID identifies the fault assertion detected by the SEC fault interface. The SEC loads the SEC_FSID.SID field value when a system fault indication is asserted. The SEC fault interface does not change the SEC_FSID.SID value until the fault is no longer pending or active, as indicated by the SEC_FSTAT.PND bit and SEC_FSTAT.ACT bit being cleared in the SEC_FSTAT register.	

Fault System Reset Delay Register

The SEC fault system reset delay register (SEC_FSRDLY) contains the number (SEC_FSRDLY.COUNT field) of (SEC) clock periods for the delay from a fault becoming active to system reset request assertion, if enabled.

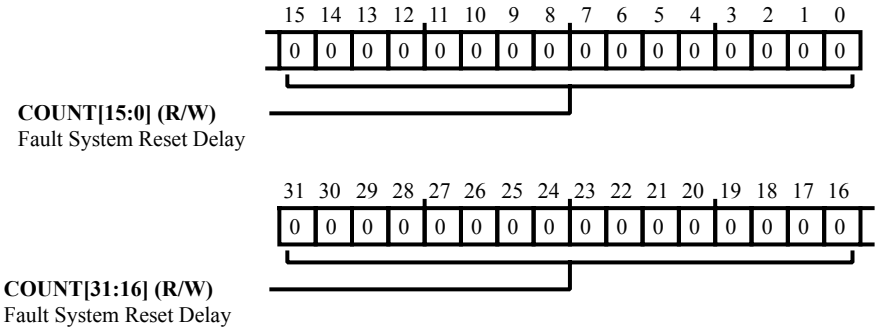

Figure 7-11: SEC_FSRDLY Register Diagram

Table 7-14: SEC_FSRDLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	COUNT	Fault System Reset Delay. The SEC_FSRDLY.COUNT bit field is the number of (SEC) clock periods for the delay from a fault becoming active to system reset request assertion

Fault System Reset Delay Current Register

The SEC fault system reset delay current register (SEC_FSRDLY_CUR) contains the active count (SEC_FSRDLY_CUR.COUNT field) in (SEC) clock periods for the delay from fault active to system reset assertion, if enabled. The count is loaded from the SEC_FSRDLY register when a fault becomes active (SEC_FSTAT.ACT bit is set). The SEC decrements the value in SEC_FSRDLY_CUR each (SEC) clock cycle while the SEC_FSTAT.ACT bit is set.

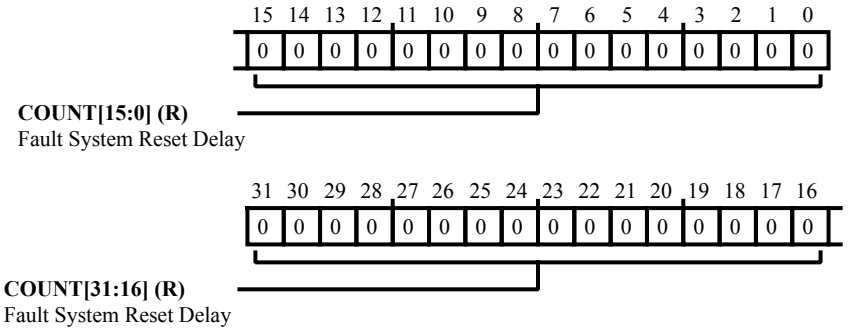

Figure 7-12: SEC_FSRDLY_CUR Register Diagram

Table 7-15: SEC_FSRDLY_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	COUNT	Fault System Reset Delay. The SEC_FSRDLY_CUR.COUNT bit field is the active count in (SEC) clock periods for the delay from fault active to system reset assertion.

Fault Status Register

The SEC fault status register (SEC_FSTAT) indicates the operational status of the SFI.

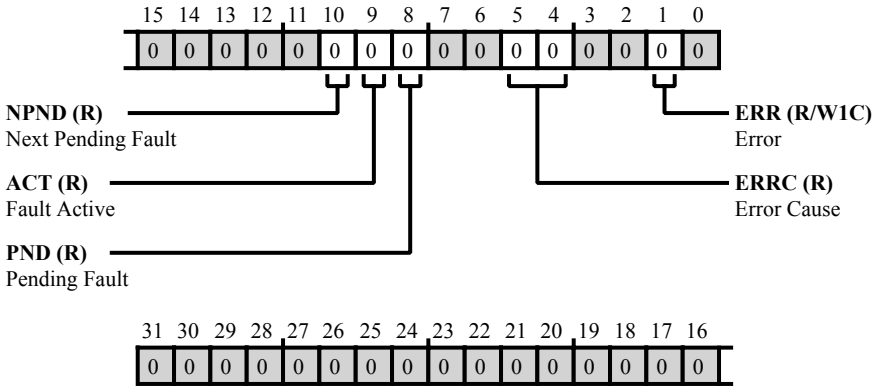

Figure 7-13: SEC_FSTAT Register Diagram

Table 7-16: SEC_FSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/NW)	NPND	Next Pending Fault. The SEC_FSTAT.NPND bit indicates that one or more sources have signaled fault assertion, but the input has not yet triggered the fault pending detection in the SEC fault interface. The SEC sets the SEC_FSTAT.NPND bit when the fault interface detects assertion of any enabled fault source input, while either SEC_FSTAT.PND or SEC_FSTAT.ACT bits are set. The SEC clears the SEC_FSTAT.NPND bit when there are no fault sources waiting.	
		0	Not Pending
		1	Pending
9 (R/NW)	ACT	Fault Active. The SEC_FSTAT.ACT bit indicates that the SEC has received a fault source input, the current fault delay count (in the SEC_FDLY_CUR register) has expired, and the fault actions are enabled. The SEC also sets the SEC_FSTAT.ACT bit on fault input detection if the SEC_FCTL.FIEN bit is set. The SEC_FSTAT.ACT bit is cleared by writing the ID value of the asserted fault from SEC_FSID register to the SEC_FEND register.	
		0	No Fault
		1	Active Fault

Table 7-16: SEC_FSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	PND	Pending Fault. The SEC_FSTAT.PND bit indicates a fault source has signalled a fault assertion to the SEC, but the SEC has not yet triggered the event actions due to the delay selected with the SEC_FDLY register. The SEC fault interface sets the SEC_FSTAT.PND bit when the SEC_FSID is updated on assertion of a fault source input. The SEC_FSTAT.PND bit is only set when the SEC_FSTAT.ACT bit is cleared. The SEC updates the SEC_FSID register with the SID value when the SEC_FSTAT.PND bit is set. The SEC_FSTAT.PND bit is cleared *either* by the SEC fault interface when the current delay count in the SEC_FDLY_CUR register expires *or* by writing the SEC_FSID.SID field value (which indicates the ID of the asserted fault) to the SEC_FEND register.	
		0	Not Pending
		1	Pending
5:4 (R/NW)	ERRC	Error Cause. When the SEC_FSTAT.ERR bit is asserted, the SEC updates SEC_FSTAT.ERRC field to convey the interrupt source error type. When the error type is source overflow, the status indicates that a source signal assertion occurred or an SEC raise operation was attempted while pending was already set. The source overflow is detected when the source is set for edge only.	
		0	Source Overflow Error
		1	Reserved
		2	End Error
		3	Reserved
1 (R/W1C)	ERR	Error. The SEC_FSTAT.ERR bit indicates an SEC fault interface error. When SEC_FSTAT.ERR is set, the SEC updates the SEC_FSTAT.ERRC field to indicate the corresponding error cause. When multiple errors occur, the SEC_FSTAT register captures the status for the first error and does not capture subsequent errors until the status is cleared.	
		0	No Error
		1	Error Occurred

Global Control Register

The SEC global control register (SEC_GCTL) provides register locking, reset, and enable for the SEC module.

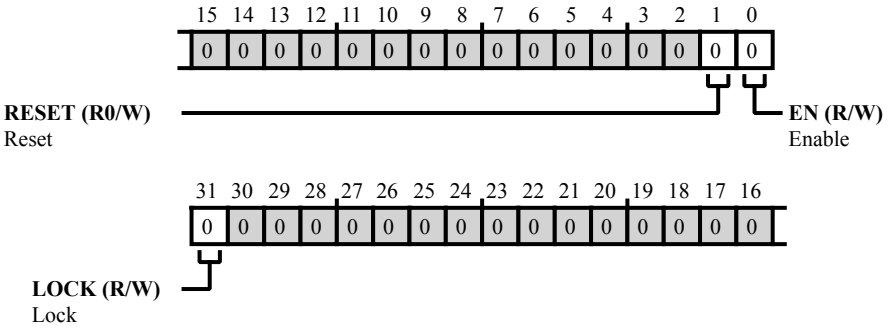

Figure 7-14: SEC_GCTL Register Diagram

Table 7-17: SEC_GCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (SPU_CTL.GLCK bit =1) and the SEC_GCTL.LOCK bit is enabled, the SEC_GCTL register is read only.	
		0	Unlock
		1	Lock
1 (R0/W)	RESET	Reset. The SEC_GCTL.RESET bit is write-1-action and triggers a soft reset to all SEC registers.	
		0	No Action
		1	Reset
0 (R/W)	EN	Enable. The SEC_GCTL.EN bit is read/write and must be set for the SEC to begin/resume SEC operation with the current configuration and status. Clearing the SEC_GCTL.EN bit halts the execution of the SFI. All SSIs remain active, along with all error detection, without resetting status registers.	
		0	Disable
		1	Enable

Global Status Register

The SEC global status register (SEC_GSTAT) contains global status bits for the SEC.

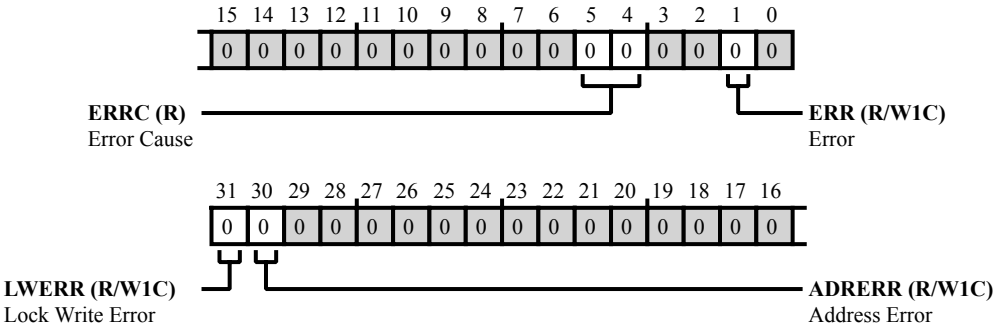

Figure 7-15: SEC_GSTAT Register Diagram

Table 7-18: SEC_GSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W1C)	LWERR	Lock Write Error. The SEC_GSTAT.LWERR bit indicates (when set) there was an attempted write to an SEC register while the SEC_GCTL.LOCK bit was set and while the global lock bit was enabled (SPU_CTL.GLCK bit =1). This status bit is sticky; write-1-to-clear it.	
		0	No Error
		1	Error Occurred
30 (R/W1C)	ADRERR	Address Error. The SEC_GSTAT.ADRERR bit indicates that the SEC generated and address error. This status bit is sticky; write-1-to-clear it.	
		0	No Error
		1	Error Occurred
5:4 (R/NW)	ERRC	Error Cause. When the SEC updates the SEC_GSTAT.ERR bit, the SEC updates the SEC_GSTAT.ERRC bits to indicate the error type. Note that for SSI errors, the error status indicates an error is active for any SSI input. This error is an OR of all the interrupt source errors.	
		0	SFI Error
		1	Reserved
		2	SSI Error
		3	Reserved

Table 7-18: SEC_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	ERR	Error. The SEC_GSTAT.ERR bit indicates an error has occurred in the SEC. When the SEC asserts this bit (=1), the SEC updates the SEC_GSTAT.ERRC field to indicate the corresponding error cause. Even if multiple errors occur, only the first error is captured on assertion of this bit. This status bit is sticky; write-1-to-clear it.	
		0	No Error
		1	Error Occurred

Global Raise Register

The SEC global raise register (SEC_RAISE) contains a source ID event set-to-pending field (SEC_RAISE.SID). When a source ID value is written to this field, the SEC raises the source's event status to pending.

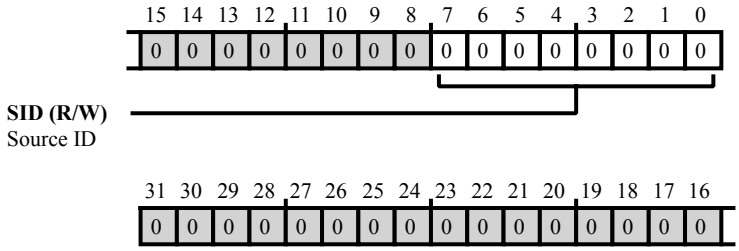

Figure 7-16: SEC_RAISE Register Diagram

Table 7-19: SEC_RAISE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	SID	Source ID. The SEC_RAISE.SID bit field is the source ID of event that is set to pending status.

Source Control Register n

The SEC source control register (SEC_SCTL[n]) contains control bits to configure the SEC event sources. This register controls the configuration of the corresponding SEC event source.

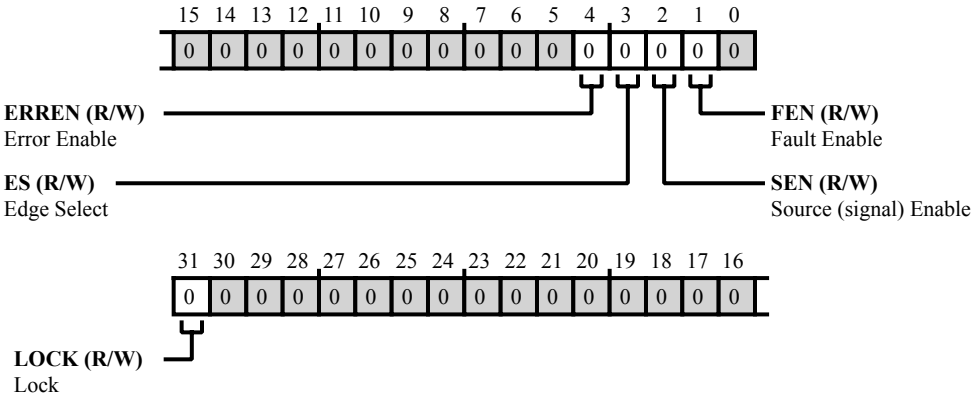

Figure 7-17: SEC_SCTL[n] Register Diagram

Table 7-20: SEC_SCTL[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (SPU_CTL.GLCK bit =1) and the SEC_SCTL[n].LOCK bit is enabled, the SEC_SCTL[n] register is read only.	
		0	Unlock
		1	Lock
4 (R/W)	ERREN	Error Enable. The SEC_SCTL[n].ERREN bit permits the SEC_SSTAT[n].ERR status bit to be set on error detection. If SEC_SCTL[n].ERREN is cleared, no errors are detected.	
		0	Disable
		1	Enable
3 (R/W)	ES	Edge Select. The SEC_SCTL[n].ES bit selects the operational and sensitivity mode of the SEC source interface input.	
		0	Level Sensitive
		1	Edge Sensitive

Table 7-20: SEC_SCTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	SEN	Source (signal) Enable. The `SEC_SCTL[n].SEN` bit controls whether the system event source input signal may affect the Pending status of the source. Clearing `SEC_SCTL[n].SEN` disables the source input signal from affecting Pending. Setting `SEC_SCTL[n].SEN` enables the source input signal to affect Pending status.	
		0	Disable
		1	Enable
1 (R/W)	FEN	Fault Enable. The `SEC_SCTL[n].FEN` bit controls whether the SEC may forward an interrupt request to the SEC fault interface as a fault source. This bit does not affect the ability of an interrupt source to set an interrupt as pending. The `SEC_SCTL[n].FEN` bit only affects whether the pending request may be forwarded to the SEC fault interface.	
		0	Disable
		1	Enable

Source Status Register n

The SEC event source status register (SEC_SSTAT[n]) contains bits indicating the status of the corresponding event source n. An event source may be: pending, active, active and pending, or neither pending nor active.

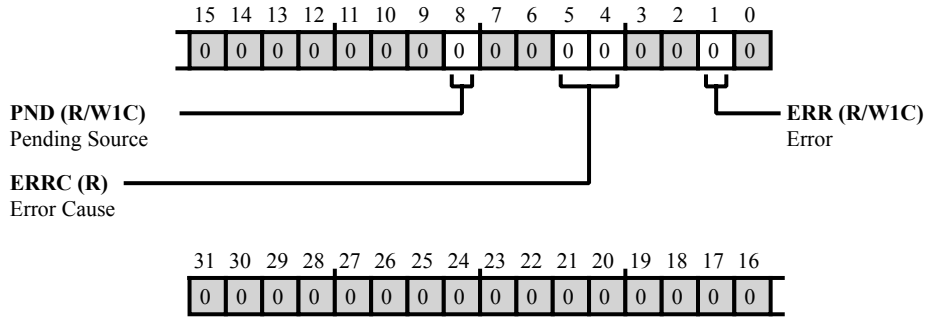

Figure 7-18: SEC_SSTAT[n] Register Diagram

Table 7-21: SEC_SSTAT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W1C)	PND	Pending Source. The SEC_SSTAT[n].PND bit indicates the source has signaled an event request, but the event request has not been (or is not currently being) serviced. A SEC_SSTAT[n].PND bit is set by the SEC on detection of an assertion of the corresponding system source input. A SEC_SSTAT[n].PND bit is cleared by a W1C operation.	
		0	Not Pending
		1	Pending
5:4 (R/NW)	ERRC	Error Cause. When the SEC_SSTAT[n].ERR bit is asserted, the SEC updates SEC_SSTAT[n].ERRC field to convey the interrupt source error type. When the error type is source overflow, the status indicates that a source signal assertion occurred or an SEC raise operation was attempted while pending was already set. The source overflow is detected when the source is set for edge only. .	
		0	Source Overflow Error
		1	Reserved
		2	End Error
		3	Reserved

Table 7-21: SEC_SSTAT[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	ERR	Error. The SEC_SSTAT[n].ERR bit indicates an error for a specific system interrupt source. When the SEC_SSTAT[n].ERR bit is set, the SEC updates the SEC_SSTAT[n].ERRC field to the value of the corresponding error cause. Even if multiple errors occur, only the first error is captured on assertion of the SEC_SSTAT[n].ERR bit.	
		0	No Error
		1	Error Occurred

8 Trigger Routing Unit (TRU)

The TRU provides system-level sequence control without core intervention. The TRU maps trigger masters (generators of triggers) to trigger slaves (receivers of triggers). Slave endpoints can be configured to respond to triggers in various ways. Common applications enabled by the TRU include:

- Automatically triggering the start of a DMA sequence after a sequence from another DMA channel completes
- Software triggering
- Synchronization of concurrent activities

TRU Features

The TRU supports the following features.

- Trigger routing of any trigger master to any trigger slave
- Software generation of any trigger master ID
- Configuration protection through register-level lock bits and global lock indication

TRU Functional Description

The following sections provide a description of the TRU.

ADSP-CM40x TRU Register List

The Trigger Routing Unit (TRU) provides simple sequence control of distributed modules without the penalties associated with core intervention (for example, interrupt overhead). The TRU receives trigger inputs from all master trigger inputs (MTI) and the TRU master trigger register (TRU_MTR). Based on these inputs, the TRU logic generates trigger outputs that initiate slave operations in the processor core and peripherals. A set of registers governs TRU operations. For more information on TRU functionality, see the TRU register descriptions.

Table 8-1: ADSP-CM40x TRU Register List

Name	Description
TRU_ERRADDR	Error Address Register

Table 8-1: ADSP-CM40x TRU Register List (Continued)

Name	Description
TRU_GCTL	Global Control Register
TRU_MTR	Master Trigger Register
TRU_SSR[n]	Slave Select Register
TRU_STAT	Status Information Register

ADSP-CM40x TRU Interrupt List

Table 8-2: ADSP-CM40x TRU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
108	TRU0_INT0	TRU0 Interrupt 0 Generated	Edge	
109	TRU0_INT1	TRU0 Interrupt 1 Generated	Edge	
110	TRU0_INT2	TRU0 Interrupt 2 Generated	Edge	
111	TRU0_INT3	TRU0 Interrupt 3 Generated	Edge	

ADSP-CM40x Trigger List

Table 8-3: ADSP-CM40x Trigger List Masters

Trigger ID	Name	Description	Sensitivity
0		Reserved	
1	CGU0_EVT	CGU0 Event	Edge
2	TIMER0_TMR0_MST	TIMER0 Timer 0 Expiration or Event	Edge
3	TIMER0_TMR1_MST	TIMER0 Timer 1 Expiration or Event	Edge
4	TIMER0_TMR2_MST	TIMER0 Timer 2 Expiration or Event	Edge
5	TIMER0_TMR3_MST	TIMER0 Timer 3 Expiration or Event	Edge
6	TIMER0_TMR4_MST	TIMER0 Timer 4 Expiration or Event	Edge
7	TIMER0_TMR5_MST	TIMER0 Timer 5 Expiration or Event	Edge
8	TIMER0_TMR6_MST	TIMER0 Timer 6 Expiration or Event	Edge
9	TIMER0_TMR7_MST	TIMER0 Timer 7 Expiration or Event	Edge
10	PINT0_BLOCK	PINT0 Block Interrupt Generated	Level
11	PINT1_BLOCK	PINT1 Block Interrupt Generated	Level
12	PINT2_BLOCK	PINT2 Block Interrupt Generated	Level
13	PINT3_BLOCK	PINT3 Block Interrupt Generated	Level

Table 8-3: ADSP-CM40x Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
14	PINT4_BLOCK	PINT4 Block Interrupt Generated	Level
15	CNT0_STAT	CNT0 Counter Status	Level
16	CNT1_STAT	CNT1 Counter Status	Level
17	CNT2_STAT	CNT2 Counter Status	Level
18	CNT3_STAT	CNT3 Counter Status	Level
19	PWM0_SYNC	PWM0 PWMTMR Group Trigger	Edge
20	PWM1_SYNC	PWM1 PWMTMR Group Trigger	Edge
21	PWM2_SYNC	PWM2 PWMTMR Group Trigger	Edge
22	ADCC0_TMR0_EVT	ADCC0 Timer 0 Event Complete	Level
23	ADCC0_TMR1_EVT	ADCC0 Timer 1 Event Complete	Level
24	SPORT0_A_DMA	SPORT0 Channel A DMA Transfer Complete	Level
25	SPORT0_B_DMA	SPORT0 Channel B DMA Transfer Complete	Level
26	SPORT1_A_DMA	SPORT1 Channel A DMA Transfer Complete	Level
27	SPORT1_B_DMA	SPORT1 Channel B DMA Transfer Complete	Level
28	SPI0_TXDMA	SPI0 TX DMA Channel Transfer Complete	Level
29	SPI0_RXDMA	SPI0 RX DMA Channel Transfer Complete	Level
30	SPI1_TXDMA	SPI1 TX DMA Channel Transfer Complete	Level
31	SPI1_RXDMA	SPI1 RX DMA Channel Transfer Complete	Level
32	EMAC0_STAT	EMAC0 Status	
33	HAE0_RXDMA_CH0	HAE0 RX DMA Channel 0 Transfer Complete	Level
34	HAE0_RXDMA_CH1	HAE0 RX DMA Channel 1 Transfer Complete	Level
35	HAE0_TXDMA	HAE0 TX DMA Channel 0 Transfer Complete	Level
36	SINC0_P0_OVLD	SINC0 Pair 0 Overload Indicator	Edge
37	SINC0_P1_OVLD	SINC0 Pair 1 Overload Indicator	Edge
38	SINC0_P2_OVLD	SINC0 Pair 2 Overload Indicator	Edge
39	SINC0_P3_OVLD	SINC0 Pair 3 Overload Indicator	Edge
40	SINC0_DATA0	SINC0 Data Move 0 Complete	Edge
41	SINC0_DATA1	SINC0 Data Move 1 Complete	Edge
42	UART0_TXDMA	UART0 Transmit DMA Transfer Complete	Level
43	UART0_RXDMA	UART0 Receive DMA Transfer Complete	Level
44	UART1_TXDMA	UART1 Transmit DMA Transfer Complete	Level

Table 8-3: ADSP-CM40x Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
45	UART1_RXDMA	UART1 Receive DMA Transfer Complete	Level
46	UART2_TXDMA	UART2 Transmit DMA Transfer Complete	Level
47	UART2_RXDMA	UART2 Receive DMA Transfer Complete	Level
48	MDMA0_SRC	Memory DMA Stream 0 Source/CRC0 Input Channel Transfer Complete	Level
49	MDMA0_DST	Memory DMA Stream 0 Destination/CRC0 Output Channel Transfer Complete	Level
50	MDMA1_SRC	Memory DMA Stream 1 Source Channel Transfer Complete	Level
51	MDMA1_DST	Memory DMA Stream 1 Destination Channel Transfer Complete	Level
52	USB0_DATA	USB0 DMA Status/Transfer Complete	Level
53	SEC0_FAULT	SEC0 Fault Indication Received	Edge
54	SOFT0	Software-driven Trigger 0	
55	SOFT1	Software-driven Trigger 1	
56	SOFT2	Software-driven Trigger 2	
57	SOFT3	Software-driven Trigger 3	
58	SOFT4	Software-driven Trigger 4	
59	SOFT5	Software-driven Trigger 5	
60	SWU0_EVT	SWU0 Event	
61	SWU1_EVT	SWU1 Event	
62	SWU2_EVT	SWU2 Event	
63	SWU3_EVT	SWU3 Event	
64	SWU4_EVT	SWU4 Event	
65	SYS_ECT_MST0	Embedded Cross Trigger Master 0	
66	SYS_ECT_MST1	Embedded Cross Trigger Master 1	
67	SYS_ECT_MST2	Embedded Cross Trigger Master 2	
68	SYS_ECT_MST3	Embedded Cross Trigger Master 3	

Table 8-4: ADSP-CM40x Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
0	RCU0_SYSRST0	RCU0 System Reset Assert (Slave 0)	None
1	RCU0_SYSRST1	RCU0 System Reset Assert (Slave 1)	None

Table 8-4: ADSP-CM40x Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
2	TIMER0_TMR0_SLV0	TIMER0 Timer 0 Slave	None
3	TIMER0_TMR1_SLV0	TIMER0 Timer 1 Slave	None
4	TIMER0_TMR2_SLV0	TIMER0 Timer 2 Slave	None
5	TIMER0_TMR3_SLV0	TIMER0 Timer 3 Slave	None
6	TIMER0_TMR4_SLV0	TIMER0 Timer 4 Slave	None
7	TIMER0_TMR5_SLV0	TIMER0 Timer 5 Slave (default configuration) or Timer0 Alternate Capture Input	None
8	CNT1_UD	CNT1 Counter Up/Down Input (alternate configuration)	Level
8	TIMER0_TMR6_SLV0	TIMER0 Timer 6 Slave (default configuration) or CNT1_UD Input	None
9	TIMER0_TMR7_SLV0	TIMER0 Timer 7 Slave	None
10	SYS_C0_NMI_S0	Generate NMI on Core 0 (Slave 0)	None
11	SYS_C0_NMI_S1	Generate NMI on Core 0 (Slave 1)	None
12	TRU0_SLV0	TRU0 Interrupt Request 0	Edge
13	TRU0_SLV1	TRU0 Interrupt Request 1	Edge
14	TRU0_SLV2	TRU0 Interrupt Request 2	Edge
15	TRU0_SLV3	TRU0 Interrupt Request 3	Edge
16	SPORT0_A_DMA	SPORT0 Channel A DMA Transfer Start	Level
17	SPORT0_B_DMA	SPORT0 Channel B DMA Transfer Start	Level
18	SPORT1_A_DMA	SPORT1 Channel A DMA Transfer Start	Level
19	SPORT1_B_DMA	SPORT1 Channel B DMA Transfer Start	Level
20	SPI0_TXDMA	SPI0 TX DMA Channel Transfer Start	Level
21	SPI0_RXDMA	SPI0 RX DMA Channel Transfer Start	Level
22	SPI1_TXDMA	SPI1 TX DMA Channel Transfer Start	Level
23	SPI1_RXDMA	SPI1 RX DMA Channel Transfer Start	Level
24	ADCC0_SLV0	ADCC0 Trigger Slave 0	None
25	ADCC0_SLV1	ADCC0 Trigger Slave 1	None
26	ADCC0_SLV2	ADCC0 Trigger Slave 2	None
27	ADCC0_SLV3	ADCC0 Trigger Slave 3	None
28	ADCC0_SLV4	ADCC0 Trigger Slave 4	None
29	ADCC0_SLV5	ADCC0 Trigger Slave 5	None

Table 8-4: ADSP-CM40x Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
30	HAE0_RXDMA_CH0	HAE0 RX DMA Channel 0 Transfer Start	Level
31	HAE0_RXDMA_CH1	HAE0 RX DMA Channel 1 Transfer Start	Level
32	HAE0_TXDMA	HAE0 TX DMA Channel 0 Transfer Start	Level
33	UART0_TXDMA	UART0 Transmit DMA Transfer Start	Level
34	UART0_RXDMA	UART0 Receive DMA Transfer Start	Level
35	UART1_TXDMA	UART1 Transmit DMA Transfer Start	Level
36	UART1_RXDMA	UART1 Receive DMA Transfer Start	Level
37	UART2_TXDMA	UART2 Transmit DMA Transfer Start	Level
38	UART2_RXDMA	UART2 Receive DMA Transfer Start	Level
39	MDMA0_SRC	Memory DMA Stream 0 Source/CRC0 Input Channel Transfer Start	Level
40	MDMA0_DST	Memory DMA Stream 0 Destination/CRC0 Output Channel Transfer Start	Level
41	MDMA1_SRC	Memory DMA Stream 1 Source Channel Transfer Start	Level
42	MDMA1_DST	Memory DMA Stream 1 Destination Channel Transfer Start	Level
43	SWU0_EN	SWU0 Event	None
44	SWU1_EN	SWU1 Event	None
45	SWU2_EN	SWU2 Event	None
46	SWU3_EN	SWU3 Event	None
47	SWU4_EN	SWU4 Event	None
48	PWM0_TRIP_TRIG0	PWM0 Trip Trigger Slave 0	None
49	PWM0_TRIP_TRIG1	PWM0 Trip Trigger Slave 1	None
50	PWM0_TRIP_TRIG2	PWM0 Trip Trigger Slave 2 (default configuration) or PWM0_SYNC	None
51	PWM1_TRIP_TRIG0	PWM1 Trip Trigger Slave 0	None
52	PWM1_TRIP_TRIG1	PWM1 Trip Trigger Slave 1	None
53	PWM1_TRIP_TRIG2	PWM1 Trip Trigger Slave 2 (default Configuration) or PWM1_SYNC	None
54	PWM2_TRIP_TRIG0	PWM2 Trip Trigger Slave 0	None
55	PWM2_TRIP_TRIG1	PWM2 Trip Trigger Slave 1	None
56	PWM2_TRIP_TRIG2	PWM2 Trip Trigger Slave 2 (default Configuration) or PWM2_SYNC	None

Table 8-4: ADSP-CM40x Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
57	SINC0_SYNC0	SINC0 Synchronization Input 0	None
58	SINC0_SYNC1	SINC0 Synchronization Input 1	None
59	SYS_ECT_SLV0	Embedded Cross Trigger Slave 0	
60	SYS_ECT_SLV1	Embedded Cross Trigger Slave 1	
61	SYS_ECT_SLV2	Embedded Cross Trigger Slave 2	
62	SYS_ECT_SLV3	Embedded Cross Trigger Slave 3	
63	CPTMR0_CPT0_SLV0	CPTMR0 Capture Timer 0 Trigger 0	Level
64	CPTMR0_CPT0_SLV1	CPTMR0 Capture Timer 0 Trigger 1	Level
65	CPTMR0_CPT1_SLV0	CPTMR0 Capture Timer 1 Trigger 0	Level
66	CPTMR0_CPT1_SLV1	CPTMR0 Capture Timer 1 Trigger 1	Level
67	CPTMR0_CPT2_SLV0	CPTMR0 Capture Timer 2 Trigger 0	Level
68	CPTMR0_CPT2_SLV1	CPTMR0 Capture Timer 2 Trigger 1	Level

TRU Definitions

Trigger Master

A trigger master is any system module that provides trigger event indication to the TRU. Trigger master modules define trigger events and conditions for assertion.

Trigger Master ID

Trigger masters are assigned a unique numeric ID according to their physical connection to the TRU. Trigger master ID 0 is reserved and defined as null.

Trigger Slave

A trigger slave is any system module that receives a trigger event indication from the TRU. Trigger slave modules define a trigger event response.

TRU Block Diagram

Trigger masters and the master trigger register (MTR) generate trigger assertions. Each trigger slave has a dedicated slave select register (SSR) that specifies the unique trigger master from which it receives the trigger indication.

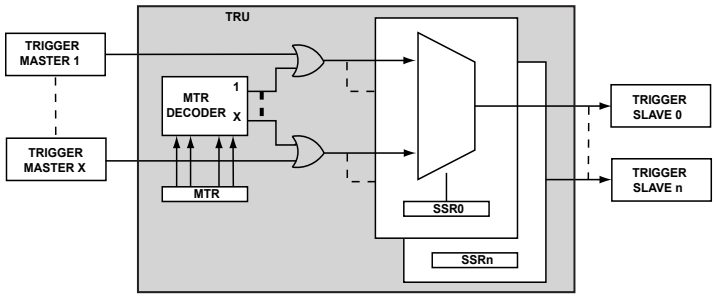

Figure 8-1: TRU Block Diagram

TRU Architectural Concepts

The TRU supports a simple trigger-in/trigger-out model for modules that comply with the triggering functional model. The TRU is the controller of the trigger system. Trigger outputs from trigger masters are mapped to trigger inputs of trigger slaves through a set of programmable registers (`TRU_SSR[n]`).

System modules are trigger master only, trigger slave only, or trigger master and trigger slave.

All of the trigger input and output signals are connected to a trigger routing unit (TRU) which manages the connections of triggers between modules.

TRU Programming Model

Implementing sequence control using the TRU requires, at a minimum, proper configuration of a trigger slave, a trigger master, and the TRU module itself. The only requirement for the configuration procedure is that the trigger master is configured and enabled as the last step.

Complete the following other steps:

- Configure the trigger slave for response to triggers.
- Configure the TRU to map the trigger master to the trigger slave through the `TRU_SSR[n]` registers.
- Configure the trigger master to generate trigger assertions.
- Alternatively, use software triggering for trigger assertion. Writing the trigger master ID to the MTR register generates software triggers.

Programming Concepts

The following concepts aid in programming the TRU.

- ***Trigger Sequence Configuration.*** A simple sequence consists of one trigger master and one trigger slave. More complex trigger sequences consist of several trigger slaves functioning as trigger slave and trigger master. Additionally, trigger sequences can loopback to the original master forming a perpetual sequence.
- ***Software Triggering.*** Writing a trigger master ID to the MTR generates a trigger within the TRU from the trigger master ID specified.

- ***Synchronization.*** The TRU can be used to coarsely synchronize events by mapping multiple trigger slaves to the same trigger master or by generating multiple trigger master assertions simultaneously through the MTR.
- ***Configuration Protection.*** The `TRU_SSR[n].LOCK` bit and the `TRU_GCTL.LOCK` bit enable register level write-protection when global lock is asserted in the SPU.

Programming Example

The following example shows the steps to create a simple trigger.

1. Write to the `TRU_GCTL` register to enable the TRU.
2. Write to the `TRU_SSR[n]` register of a specific trigger slave to assign it to a specific trigger master.
3. Enable the trigger slave to wait for and accept a trigger.
4. Enable the trigger master to generate a trigger.

TRU Event Control

The TRU is a major part of event control solutions. It is the center of the trigger functional model and can extend to support the interrupt and fault management models as well.

TRU Status and Error Signals

The TRU does not have dedicated status and error output signals other than the MMR interface. Slave errors are reported to the master over the standard bus protocol.

ADSP-CM40x TRU Register Descriptions

Trigger Routing Unit (TRU) contains the following registers.

Table 8-5: ADSP-CM40x TRU Register List

Name	Description
`TRU_ERRADDR`	Error Address Register
`TRU_GCTL`	Global Control Register
`TRU_MTR`	Master Trigger Register
`TRU_SSR[n]`	Slave Select Register
`TRU_STAT`	Status Information Register

Error Address Register

The TRU error address register (TRU_ERRADDR) holds the address from the memory mapped register access generating an access error of TRU registers.

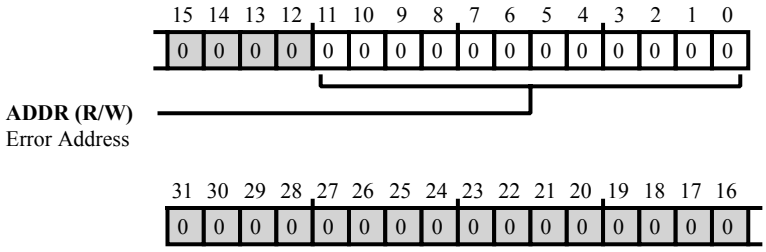

Figure 8-2: TRU_ERRADDR Register Diagram

Table 8-6: TRU_ERRADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/W)	ADDR	Error Address. The TRU_ERRADDR.ADDR holds the address from the memory mapped register access generating an access error of TRU registers. These errors occur on access to the TRU_SSR[n] or TRU_MTR registers when these registers are locked or on access to an invalid address. See the TRU_SSR[n] and TRU_MTR register descriptions for more information about locking. The TRU_ERRADDR register holds the address of the first error to occur. In the event of multiple errors occurring, the TRU_ERRADDR register contains the address of the first error. To re-enable the TRU_ERRADDR register for update, both status bits (TRU_STAT.LWERR and TRU_STAT.ADDRERR) in the TRU_STAT register must be cleared.

Global Control Register

The TRU global control register (TRU_GCTL) provides register locking, TRU reset, and TRU enable.

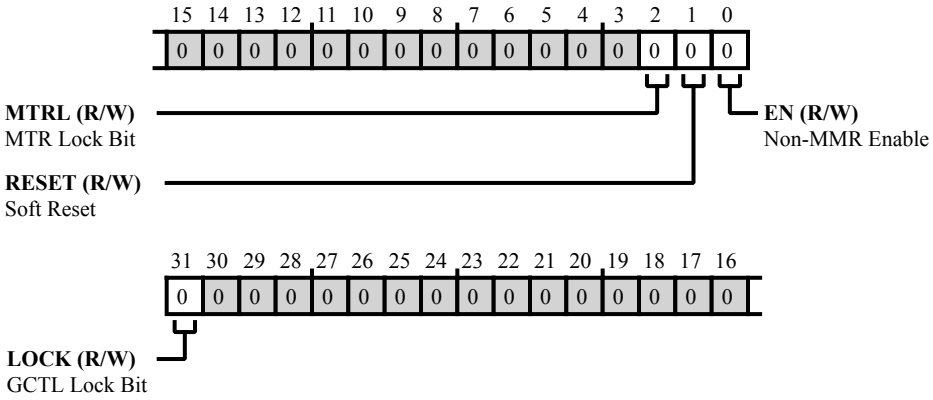

Figure 8-3: TRU_GCTL Register Diagram

Table 8-7: TRU_GCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	GCTL Lock Bit. If the global lock is enabled (SPU_CTL.GLCK bit =1) and the TRU_GCTL.LOCK bit is enabled, the TRU_GCTL register is read only.	
		0	Read write
		1	Read only
2 (R/W)	MTRL	MTR Lock Bit. If the global lock is enabled (SPU_CTL.GLCK bit =1) and the TRU_GCTL.MTRL bit is enabled, the TRU_MTR register is read only.	
		0	Read write
		1	Read only
1 (R/W)	RESET	Soft Reset. The TRU_GCTL.RESET bit is write-1-action and triggers a soft reset to all TRU registers.	
		0	No action
		1	Soft reset

Table 8-7: TRU_GCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Non-MMR Enable. The `TRU_GCTL.EN` bit is read/write and must be set for the TRU to propagate trigger events. All TRU register read/write operations continue to operate independent of the `TRU_GCTL.EN` bit.	
		0	No trigger events
		1	Propagate trigger events

Master Trigger Register

The TRU master trigger register (TRU_MTR) permits trigger generation through software by writing a trigger master ID value to one of the four fields in the TRU_MTR register. If the global lock is enabled SPU_CTL.GLCK bit =1) and the TRU_GCTL.LOCK bit is set, the TRU_MTR register is read only.

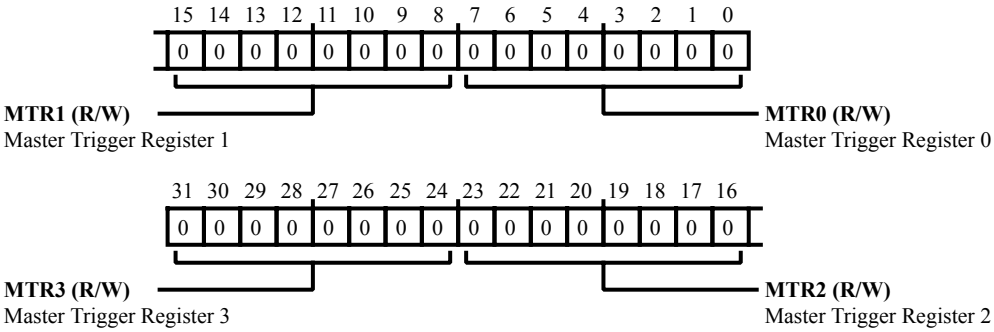

Figure 8-4: TRU_MTR Register Diagram

Table 8-8: TRU_MTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:24 (R/W)	MTR3	Master Trigger Register 3. The TRU_MTR.MTR3 bit field is the trigger master ID value for master 3.	
		0	No Master Specified
		1-255	Range of valid masters
23:16 (R/W)	MTR2	Master Trigger Register 2. The TRU_MTR.MTR2 bit field is the trigger master ID value for master 2.	
		0	No Master Specified
		1-255	Range of valid masters
15:8 (R/W)	MTR1	Master Trigger Register 1. The TRU_MTR.MTR1 bit field is the trigger master ID value for master 1.	
		0	No Master Specified
		1-255	Range of valid masters
7:0 (R/W)	MTR0	Master Trigger Register 0. The TRU_MTR.MTR0 bit field is the trigger master ID value for master 0.	
		0	No Master Specified
		1-255	Range of valid masters

Slave Select Register

The TRU slave select registers (TRU_SSR[n]) each provide slave selection and register locking.

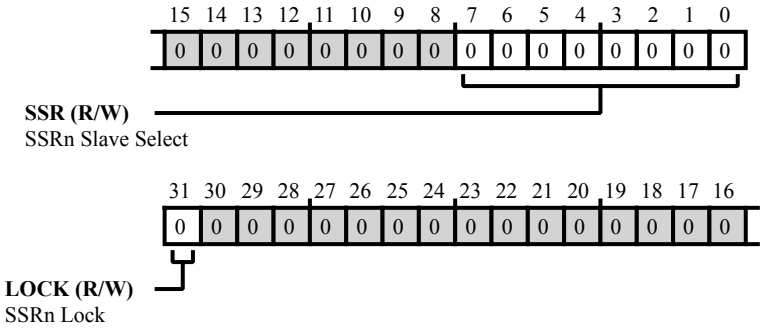

Figure 8-5: TRU_SSR[n] Register Diagram

Table 8-9: TRU_SSR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	SSRn Lock. If the global lock is enabled (SPU_CTL.GLCK bit =1) and the TRU_SSR[n].LOCK bit is enabled, the TRU_SSR[n] register is read only.	
		0	Unlock register
		1	Lock register
7:0 (R/W)	SSR	SSRn Slave Select. The TRU_SSR[n] register selects the trigger master ID to which the trigger slave responds. For example, when a TRU_SSR[n] register is set to respond to trigger master ID n, a trigger that is generated by trigger master ID n results in a trigger out to the slave.	
		0	No Master Specified
		1-255	Range of valid masters

Status Information Register

The TRU status register (TRU_STAT) contains the status of TRU_MTR and TRU_SSR[n] register writes and status of bus read/write errors.

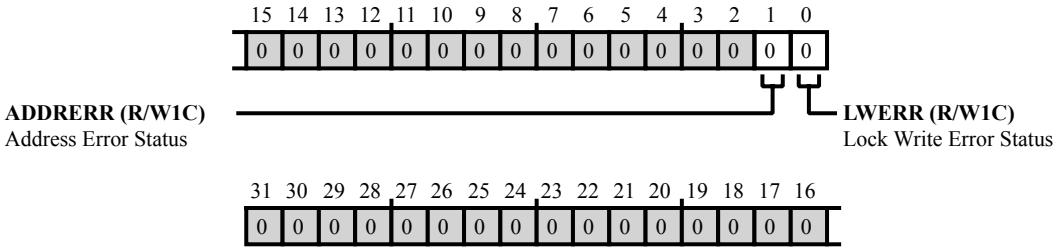

Figure 8-6: TRU_STAT Register Diagram

Table 8-10: TRU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	ADDRERR	Address Error Status. The TRU_STAT.ADDRERR bit is set when an invalid address is provided for an MMR access while the TRU is selected. Writing a one to this bit clears the error indication. The TRU_ERRADDR register also is updated when an address error occurs during an MMR access while the TRU is selected.	
		0	No error
		1	Error occurred
0 (R/W1C)	LWERR	Lock Write Error Status. If TRU_STAT.LWERR is set, a lock write error has occurred. Writing a one to this bit clears the error indication.	
		0	No error
		1	Error occurred

9 Static Memory Controller (SMC)

The static memory controller is a protocol converter and data transfer interface between the internal processor bus and the external L3 memory. It provides a glueless interface to various external memories and peripheral devices, including:

- SRAM
- ROM
- EPROM
- NOR flash memory
- FPGA/ASIC devices

The SMC acts as an SCB slave. The processor SCB interconnect fabric arbitrates accesses to the SMC. On the chip boundary, the SMC connects to an address bus, a data bus, and signal pins for memory control (such as read, write, output enable, and memory select lines).

SMC Features

SMC features include:

- 16-bit I/O width
- Provides flexible timing control through extended timing parameters
- Supports asynchronous access extension (`SMC_ARDY` pin)
- Supports 8-bit data masking writes

SMC Definitions

The timing registers contain bits toprogram the setup time, hold time, and access time for read and write access to each bankseparately. The SMC allows for different setup, hold, or access times for reads and writes. The `SMC_B0TIM` – `SMC_B3TIM` registers control the timing characteristics of the asynchronousmemory interface using the following parameter definitions. Each of these parameters can beprogrammed in terms of SCLK clock cycles.

Read setup time

Thetime between the beginning of a memory cycle ($\overline{\text{SMC\_AMS0}}$ low) and the read-enable assertion ($\overline{\text{SMC\_ARE}}$ low).

Read hold time

The time between read-enable deassertion ($\overline{\text{SMC\_ARE}}$ high) and the end of the memory cycle ($\overline{\text{SMC\_AMS0}}$ high).

Read access

The time between read-enable assertion ($\overline{\text{SMC\_ARE}}$ low) and deassertion ($\overline{\text{SMC\_ARE}}$ high).

Write setup time

The time between the beginning of a memory cycle ($\overline{\text{SMC\_AMS0}}$ low) and the write-enable assertion ($\overline{\text{SMC\_AWE}}$ low).

Write hold time

The time between write-enable deassertion ($\overline{\text{SMC\_AWE}}$ high) and the end of the memory cycle ($\overline{\text{SMC\_AMS0}}$ high).

Write access

The time between write-enable assertion ($\overline{\text{SMC\_AWE}}$ low) and deassertion ($\overline{\text{SMC\_AWE}}$ high).

The SMC provides another registerfor defining more timing characteristics of control signals by programming the extended timingregisters SMC_B0TIM – SMC_B3TIM. These registers contain bits to program following timingcharacteristics.

Pre-setup time

The number of cycles $\overline{\text{SMC\_AMS0}}$ is asserted before $\overline{\text{SMC\_AOE}}$ is asserted.

Pre-access time

The number of cycles inserted after $\overline{\text{SMC\_AOE}}$/ is deasserted, before $\overline{\text{SMC\_ARE}}$ is asserted for the next access.

Memory idle time

The number of bus idle cycles between $\overline{\text{SMC\_AMS0}}$ deasserting edge and next asserting edge.

Memory transitiontime

The number of bus idle cycles extending the idle time cycles. These idle cycles occur in the case wherea subsequent access has a different data direction or the access is to differentbank.

Bus contention

State of the bus in which more than one device on the bus attempts to place values on the bus at the sametime. For more information, see Avoiding Bus Contention.

ARDY signal

The SMC uses SMC_ARDY signal to insert wait states for slower asynchronous memories.There is no upper limit to how many wait states the ARDY signal can enter. As long as it is held,the processor waits for the access to the asynchronous memory. Once asserted, the processor accessesthe memory according to the timing diagrams. For more information, see ARDY Input Control.

SMC Functional Description

The SMC contains memory-mapped registers that control the access characteristics for each asynchronous memory bank. Different banks can be programmed in various modes and independently controlled using the functional and cycle time bit settings for each bank.

Independent bank control

The SMC provides separate sets of registers, SMC_B0CTL through SMC_B3CTL (control), SMC_B0TIM through SMC_B3TIM (timing) and SMC_B0ETIM through SMC_B3ETIM (extended timing) to control the mode and timing characteristic of each bank independently. The control registers contain bits for enabling the bank and bits for selecting mode of operation.

Bank select control signal control

The control registers also contain bits to control the type of bank select control signal. External FIFO devices often do not have a separate chip select pin. As a result, for a read, the FIFOs output enable (SMC_AOE) pin must be connected to the OR of the SMC_AMS0 and the SMC_ARE. Similarly, the write case requires an OR between SMC_AMS0 and SMC_AWE. The SMC provides this function so that an external OR gate is not required. The appropriate AMS function can be selected for each memory bank region using the SMC_B0CTL.SELCTRL bits.

SMC Architectural Concepts

The SMC can support connection to multiple different external banks, with each bank controlled by the SMC_AMS[n] chip select signal. Check the processor data sheet for details on the bank address ranges and configurations.

NOTE: The processor data sheet shows the address range allocated to each bank. It is not necessary to populate all of an enabled memory bank.

The processor does not directly support 8-bit accesses to the external memories. So, the SMC address lines start from SMC_A01; there is no SMC_A0 pin.

The SMC does indirectly support 8-bit accesses through the additional byte enable signals SMC_ABE0 and SMC_ABE1. Some 16-bit memory systems allow the processor to perform 8-bit reads and writes, which are selected through the SMC_ABE0 and SMC_ABE1 signals.

The byte enable pins are both tri-stated during all asynchronous reads and are driven low during 16-bit asynchronous writes. When an asynchronous write is made to the upper byte of a 16-bit memory, SMC_ABE1=0 and SMC_ABE0=1. When an asynchronous write is made to the lower byte of a 16-bit memory, SMC_ABE1=1 and SMC_ABE0=0.

NOTE: Some SRAM devices expect byte-enable signals to be driven low during read accesses rather than being tri-stated, which can be achieved using external pull-down resistors. For applications requiring alternate functions on the byte-enable pins during run-time where pull-down resistors are not an option, the same functionality can be achieved using the external logic shown as follows:

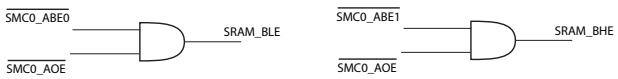

Figure 9-1: External Logic

Avoiding Bus Contention

Bus contention occurs when one device is getting off the bus and another is getting on. If the first device is slow to three-state and the second device is quick to drive, the devices contend. Bus contention causes excessive power dissipation and can lead to device failure.

There are two cases where contention can occur.

- When a read followed by a write to the same memory space occurs, there is a potential for bus contention. The data bus drivers used for the write can potentially contend with the drivers used by the memory device being read.
- When there are back-to-back reads from two different memory spaces, the two memory devices addressed by the two reads can contend at the transition between the two read operations.

To avoid contention, program the turnaround time appropriately in the extended time registers (SMC_B0ETIM – SMC_B3ETIM). The programming is done by setting the number of clock cycles between these types of accesses on a bank-by-bank basis.

The idle time bit (SMC_B0ETIM.IT) applies to similar back-to-back access types on the same bank. The transition time bit (SMC_B0ETIM.TT) applies to the SMC_B0ETIM.IT bit. For actual turnaround situations, idle time and transition time function in an accumulated fashion. The sequence of access types and times are:

- A write followed by write to same bank – SMC_B0ETIM.IT

- A read followed by read to same bank – SMC_B0ETIM.IT
- A write followed by read to same bank – SMC_B0ETIM.IT + SMC_B0ETIM.TT
- A read followed by write to same bank – SMC_B0ETIM.IT + SMC_B0ETIM.TT
- Any access to a given bank followed by any access to a different bank – SMC_B0ETIM.IT + SMC_B0ETIM.TT

The reset value of turnaround transition time is two cycles. Program the SMC_B0ETIM.TT bit to a value either greater than or equal to two cycles, depending on memory AC-timing specifications. It is important to be aware that the SMC_B0ETIM.TT bit is programmed to 0 *only* when:

- There are *either* only read accesses *or* only write accesses possible to external memory devices for the current device configuration or processor operation situation.

ARDY Input Control

Each bank can be programmed to sample the SMC_ARDY input after the read or write access timer has counted down or to ignore this input signal. If enabled and disabled at the sample window, the SMC module uses SMC_ARDY to extend the access time, as required.

The processor treats SMC_ARDY as an asynchronous input. The input must reach the desired value (either asserted or deasserted) more than two SCLK cycles before the completion of access time (scheduled rising edge of SMC_AWE or SMC_ARE). This timing determines whether the SMC extends the access with the assertion of SMC_ARDY. After the processor samples SMC_ARDY high, the total delay between SMC_ARDY going high at the pads and SMC_ARE being deasserted at the pads is a maximum of five SCLK cycles. (The memory device asserts SMC_ARDY).

Asynchronous SRAM writes are also possible with the SMC_ARDY signal enabled. In asynchronous SRAM writes, the write access is extended beyond the programmed write access cycles depending on the SMC_ARDY signal state. Once SMC_ARDY is sampled asserted, the SMC_AWE signal is deasserted after two *CLKOUT* cycles and the write access ends.

The polarity of SMC_ARDY is programmable on a per-bank basis. Since SMC_ARDY is not sampled until an access is in-progress to a bank in which the SMC_ARDY enable is asserted, it is not driven by default. When using flash memory, connect the WAIT input to SMC_ARDY.

To avoid stalls in case of erroneous SMC_ARDY behavior, set the SMC_B0CTL.RDYABTEN bit to enable the SMC_ARDY abort counter. When the abort counter is enabled, it starts counting down as soon as the programmed read/write access cycles expire. If the SMC interface does not sample the SMC_ARDY signal as asserted within 64 cycles, a counter-timeout occurs. This timeout ensures that the processor does not hang if SMC_ARDY is not sampled correctly.

SMC Operating Modes

The SMC supports the following operating modes.

- Asynchronous Flash Mode
- Asynchronous Page Mode

Asynchronous Flash Mode

When the access selected mode is asynchronous flash (SMC_B0CTL.MODE=01), external bank accesses operate the same as in standard asynchronous mode, except for the pin configuration. Use this mode when accessing burst devices in non-read array modes.

Asynchronous Page Mode

When asynchronous page mode access is selected (SMC_B0CTL.MODE=10), asynchronous page reads are enabled. The SMC module supports page sizes of 4, 8 and 16 words. When performing a page mode read, the first access in the page proceeds according to the read access time configured in SMC_B0TIM register. This access opens the page and the subsequent reads in that page have a period equal to the page wait states programmed in the SMC_B0ETIM register. Besides the start of the setup phase, the read address is incremented at the start of every page cycle.

The SMC module supports page mode access only for back-to-back accesses, such as cache line fills (16 words), 64-bit instruction reads (4 words), and DMA reads. It treats write accesses in asynchronous page mode as simple asynchronous flash write accesses.

SMC Event Control

SMC event control consists of recording status of SMC errors. Accesses to reserved locations and writes to read-only registers result in bus errors. SMC translates bus errors into internal SCB crossbar errors which get translated into interrupts. To report errors occurring in the slave memory devices (for both this memory interface and the MMR interface), the core combines the SCB crossbar response signals. This combination generates a combined error signal indication which is routed to the fault management unit.

SMC Programmable Timing Characteristics

This section describes the programmable timing characteristics for the SMC. Timing relationships depend on the programming of the SMC bank registers, whether initiation is from the core or from DMA. The relationships also depend on the sequence of transactions (read followed by read, read followed by write, and others).

NOTE: All memory control, address, and data signals are driven out of chip based on the falling edge of the *CLKOUT* signal. The *CLKOUT* signal is SCLK on the chip pins (pad delayed).

Asynchronous SRAM Reads and Writes

The ***Basic Asynchronous SRAM Write Followed by Read*** figure shows a basic single write and read operation to an external device with SMC programmed in asynchronous SRAM mode.

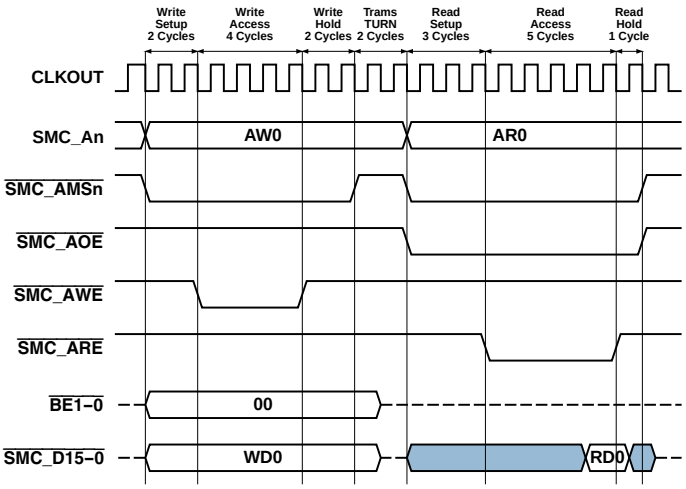

Figure 9-2: Basic Asynchronous SRAM Write Followed by Read

For the current bank, the programmed time cycles are:

- write setup time=2 cycles
- write access time=4 cycles
- write hold time is=2 cycles
- read setup time=3 cycles
- read access time=5 cycles
- read hold time=1 cycle
- turnaround transition time=2 cycles
- idle transition time=0 cycles

The asynchronous SRAM bus cycles proceed as follows.

1. At the start of the write setup period, the chip select signal ($\overline{\texttt{SMC\_AMS[n]}}$) for the target bank asserts. The write data (`WD0`), address (`AW0`), and byte enables become valid.
2. At the end of the setup phase and at the start of the write access period, the write enable ($\overline{\texttt{SMC\_AWE}}$) asserts.
3. At the end of the programmed write access, the $\overline{\texttt{SMC\_AWE}}$ signal deasserts. The target device is assumed to have captured the write data before $\overline{\texttt{SMC\_AWE}}$ deasserts.
4. At the end of the write hold period, the $\overline{\texttt{SMC\_AWE}}$ signal deasserts because the pending access is a read access, and the turnaround transition time cycles start. The write data and byte enables become invalid within 1 cycle of the $\overline{\texttt{SMC\_AMS0}}$ signal deasserting.
5. At the end of turnaround transition time, the read setup period starts with the assertion of the $\overline{\texttt{SMC\_AMS0}}$ and $\overline{\texttt{SMC\_AOE}}$ signals and a new read address (`AR0`) presented on the address bus.

6. At the start of the read access period, the read enable signal, SMC_ARE asserts.
7. At the end of the read access period, the SMC_ARE signal deasserts and the read hold period starts. Read data is latched along with SMC_ARE deasserting.
8. At the end of the read hold period, the SMC pulls the SMC_AMS[n] signal high and appends turnaround transition cycles unless there is a pending read request to the same bank.

Asynchronous SRAM Reads with IDLE Transition Cycles Inserted

The ***Asynchronous SRAM Read with IDLE Transition*** figure shows two consecutive asynchronous SRAM modes reads to the same bank separated by programmed IDLE transition time cycles.

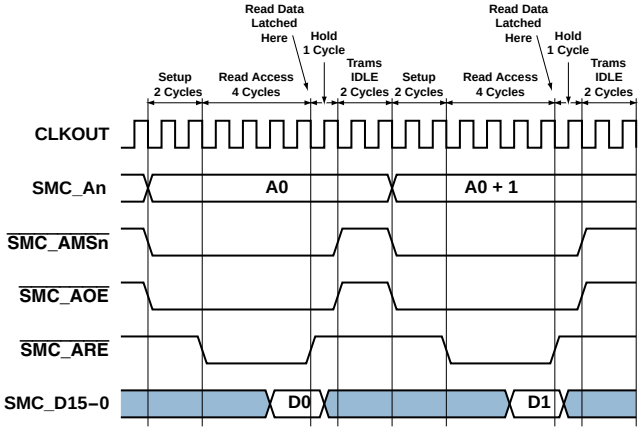

Figure 9-3: Asynchronous SRAM Read with IDLE Transition

Programmed cycle times are:

- SMC_B0TIM.RST=2 cycles
- SMC_B0TIM.RAT=4 cycles
- SMC_B0TIM.RHT=1 cycle
- IDLE transition time=2 cycles

At the start of the IDLE transition cycle, the SMC deasserts the SMC_AMS[n] and SMC_AOE signal. The setup period of the second read starts at the end of the IDLE transition cycle with the assertion of the SMC_AMS[n] and SMC_AOE signals and a new address on the address bus.

High-Speed Asynchronous SRAM Read Burst

The ***Fast Asynchronous SRAM Reads, Burst of Four Word*** figure shows a high-speed asynchronous SRAM read bus cycle. This read bus cycle is typical for SRAM devices with small access times connecting through SCB read bursts, especially for boot purposes.

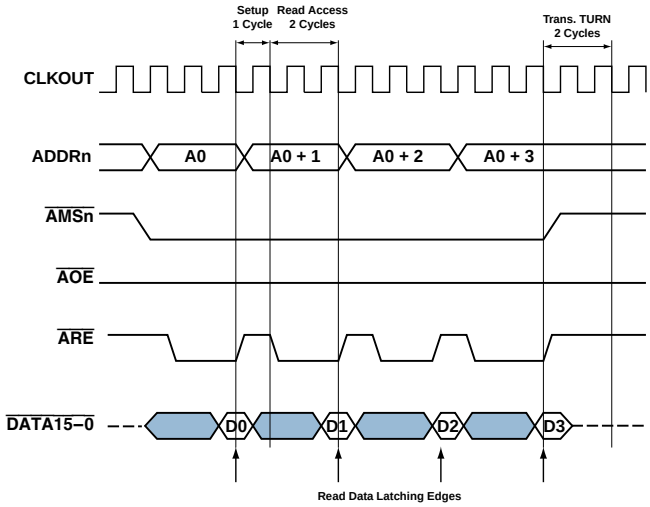

Figure 9-4: Fast Asynchronous SRAM Reads, Burst of Four Word

In this case, the target SMC bank has been programmed with:

- read setup time=1 cycle
- read access time=2 cycles
- read hold time=0
- SMC_B0ETIM.PREAT=0
- SMC_B0ETIM.PREST=0
- IDLE transition time=0

The SMC_AMS[n] signal asserts at the start of the setup cycle of the first read out of the burst. Since the hold time and the IDLE transition time have been programmed to 0, the SMC_AMS[n] signal does not deassert until the entire set of reads concludes. Only the SMC_ARE signal deasserts periodically for 1 cycle for the setup period. The read address changes to the next address at the start of each individual setup cycle. Read data words are latched at the end of each individual read access period.

High-Speed Asynchronous SRAM Writes

High-speed asynchronous SRAM writes are similar to the high-speed read accesses. The ***Fast Asynchronous SRAM Writes*** figure shows the bus protocol for a write burst of 4 words. Here, the write setup time is 1 cycle and the write access time has been programmed to 2 cycles. Write hold time, pre-access time, pre-setup time, and idle transition time are programmed to 0.

The chip select signal, SMC_AMS[n], asserts at the start of the entire write burst and deasserts only at the end of the last individual write access period. Write address, byte enables and write data for each individual write access are presented onto the bus at the start of each individual write setup cycle. The SMC_AWE signal asserts for the write access period and deasserts during the setup period for each individual data write.

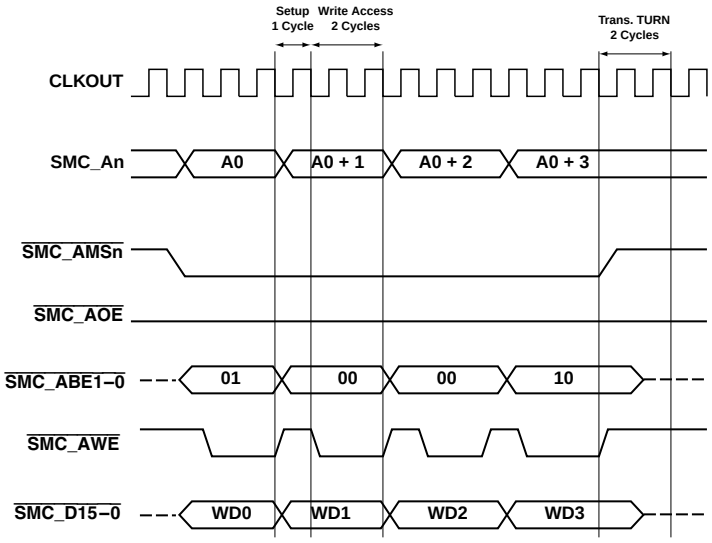

Figure 9-5: Fast Asynchronous SRAM Writes

Asynchronous SRAM Reads with ARDY

The ***Asynchronous SRAM Read with ARDY*** figure shows an extended asynchronous SRAM read bus cycle with `SMC_ARDY` enabled.

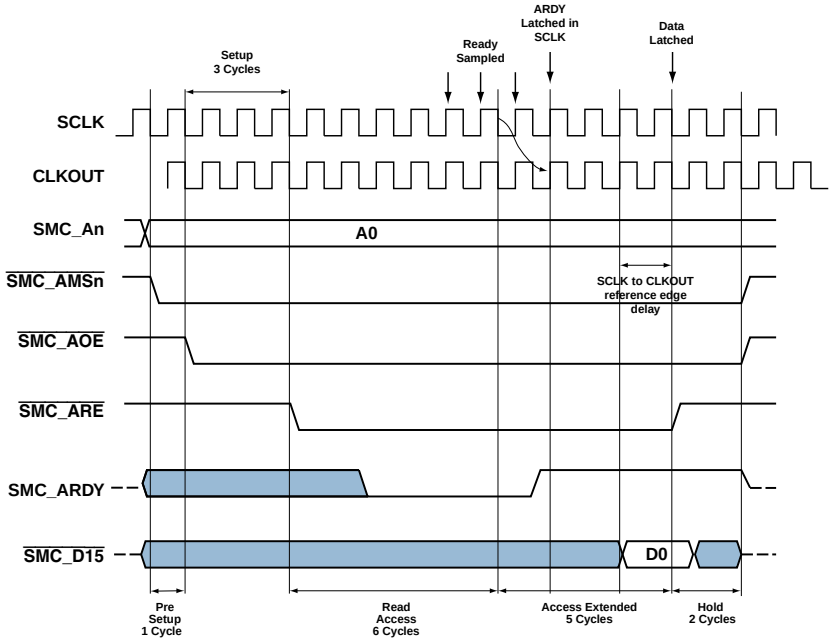

Figure 9-6: Asynchronous SRAM Read with ARDY

The programmed SMC bank control parameters are:

- pre-setup time=1 cycle
- read setup time=3 cycles

- read access time=6 cycles
- read hold time=2 cycles
- SMC_B0CTL.RDYPOL=1 (memory is ready when SMC_ARDY=1)

The bus cycles proceed as follows:

- At the start of the pre-setup phase, SMC_AMS[n] asserts, and read address SMC_A01 is presented on the address bus.
- At the start of the setup period, SMC_AOE asserts.
- At the start of the read access, SMC_ARE asserts.
- The *CLKOUT* signal is SCLK which is driven out of the pads. The *CLKOUT* signal falling edge can be delayed from the internal SCLK falling edge. See the data sheet for the specification related to this delay. All output signals out of the pads, for example SMC_ARE and SMC_AOE, are driven regarding the falling edge of *CLKOUT*.
- The SMC starts sampling the SMC_ARDY signal on every rising edge of internal SCLK two cycles before the programmed number of read access cycles expires. The read access is extended (SMC_ARE is kept asserted) until the SMC samples SMC_ARDY high.
- Once the SMC samples the SMC_ARDY signal (asserted by memory device) high in SCLK, the read signal is pulled off internally in the SCLK domain. The total delay between the SMC_ARDY signal going high at the pads and the deassertion of the SMC_ARE signal at the pads can be a maximum of five SCLK cycles.
- Read data is latched at the falling edge of *CLKOUT* on the same edge where the SMC deasserts SMC_ARE.
- Hold bus cycles start after the SMC deasserts the SMC_ARE signal.
- At the end of the hold period, the SMC_AMS[n] and SMC_AOE signals deassert and the SMC goes into the transition state.

Asynchronous Flash Reads

The ***Asynchronous Flash Read with Pre-Setup and Pre-Access Cycles*** figure illustrates a single asynchronous flash mode read bus cycle.

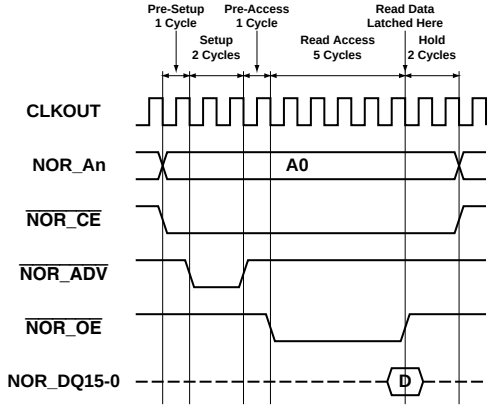

Figure 9-7: Asynchronous Flash Read with Pre-Setup and Pre-Access Cycles

In this case, the target SMC bank has been programmed with:

- pre-setup time=1 cycle
- read setup time=2 cycles
- pre-access time=1 cycle
- read access time=5 cycles
- read hold time=2 cycles

The read bus cycle is almost identical to the asynchronous SRAM read bus cycle. The only difference is the behavior of the SMC_AOE signal which the SMC uses as the flash address valid SMC_NORDV signal. The flash address valid SMC_NORDV signal asserts at the start of the setup cycle and deasserts at the end of the setup cycle.

The pre-access cycle inserts one-cycle gap between the deassertion of flash address valid SMC_NORDV and the assertion of the flash read strobe NOR_OE at the start of read access. The SMC also uses asynchronous flash reads with SMC_ARDY enabled for flash devices which use SMC_NORWT in asynchronous mode. In this case, the read bus cycle operation is identical to the asynchronous SRAM with SMC_ARDY enabled except for the SMC_AOE/SMC_NORDV signal behavior.

The ***32-bit Asynchronous Flash Read*** figure shows a 32-bit read access to a flash device in asynchronous mode which is split into two 16-bit external memory accesses. For this bank, read setup and read hold are programmed as two cycles whereas the read access time is five cycles. The flash device chip select signal (NOR_CE) remains asserted for the entire duration of both read accesses. NOR_CE is deasserted at the end of the hold period of the second read access. The SMC asserts the SMC_NORDV signal during the setup phase of both read accesses. Read data is latched at the end of the read access period.

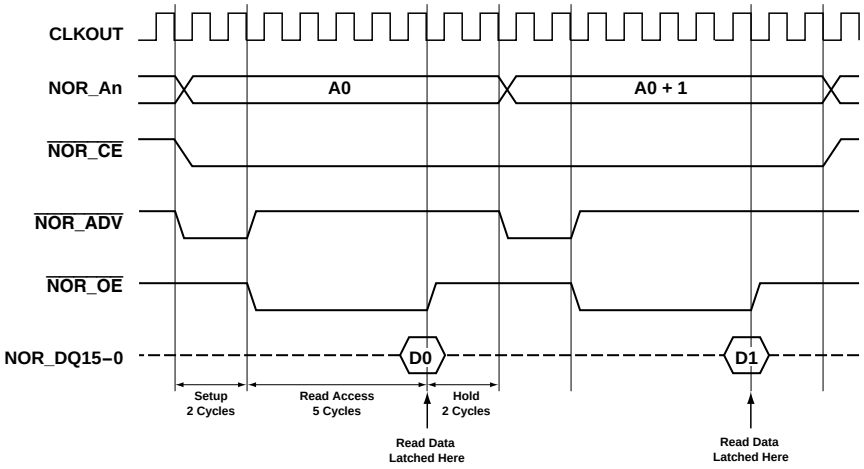

Figure 9-8: 32-bit Asynchronous Flash Read

Asynchronous Flash Writes

The ***Asynchronous Flash Write Operation*** figure shows a single asynchronous flash write bus cycle.

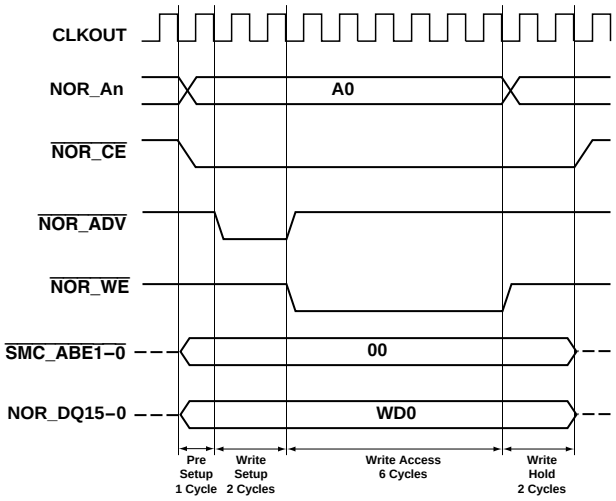

Figure 9-9: Asynchronous Flash Write Operation

For this example, the SMC has been programmed with:

- pre-setup time=1 cycle
- write setup time=2 cycles
- write access time=6 cycles
- write hold time=2 cycles
- pre-access time=0

The asynchronous flash write bus cycle is again almost identical to the asynchronous SRAM write. The SMC_AWE pin is connected to flash write enable signal (NOR_WE). However, in asynchronous flash writes, the SMC uses the SMC_AOE signal as the address valid signal (SMC_NORDV). The SMC_AOE signal asserts during the setup period, unlike in asynchronous SRAM writes where the SMC_AOE signal never asserts.

Asynchronous Flash Page Mode Reads

The ***Asynchronous Page Mode Read Bus Cycle*** figure shows an asynchronous page mode bus read cycle for a burst of five reads. The reads are split into four reads followed by a single read.

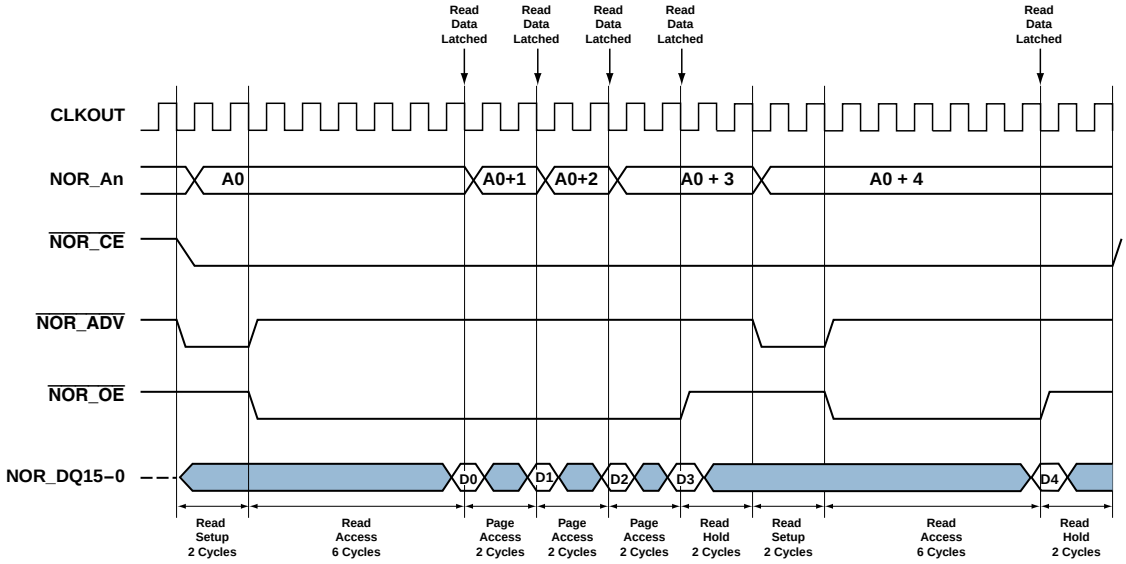

Figure 9-10: Asynchronous Page Mode Read Bus Cycle

The programmed bank parameters are:

- read setup time=2 cycles
- read access time=6 cycles
- page wait=2 cycles
- hold time=2 cycles

The maximum number of read bursts in a total page access depends on the bank SMC_B0CTL.PGSZ bits (00=4 words, 01=8 words, 1x=16 words). The first read access is extended for three more page-read cycles whose period is equal to the page wait states. Besides the start of the setup phase, the read address is incremented at the start of every page cycle. Read data is latched with the falling edge of *CLKOUT* the end of the read access period, and also at the end of the page cycles.

Asynchronous FIFO Reads and Writes

The ***Asynchronous FIFO Read Bus Cycles*** figure shows read bus cycles for an asynchronous FIFO device. The SMC bank is programmed in asynchronous SRAM mode, with SMC_B0CTL.SELCTRL =01 (SMC_AMS[n] is OR'ed with SMC_ARE).

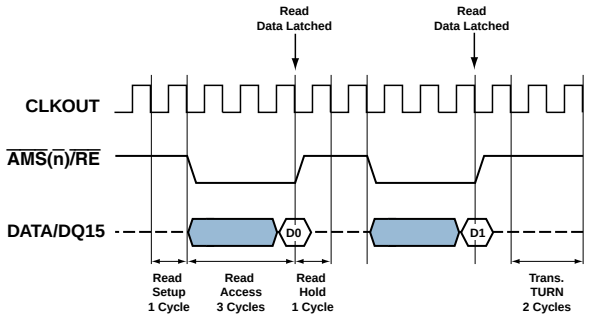

Figure 9-11: Asynchronous FIFO Read Bus Cycles

Other settings are:

- read setup time=1 cycle
- read access time=3 cycles
- read hold time=1 cycle
- idle transition time=0 cycles
- turnaround transition time=2 cycles

The SMC_AMS[n] signal connects to the read enable (RE) of the FIFO device, and the data bus connects to the output data bus (DQ) of the FIFO. The SMC_AMS[n] signal or the FIFO read strobe asserts only during the read access. Read data is latched at the falling edge of *CLKOUT* at the end of the read access, when the SMC deasserts SMC_AMS[n].

The ***Asynchronous FIFO Write Bus Cycles*** figure illustrates write bus cycles for an asynchronous FIFO device. The SMC bank is programmed in asynchronous SRAM mode, with SMC_B0CTL.SELCTRL = 10 (SMC_AMS[n] is OR-ed with SMC_AWE). Other settings are:

- write setup time=1 cycle
- write access time=3 cycles
- write hold time=1 cycle
- idle transition time=0
- turnaround transition time=2 cycles

The SMC_AMS[n] signal connects to the write enable ($\overline{WE}$) of the FIFO device. The data bus connects to the input data bus (DIN) of the FIFO. The SMC_AMS[n] signal or the FIFO write strobe asserts only during the write access.

However, the SMC asserts write data at the start of the setup cycle and removes it at the end of the hold period for each individual write access.

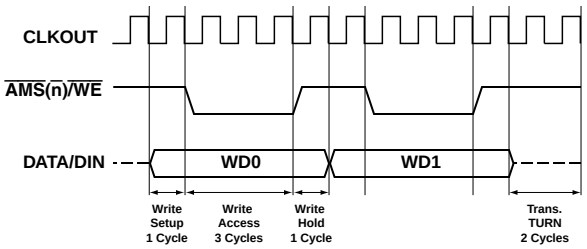

Figure 9-12: Asynchronous FIFO Write Bus Cycles

SMC Programming Model

Following are general guidelines for configuring and enabling the SMC interface. Failure to follow these guidelines can lead to erroneous behavior.

- In asynchronous page mode, always program `SMC_B0CTL.RDYEN` to 0.
- Enable the ARDY abort counter (the `SMC_B0CTL.RDYABTEN` bit =1) whenever the `SMC_ARDY` signal is enabled (the `SMC_B0CTL.RDYEN` is set to 1). Doing so ensures that the interface does not hang due to erroneous `SMC_ARDY` signal behavior or erroneous sampling of the `SMC_ARDY` signal.
- Do not program read access time (`SMC_B0TIM.RAT`), write access time (`SMC_B0TIM.WAT`), read setup time (`SMC_B0TIM.RST`), and write setup time (`SMC_B0TIM.WST`) to 0.
- Never program page mode wait-states (`SMC_B0ETIM.PGWS`) to 0 or 1.
- Program the page size bits (`SMC_B0CTL.PGSZ`) to match the configurations of the flash device connecting to the SMC interface.
- Select the `SMC_B0CTL.RDYPOL` bit to be the complement of the `WAIT` polarity that is configured in the flash device.
- In asynchronous SRAM and asynchronous flash modes with `SMC_ARDY` enabled, and where `SMC_B0TIM.RHT`, `SMC_B0TIM.WHT`, `SMC_B0TIM.RAT`, and `SMC_B0TIM.WAT` are the read and write hold and access times and `SMC_B0ETIM.IT` and `SMC_B0ETIM.TT` are the idle and transition times, ensure that the following conditions are true:
  - `SMC_B0TIM.RHT` + `SMC_B0ETIM.IT` + `SMC_B0ETIM.TT`>= 2
  - `SMC_B0TIM.WHT` + `SMC_B0ETIM.IT` + `SMC_B0ETIM.TT`>= 2
  - `SMC_B0TIM.RAT`>= 5
  - `SMC_B0TIM.WAT`>= 5

ADSP-CM40x SMC Register Descriptions

Static Memory Controller (SMC) contains the following registers.

Table 9-1: ADSP-CM40x SMC Register List

Name	Description
SMC_B0CTL	Bank 0 Control Register
SMC_B0ETIM	Bank 0 Extended Timing Register
SMC_B0TIM	Bank 0 Timing Register
SMC_B1CTL	Bank 1 Control Register
SMC_B1ETIM	Bank 1 Extended Timing Register
SMC_B1TIM	Bank 1 Timing Register
SMC_B2CTL	Bank 2 Control Register
SMC_B2ETIM	Bank 2 Extended Timing Register
SMC_B2TIM	Bank 2 Timing Register
SMC_B3CTL	Bank 3 Control Register
SMC_B3ETIM	Bank 3 Extended Timing Register
SMC_B3TIM	Bank 3 Timing Register

Bank 0 Control Register

The SMC_B0CTL register enables bank 0 accesses and configures the memory access features for this bank.

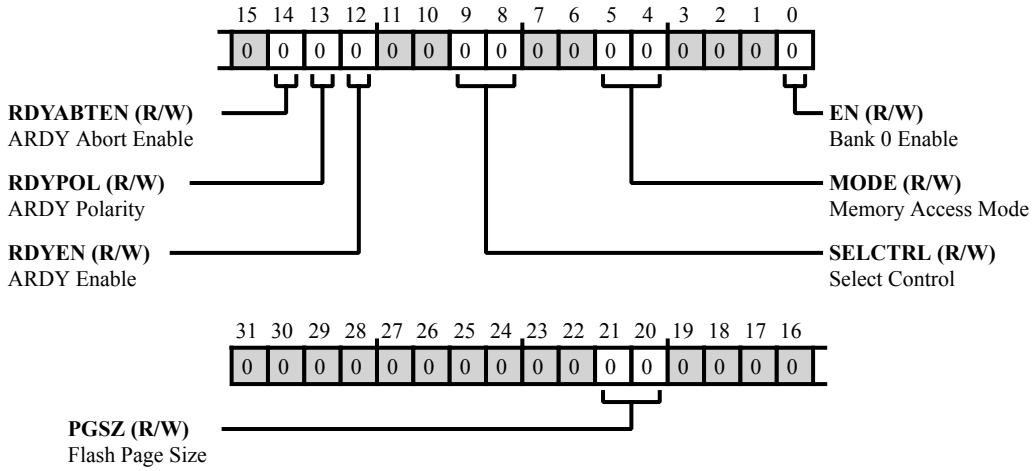

Figure 9-13: SMC_B0CTL Register Diagram

Table 9-2: SMC_B0CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/W)	PGSZ	Flash Page Size. The SMC_B0CTL.PGSZ bits select the flash page size, if page flash or sync burst flash protocol has been enabled (SMC_B0CTL.MODE > 1). Note that the SMC_B0CTL.PGSZ bits must be set to match the flash protocol of the external flash memory device in the system. The typical SMC_B0CTL.PGSZ selection for external devices supporting async flash or async flash page protocols is 4 or 8 words. The typical SMC_B0CTL.PGSZ selection for external devices supporting sync burst flash protocol is 16 words.	
		0	4 words
		1	8 words
		2	16 words
		3	16 words
14 (R/W)	RDYABTEN	ARDY Abort Enable. The SMC_B0CTL.RDYABTEN bit enables the abort counter for the SMC_ARDY pin, if enabled (SMC_B0CTL.RDYEN =1). After SMC_B0TIM.RAT or SMC_B0TIM.WAT cycles, the SMC starts sampling the SMC_ARDY pin and starts the abort down counter (if enabled). The abort count is 64 cycles of SCLK. If the SMC detects that SMC_ARDY remains de-asserted when the counter expires, the SMC aborts the access and returns an error response back on the system bus.	
		0	Disable abort counter
		1	Enable abort counter

Table 9-2: SMC_B0CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	RDYPOL	ARDY Polarity. The SMC_B0CTL.RDYPOL bit selects the polarity (active high or low) for the SMC_ARDY pin, if enabled (SMC_B0CTL.RDYEN =1). When the SMC samples the SMC_ARDY pin in the selective active state, the transaction completes.	
		0	Low active ARDY
		1	High active ARDY
12 (R/W)	RDYEN	ARDY Enable. The SMC_B0CTL.RDYEN bit enables SMC_ARDY pin operation for bank 0 accesses. When enabled, the SMC uses SMC_ARDY (after the access time countdown) to determine completion of access to this memory bank. When disabled, the SMC ignores SMC_ARDY for accesses to this memory bank.	
		0	Disable ARDY
		1	Enable ARDY
9:8 (R/W)	SELCTRL	Select Control. The SMC_B0CTL.SELCTRL bits select the handling of the $\overline{\text{SMC\_AMS[n]}}$, $\overline{\text{SMC\_ARE}}$, $\overline{\text{SMC\_AOE}}$, and $\overline{\text{SMC\_AWE}}$ pins for memory access control.	
		0	AMS0 only
		1	AMS0 ORed with ARE
		2	AMS0 ORed with AOE
		3	AMS0 ORed with AWE
5:4 (R/W)	MODE	Memory Access Mode. The SMC_B0CTL.MODE bits select the protocol the SMC uses for static memory read/write access. Note that the write protocol for async flash, async flash page, and sync burst flash are all similar; only the read protocols differ for these modes.	
		0	Async SRAM protocol
		1	Async flash protocol
		2	Async flash page protocol
		3	Reserved
0 (R/W)	EN	Bank 0 Enable. The SMC_B0CTL.EN bit enables accesses to the memory in bank 0. When this bit is disabled, accesses to bank 0 return an error response.	
		0	Disable access
		1	Enable access

Bank 0 Extended Timing Register

The SMC_B0ETIM register configures extensions to access times and idle times, augmenting the setup, hold, and access times configured with the SMC_B0TIM register.

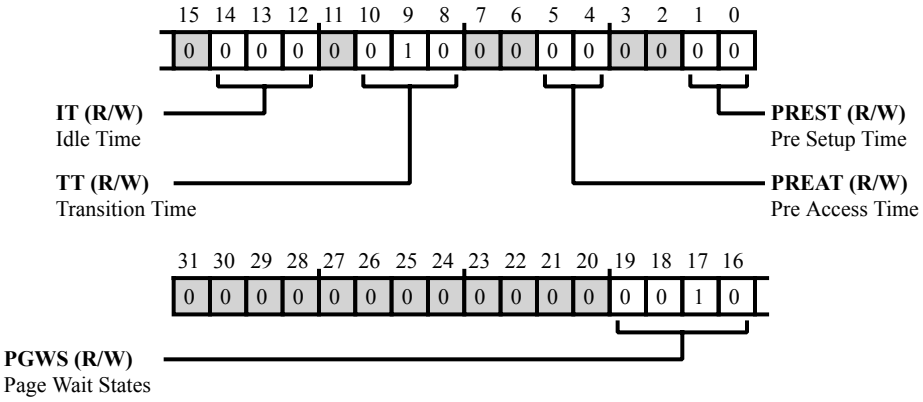

Figure 9-14: SMC_B0ETIM Register Diagram

Table 9-3: SMC_B0ETIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16 (R/W)	PGWS	Page Wait States. The SMC_B0ETIM.PGWS bits select a page access extension time (in SCLK cycles) that the SMC waits during read accesses when configured for flash page protocol (SMC_B0CTL.MODE =2). The wait time is from 2 to 15 SCLK cycles.	
		0	Not supported
		1	Not supported
		2-15	2-15 SCLK clock cycles
14:12 (R/W)	IT	Idle Time. The SMC_B0ETIM.IT bits select a bus idle time (in SCLK cycles) that the SMC waits between de-asserting the SMC_AMS[n] pin and asserting the SMC_AMS[n] pin for the next access. Note that the SMC_B0ETIM.IT period may be extended using the SMC_B0ETIM.TT selection. The idle time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		7	7 SCLK clock cycles

Table 9-3: SMC_B0ETIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10:8 (R/W)	TT	Transition Time. The SMC_B0ETIM.TT bits select a bus idle time (in SCLK cycles) that the SMC extends the SMC_B0ETIM.IT to allow for the subsequent access either using a different transfer direction or accessing a different bank. The transition time is from 1 to 7 SCLK cycles.	
		0	No bank transition
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The SMC_B0ETIM.PREAT bits select the pre-access time (in SCLK cycles) that the SMC waits after de-asserting the SMC_AOE/ADV pin before asserting the SMC_ARE/SMC_AWE pin for the current access. The pre-access time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles
1:0 (R/W)	PREST	Pre Setup Time. The SMC_B0ETIM.PREST bits select the pre-setup time (in SCLK cycles) that the SMC asserts the SMC_AMS[n] pin before asserting the SMC_AOE/ADV pin for an access. The pre-setup time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles

Bank 0 Timing Register

The SMC_B0TIM register configures bank 0 read and write access, setup, and hold timing for this bank. Note that read and write timing configurations are independent and may differ.

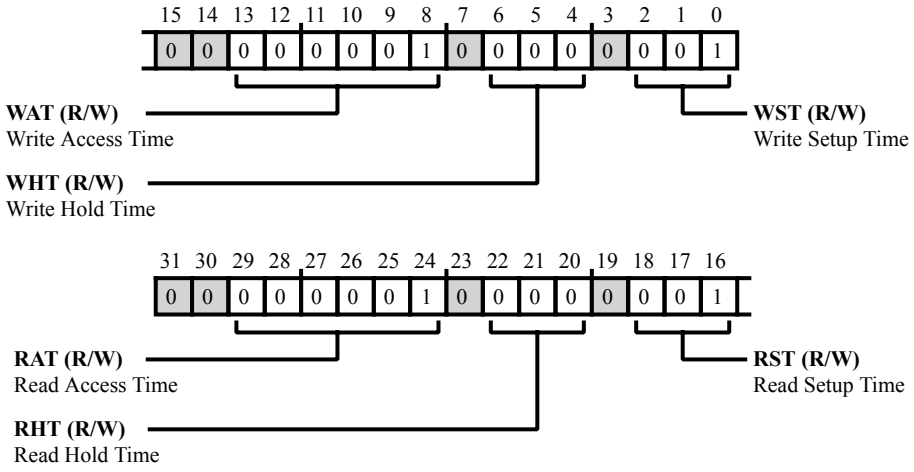

Figure 9-15: SMC_B0TIM Register Diagram

Table 9-4: SMC_B0TIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29:24 (R/W)	RAT	Read Access Time. The SMC_B0TIM.RAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_ARE pin for a read access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
22:20 (R/W)	RHT	Read Hold Time. The SMC_B0TIM.RHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the SMC_ARE pin before asserting the SMC_AOE pin for the next access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

Table 9-4: SMC_B0TIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	RST	Read Setup Time. The SMC_B0TIM.RST bits select the setup time (in SCLK cycles) that the SMC asserts the SMC_AOE pin before asserting the SMC_ARE pin for an access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
13:8 (R/W)	WAT	Write Access Time. The SMC_B0TIM.WAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_AWE pin for a write access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
6:4 (R/W)	WHT	Write Hold Time. The SMC_B0TIM.WHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the SMC_AWE pin before de-asserting the SMC_AOE pin for the current access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
2:0 (R/W)	WST	Write Setup Time. The SMC_B0TIM.WST bits select the setup time (in SCLK cycles) that the SMC asserts the SMC_AOE pin before asserting the SMC_AWE pin for a write access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

Bank 1 Control Register

The SMC_B1CTL register enables bank 1 accesses and configures the memory access features for this bank.

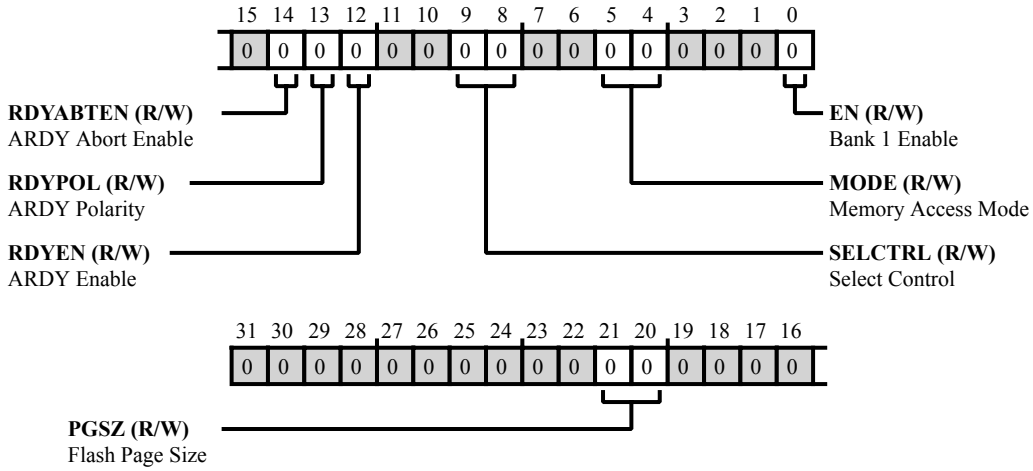

Figure 9-16: SMC_B1CTL Register Diagram

Table 9-5: SMC_B1CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/W)	PGSZ	Flash Page Size. The SMC_B1CTL.PGSZ bits select the flash page size, if page flash or sync burst flash protocol has been enabled (SMC_B1CTL.MODE > 1). Note that the SMC_B1CTL.PGSZ bits must be set to match the flash protocol of the external flash memory device in the system. The typical SMC_B1CTL.PGSZ selection for external devices supporting async flash or async flash page protocols is 4 or 8 words. The typical SMC_B1CTL.PGSZ selection for external devices supporting sync burst flash protocol is 16 words.	
		0	4 words
		1	8 words
		2	16 words
		3	16 words
14 (R/W)	RDYABTEN	ARDY Abort Enable. The SMC_B1CTL.RDYABTEN bit enables the abort counter for the SMC_ARDY pin, if enabled (SMC_B1CTL.RDYEN =1). After SMC_B1TIM.RAT or SMC_B1TIM.WAT cycles, the SMC starts sampling the SMC_ARDY pin and starts the abort down counter (if enabled). The abort count is 64 cycles of SCLK. If the SMC detects that SMC_ARDY remains de-asserted when the counter expires, the SMC aborts the access and returns an error response back on the system bus.	
		0	Disable abort counter
		1	Enable abort counter

Table 9-5: SMC_B1CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	RDYPOL	ARDY Polarity. The SMC_B1CTL.RDYPOL bit selects the polarity (active high or low) for the SMC_ARDY pin, if enabled (SMC_B1CTL.RDYEN =1). When the SMC samples the SMC_ARDY pin in the selective active state, the transaction completes.	
		0	Low active ARDY
		1	High active ARDY
12 (R/W)	RDYEN	ARDY Enable. The SMC_B1CTL.RDYEN bit enables SMC_ARDY pin operation for bank 1 accesses. When enabled, the SMC uses SMC_ARDY (after the access time countdown) to determine completion of access to this memory bank. When disabled, the SMC ignores SMC_ARDY for accesses to this memory bank.	
		0	Disable ARDY
		1	Enable ARDY
9:8 (R/W)	SELCTRL	Select Control. The SMC_B1CTL.SELCTRL bits select the handling of the $\overline{\text{SMC\_AMS[n]}}$, $\overline{\text{SMC\_ARE}}$, $\overline{\text{SMC\_AOE}}$, and $\overline{\text{SMC\_AWE}}$ pins for memory access control.	
		0	AMS1 only
		1	AMS1 ORed with ARE
		2	AMS1 ORed with AOE
		3	AMS1 ORed with AWE
5:4 (R/W)	MODE	Memory Access Mode. The SMC_B1CTL.MODE bits select the protocol the SMC uses for static memory read/write access. Note that the write protocol for async flash, async flash page, and sync burst flash are all similar; only the read protocols differ for these modes.	
		0	Async SRAM protocol
		1	Async flash protocol
		2	Async flash page protocol
		3	Reserved
0 (R/W)	EN	Bank 1 Enable. The SMC_B1CTL.EN bit enables accesses to the memory in bank 1. When this bit is disabled, accesses to bank 1 return an error response.	
		0	Disable access
		1	Enable access

Bank 1 Extended Timing Register

The SMC_B1ETIM register configures extensions to access times and idle times, augmenting the setup, hold, and access times configured with the SMC_B1TIM register.

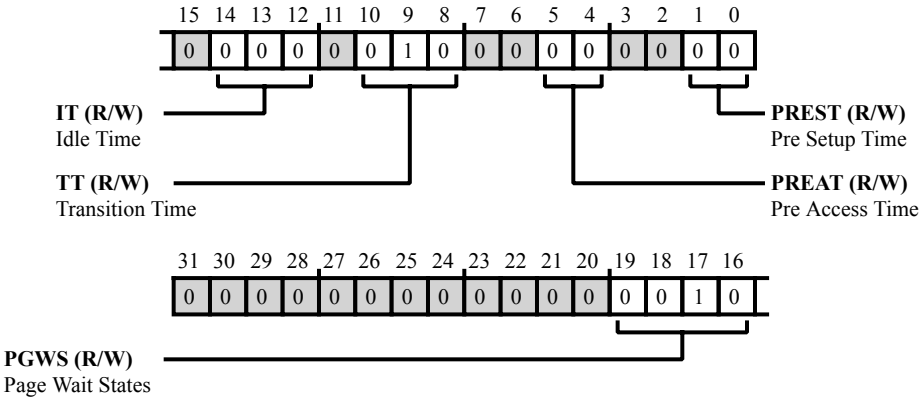

Figure 9-17: SMC_B1ETIM Register Diagram

Table 9-6: SMC_B1ETIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16 (R/W)	PGWS	Page Wait States. The SMC_B1ETIM.PGWS bits select a page access extension time (in SCLK cycles) that the SMC waits during read accesses when configured for flash page protocol (SMC_B1CTL.MODE =2). The wait time is from 2 to 15 SCLK cycles.	
		0	Not supported
		1	Not supported
		2-15	2-15 SCLK clock cycles
14:12 (R/W)	IT	Idle Time. The SMC_B1ETIM.IT bits select a bus idle time (in SCLK cycles) that the SMC waits between de-asserting the SMC_AMS[n] pin and asserting the SMC_AMS[n] pin for the next access. Note that the SMC_B1ETIM.IT period may be extended using the SMC_B1ETIM.TT selection. The idle time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		7	7 SCLK clock cycles

Table 9-6: SMC_B1ETIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10:8 (R/W)	TT	Transition Time. The SMC_B1ETIM.TT bits select a bus idle time (in SCLK cycles) that the SMC extends the SMC_B1ETIM.IT to allow for the subsequent access either using a different transfer direction or accessing a different bank. The transition time is from 1 to 7 SCLK cycles.	
		0	No bank transition
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The SMC_B1ETIM.PREAT bits select the pre-access time (in SCLK cycles) that the SMC waits after de-asserting the SMC_AOE/ADV pin before asserting the SMC_ARE/SMC_AWE pin for the current access. The pre-access time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles
1:0 (R/W)	PREST	Pre Setup Time. The SMC_B1ETIM.PREST bits select the pre-setup time (in SCLK cycles) that the SMC asserts the SMC_AMS[n] pin before asserting the SMC_AOE/ADV pin for an access. The pre-setup time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles

Bank 1 Timing Register

The SMC_B1TIM register configures bank 1 read and write access, setup, and hold timing for this bank. Note that read and write timing configurations are independent and may differ.

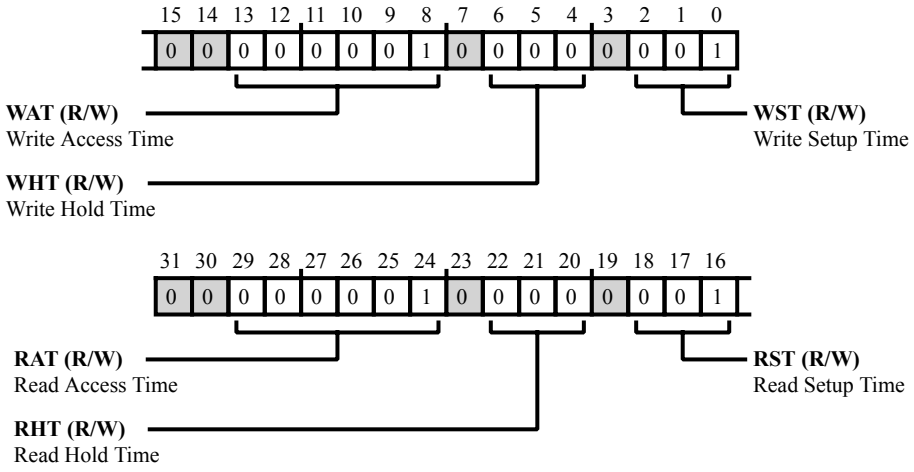

Figure 9-18: SMC_B1TIM Register Diagram

Table 9-7: SMC_B1TIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29:24 (R/W)	RAT	Read Access Time. The SMC_B1TIM.RAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_ARE pin for a read access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
22:20 (R/W)	RHT	Read Hold Time. The SMC_B1TIM.RHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the SMC_ARE pin before asserting the SMC_AOE pin for the next access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

Table 9-7: SMC_B1TIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	RST	Read Setup Time. The SMC_B1TIM.RST bits select the setup time (in SCLK cycles) that the SMC asserts the SMC_AOE pin before asserting the SMC_ARE pin for an access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
13:8 (R/W)	WAT	Write Access Time. The SMC_B1TIM.WAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_AWE pin for a write access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
6:4 (R/W)	WHT	Write Hold Time. The SMC_B1TIM.WHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the SMC_AWE pin before de-asserting the SMC_AOE pin for the current access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
2:0 (R/W)	WST	Write Setup Time. The SMC_B1TIM.WST bits select the setup time (in SCLK cycles) that the SMC asserts the SMC_AOE pin before asserting the SMC_AWE pin for a write access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

Bank 2 Control Register

The SMC_B2CTL register enables bank 2 accesses and configures the memory access features for this bank.

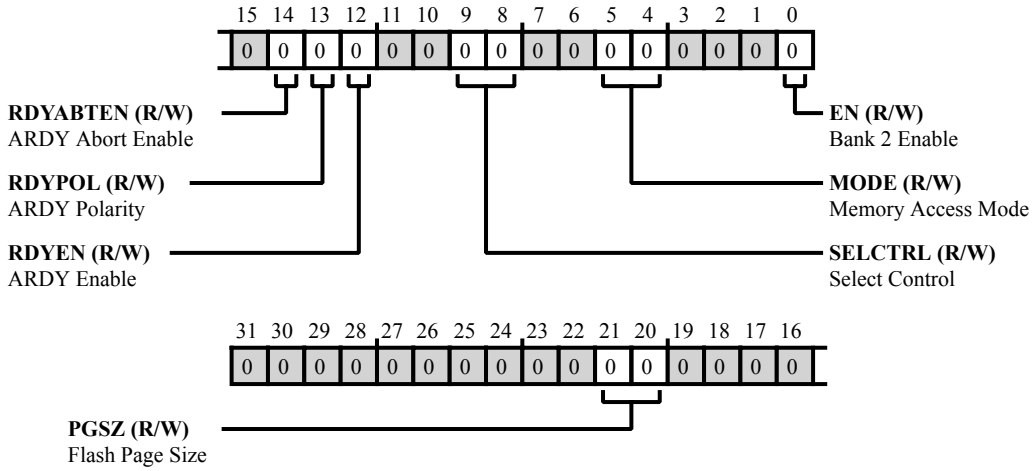

Figure 9-19: SMC_B2CTL Register Diagram

Table 9-8: SMC_B2CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/W)	PGSZ	Flash Page Size. The SMC_B2CTL.PGSZ bits select the flash page size, if page flash or sync burst flash protocol has been enabled (SMC_B2CTL.MODE > 1). Note that the SMC_B2CTL.PGSZ bits must be set to match the flash protocol of the external flash memory device in the system. The typical SMC_B2CTL.PGSZ selection for external devices supporting async flash or async flash page protocols is 4 or 8 words. The typical SMC_B2CTL.PGSZ selection for external devices supporting sync burst flash protocol is 16 words.	
		0	4 words
		1	8 words
		2	16 words
		3	16 words
14 (R/W)	RDYABTEN	ARDY Abort Enable. The SMC_B2CTL.RDYABTEN bit enables the abort counter for the SMC_ARDY pin, if enabled (SMC_B2CTL.RDYEN =1). After SMC_B2TIM.RAT or SMC_B2TIM.WAT cycles, the SMC starts sampling the SMC_ARDY pin and starts the abort down counter (if enabled). The abort count is 64 cycles of SCLK. If the SMC detects that SMC_ARDY remains de-asserted when the counter expires, the SMC aborts the access and returns an error response back on the system bus.	
		0	Disable abort counter
		1	Enable abort counter

Table 9-8: SMC_B2CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	RDYPOL	ARDY Polarity. The SMC_B2CTL.RDYPOL bit selects the polarity (active high or low) for the SMC_ARDY pin, if enabled (SMC_B2CTL.RDYEN =1). When the SMC samples the SMC_ARDY pin in the selective active state, the transaction completes.	
		0	Low active ARDY
		1	High active ARDY
12 (R/W)	RDYEN	ARDY Enable. The SMC_B2CTL.RDYEN bit enables SMC_ARDY pin operation for bank 2 accesses. When enabled, the SMC uses SMC_ARDY (after the access time countdown) to determine completion of access to this memory bank. When disabled, the SMC ignores SMC_ARDY for accesses to this memory bank.	
		0	Disable ARDY
		1	Enable ARDY
9:8 (R/W)	SELCTRL	Select Control. The SMC_B2CTL.SELCTRL bits select the handling of the $\overline{\text{SMC\_AMS[n]}}$, $\overline{\text{SMC\_ARE}}$, $\overline{\text{SMC\_AOE}}$, and $\overline{\text{SMC\_AWE}}$ pins for memory access control.	
		0	AMS2 only
		1	AMS2 ORed with ARE
		2	AMS2 ORed with AOE
		3	AMS2 ORed with AWE
5:4 (R/W)	MODE	Memory Access Mode. The SMC_B2CTL.MODE bits select the protocol the SMC uses for static memory read/write access. Note that the write protocol for async flash, async flash page, and sync burst flash are all similar; only the read protocols differ for these modes.	
		0	Async SRAM protocol
		1	Async flash protocol
		2	Async flash page protocol
		3	Reserved
0 (R/W)	EN	Bank 2 Enable. The SMC_B2CTL.EN bit enables accesses to the memory in bank 2. When this bit is disabled, accesses to bank 2 return an error response.	
		0	Disable access
		1	Enable access

Bank 2 Extended Timing Register

The SMC_B2ETIM register configures extensions to access times and idle times, augmenting the setup, hold, and access times configured with the SMC_B2TIM register.

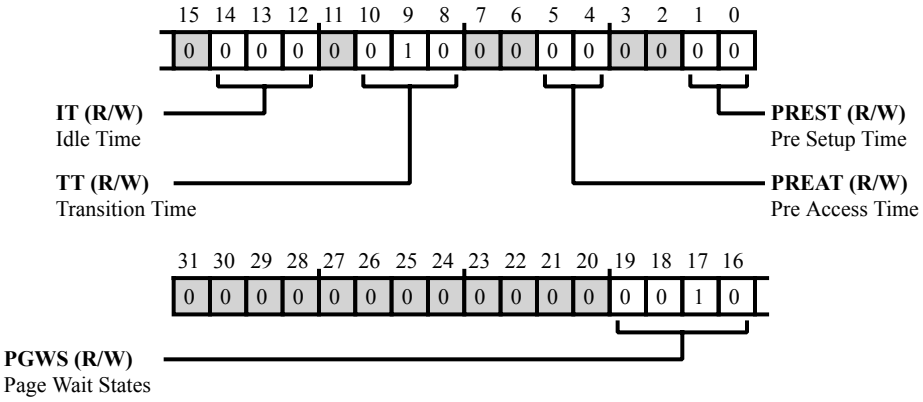

Figure 9-20: SMC_B2ETIM Register Diagram

Table 9-9: SMC_B2ETIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16 (R/W)	PGWS	Page Wait States. The SMC_B2ETIM.PGWS bits select a page access extension time (in SCLK cycles) that the SMC waits during read accesses when configured for flash page protocol (SMC_B2CTL.MODE =2). The wait time is from 2 to 15 SCLK cycles.	
		0	Not supported
		1	Not supported
		2-15	2-15 SCLK clock cycles
14:12 (R/W)	IT	Idle Time. The SMC_B2ETIM.IT bits select a bus idle time (in SCLK cycles) that the SMC waits between de-asserting the SMC_AMS[n] pin and asserting the SMC_AMS[n] pin for the next access. Note that the SMC_B2ETIM.IT period may be extended using the SMC_B2ETIM.TT selection. The idle time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		7	7 SCLK clock cycles

Table 9-9: SMC_B2ETIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10:8 (R/W)	TT	Transition Time. The SMC_B2ETIM.TT bits select a bus idle time (in SCLK cycles) that the SMC extends the SMC_B2ETIM.IT to allow for the subsequent access either using a different transfer direction or accessing a different bank. The transition time is from 1 to 7 SCLK cycles.	
		0	No bank transition
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The SMC_B2ETIM.PREAT bits select the pre-access time (in SCLK cycles) that the SMC waits after de-asserting the SMC_AOE/ADV pin before asserting the SMC_ARE/SMC_AWE pin for the current access. The pre-access time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles
1:0 (R/W)	PREST	Pre Setup Time. The SMC_B2ETIM.PREST bits select the pre-setup time (in SCLK cycles) that the SMC asserts the SMC_AMS[n] pin before asserting the SMC_AOE/ADV pin for an access. The pre-setup time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles

Bank 2 Timing Register

The SMC_B2TIM register configures bank 2 read and write access, setup, and hold timing for this bank. Note that read and write timing configurations are independent and may differ.

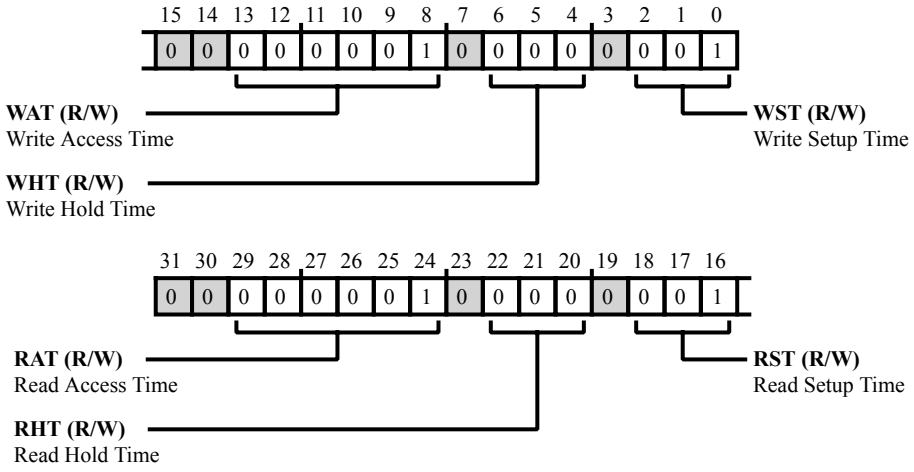

Figure 9-21: SMC_B2TIM Register Diagram

Table 9-10: SMC_B2TIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29:24 (R/W)	RAT	Read Access Time. The SMC_B2TIM.RAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_ARE pin for a read access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
22:20 (R/W)	RHT	Read Hold Time. The SMC_B2TIM.RHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the SMC_ARE pin before asserting the SMC_AOE pin for the next access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

Table 9-10: SMC_B2TIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	RST	Read Setup Time. The SMC_B2TIM.RST bits select the setup time (in SCLK cycles) that the SMC asserts the SMC_AOE pin before asserting the SMC_ARE pin for an access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
13:8 (R/W)	WAT	Write Access Time. The SMC_B2TIM.WAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_AWE pin for a write access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
6:4 (R/W)	WHT	Write Hold Time. The SMC_B2TIM.WHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the SMC_AWE pin before de-asserting the SMC_AOE pin for the current access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
2:0 (R/W)	WST	Write Setup Time. The SMC_B2TIM.WST bits select the setup time (in SCLK cycles) that the SMC asserts the SMC_AOE pin before asserting the SMC_AWE pin for a write access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

Bank 3 Control Register

The SMC_B3CTL register enables bank 3 accesses and configures the memory access features for this bank.

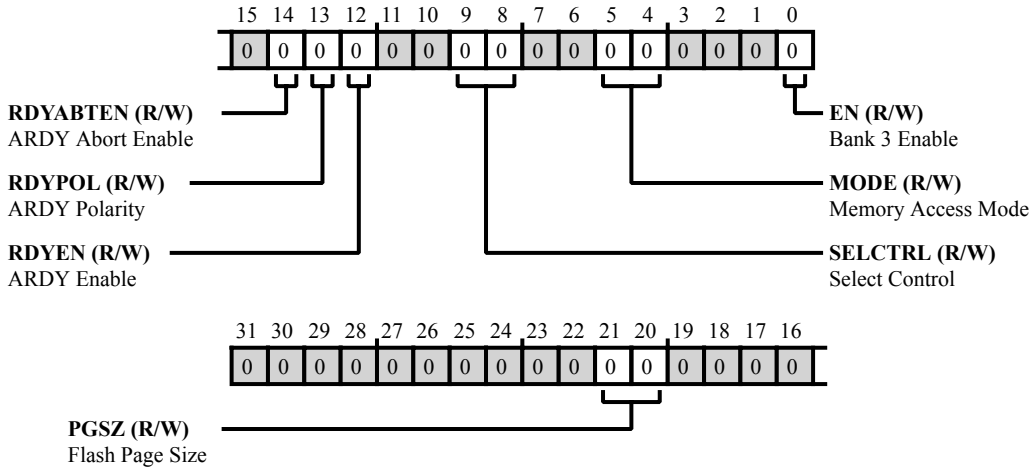

Figure 9-22: SMC_B3CTL Register Diagram

Table 9-11: SMC_B3CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/W)	PGSZ	Flash Page Size. The SMC_B3CTL.PGSZ bits select the flash page size, if page flash or sync burst flash protocol has been enabled (SMC_B3CTL.MODE > 1). Note that the SMC_B3CTL.PGSZ bits must be set to match the flash protocol of the external flash memory device in the system. The typical SMC_B3CTL.PGSZ selection for external devices supporting async flash or async flash page protocols is 4 or 8 words. The typical SMC_B3CTL.PGSZ selection for external devices supporting sync burst flash protocol is 16 words.	
		0	4 words
		1	8 words
		2	16 words
		3	16 words
14 (R/W)	RDYABTEN	ARDY Abort Enable. The SMC_B3CTL.RDYABTEN bit enables the abort counter for the SMC_ARDY pin, if enabled (SMC_B3CTL.RDYEN =1). After SMC_B3TIM.RAT or SMC_B3TIM.WAT cycles, the SMC starts sampling the SMC_ARDY pin and starts the abort down counter (if enabled). The abort count is 64 cycles of SCLK. If the SMC detects that SMC_ARDY remains de-asserted when the counter expires, the SMC aborts the access and returns an error response back on the system bus.	
		0	Disable abort counter
		1	Enable abort counter

Table 9-11: SMC_B3CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	RDYPOL	ARDY Polarity. The SMC_B3CTL.RDYPOL bit selects the polarity (active high or low) for the SMC_ARDY pin, if enabled (SMC_B3CTL.RDYEN =1). When the SMC samples the SMC_ARDY pin in the selective active state, the transaction completes.	
		0	Low active ARDY
		1	High active ARDY
12 (R/W)	RDYEN	ARDY Enable. The SMC_B3CTL.RDYEN bit enables SMC_ARDY pin operation for bank 3 accesses. When enabled, the SMC uses SMC_ARDY (after the access time countdown) to determine completion of access to this memory bank. When disabled, the SMC ignores SMC_ARDY for accesses to this memory bank.	
		0	Disable ARDY
		1	Enable ARDY
9:8 (R/W)	SELCTRL	Select Control. The SMC_B3CTL.SELCTRL bits select the handling of the $\overline{\text{SMC\_AMS[n]}}$, $\overline{\text{SMC\_ARE}}$, $\overline{\text{SMC\_AOE}}$, and $\overline{\text{SMC\_AWE}}$ pins for memory access control.	
		0	AMS3 only
		1	AMS3 ORed with ARE
		2	AMS3 ORed with AOE
		3	AMS3 ORed with AWE
5:4 (R/W)	MODE	Memory Access Mode. The SMC_B3CTL.MODE bits select the protocol the SMC uses for static memory read/write access. Note that the write protocol for async flash, async flash page, and sync burst flash are all similar; only the read protocols differ for these modes.	
		0	Async SRAM protocol
		1	Async flash protocol
		2	Async flash page protocol
		3	Reserved
0 (R/W)	EN	Bank 3 Enable. The SMC_B3CTL.EN bit enables accesses to the memory in bank 3. When this bit is disabled, accesses to bank 3 return an error response.	
		0	Disable access
		1	Enable access

Bank 3 Extended Timing Register

The SMC_B3ETIM register configures extensions to access times and idle times, augmenting the setup, hold, and access times configured with the SMC_B3TIM register.

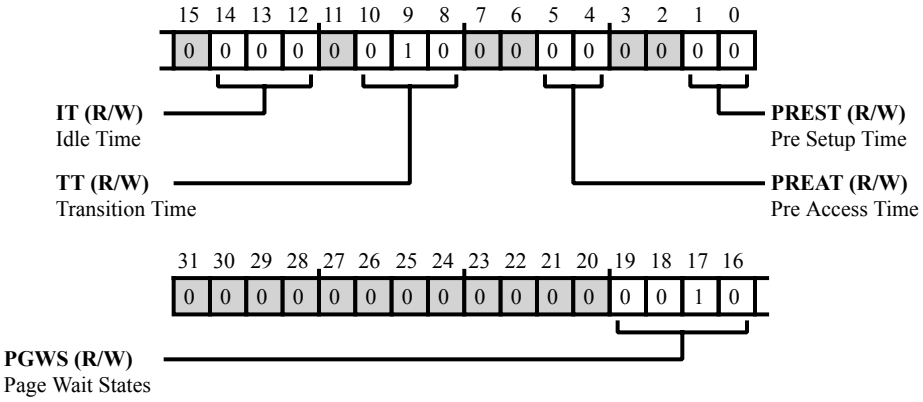

Figure 9-23: SMC_B3ETIM Register Diagram

Table 9-12: SMC_B3ETIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16 (R/W)	PGWS	Page Wait States. The SMC_B3ETIM.PGWS bits select a page access extension time (in SCLK cycles) that the SMC waits during read accesses when configured for flash page protocol (SMC_B3CTL.MODE =2). The wait time is from 2 to 15 SCLK cycles.	
		0	Not supported
		1	Not supported
		2-15	2-15 SCLK clock cycles
14:12 (R/W)	IT	Idle Time. The SMC_B3ETIM.IT bits select a bus idle time (in SCLK cycles) that the SMC waits between de-asserting the SMC_AMS[n] pin and asserting the SMC_AMS[n] pin for the next access. Note that the SMC_B3ETIM.IT period may be extended using the SMC_B3ETIM.TT selection. The idle time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		7	7 SCLK clock cycles

Table 9-12: SMC_B3ETIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10:8 (R/W)	TT	Transition Time. The SMC_B3ETIM.TT bits select a bus idle time (in SCLK cycles) that the SMC extends the SMC_B3ETIM.IT to allow for the subsequent access either using a different transfer direction or accessing a different bank. The transition time is from 1 to 7 SCLK cycles.	
		0	No bank transition
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The SMC_B3ETIM.PREAT bits select the pre-access time (in SCLK cycles) that the SMC waits after de-asserting the SMC_AOE/ADV pin before asserting the SMC_ARE/SMC_AWE pin for the current access. The pre-access time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles
1:0 (R/W)	PREST	Pre Setup Time. The SMC_B3ETIM.PREST bits select the pre-setup time (in SCLK cycles) that the SMC asserts the SMC_AMS[n] pin before asserting the SMC_AOE/ADV pin for an access. The pre-setup time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles

Bank 3 Timing Register

The SMC_B3TIM register configures bank 3 read and write access, setup, and hold timing for this bank. Note that read and write timing configurations are independent and may differ.

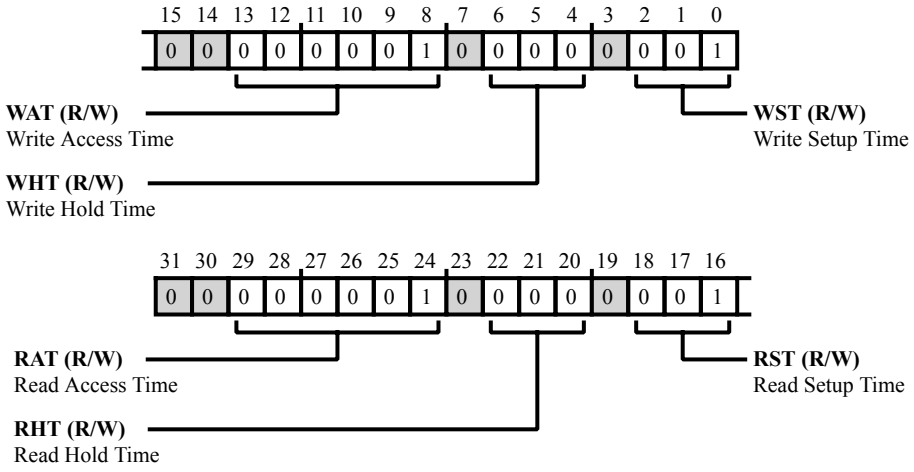

Figure 9-24: SMC_B3TIM Register Diagram

Table 9-13: SMC_B3TIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29:24 (R/W)	RAT	Read Access Time. The SMC_B3TIM.RAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_ARE pin for a read access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
22:20 (R/W)	RHT	Read Hold Time. The SMC_B3TIM.RHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the SMC_ARE pin before asserting the SMC_AOE pin for the next access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

Table 9-13: SMC_B3TIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	RST	Read Setup Time. The SMC_B3TIM.RST bits select the setup time (in SCLK cycles) that the SMC asserts the SMC_AOE pin before asserting the SMC_ARE pin for an access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
13:8 (R/W)	WAT	Write Access Time. The SMC_B3TIM.WAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_AWE pin for a write access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
6:4 (R/W)	WHT	Write Hold Time. The SMC_B3TIM.WHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the SMC_AWE pin before de-asserting the SMC_AOE pin for the current access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
2:0 (R/W)	WST	Write Setup Time. The SMC_B3TIM.WST bits select the setup time (in SCLK cycles) that the SMC asserts the SMC_AOE pin before asserting the SMC_AWE pin for a write access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

10 SPI (Internal) Flash

The SPI Flash is a high-performance multiple input/output serial flash memory device featuring functionality for eXecute-In-Place code execution, advanced write protection mechanisms, and a high-speed SPI-compatible bus interface.

High-performance, dual and quad input/output instructions enable double or quadruple the transfer bandwidth.

Features

The SPI Flash includes the following features:

- SPI-compatible serial bus interface
- SPI clock polarity and phase modes 0 and 3
- Normal and Fast Read support
- Dual/Quad I/O instructions provide increased throughput for read operations
- Sequential and wrapped read support
- Command Skip Mode for improved read performance
- Configurable number of dummy cycles
- Array organization of 4KB sector size, 16 sectors per block
- Sector, block and bulk erase support
- Page program up to 256 bytes
- Sector/ Block Protect functionality
- 768 Byte OTP Memory
- Software Write Protect functionality

Functional Description

The SPI protocol only uses between 4 and 6 signals depending on the operating mode.

- chip select ($\overline{CS}$)
- clock (CLK)
- serial data
  - SI (IO0)
  - SO (IO1)
  - IO2
  - IO3

The standard single input/output SPI serial protocol is supported in which SI is used for input to the flash and SO for output operations. Dual Output and Dual Input/Output (Dual I/O) operations use SI and SO to allow for transfers of up to 2 bits per clock cycle. The Quad Output and Quad Input/Output (Quad I/O) operations use SI, SO, IO2 and IO3 to allow transfer of up to 4 bits per clock cycle.

The ability to execute code from the SPI flash is often referred to as eXecute-In-Place (XIP). The Quad I/O functionality and high SPI clock frequencies supported by the device help ensure latencies are kept to a minimum. In order to further improve read performance the flash supports a command skip mode in which the last read command issued is sustained over subsequent $\overline{CS}$ assertions. This eliminates the need to issue the read command improving code execution performance.

Definitions

To make the best use of the SPI Flash, it is useful to understand the following terms.

Block

A group of sectors. All blocks consist of 64 sectors. Blocks are an erasable and lockable unit.

Command

The 8-bit code indicating the operation that is to be performed. The command is almost always the first 8 bits transferred to the memory in any given $\overline{CS}$ assertion. Exclusions relate to a specific read modes of operations such as command skip mode.

Command Skip Mode

Refers to a memory mode of operation in which the flash is configured such that the read command is not required for subsequent read requests. This is also commonly referred to as XIP mode or Continuous Read mode.

Dual I/O DPI command

Dual I/O DPI is a type of read command which supports Dual Peripheral Interface (DPI) operations in which the commands are input over 2-bit bus. For Dual I/O DPI read command all transfers of command, address, dummy

cycles, mode bytes and read data take place on the 2-bit bus. This read command is not supported by the internal flash.

Dual Output SPI command

Dual Output SPI is a type of read command in which command, address and dummy cycles are all input on a single SPI line (SO) while the data read is output on the 2-bit bus.

Dual I/O SPI command

Dual I/O SPI is a type of read command in which command is input on a single SPI line (SO), while all other transfers of address, dummy cycles, mode bytes and read data take place on the 2-bit bus.

Dummy Cycles

Dummy cycles are used to implement latency into the operation for memory read operations. They generally occur after the address has been issued and prior to the data being returned.

Lock

Generally refers to the protection state of a sector or block. Sectors and blocks can be locked in order to prevent further erase and program operations.

LSB

Least Significant Bit. Generally the right most bit with the lowest order of magnitude value within a group of bits.

MSB

Most significant Bit. Generally the left most bit with the highest order of magnitude value withing a group of bits.

Non-Volatile

No power is required to maintain data stored in the memory.

Page

Group of 256 bytes of data and 256-byte aligned.

Quad I/O QPI command

Quad I/O QPI is a type of read command which supports Quad Peripheral Interface (QPI) operations in which the commands are input over 4-bit bus. For Quad I/O QPI read command all transfers of command, address, dummy cycles, mode bytes and read data take place on the 4-bit bus. This read command is not supported by the internal flash.

Quad I/O SPI command

Quad I/O SPI is a type of read command in which command is input on a single SPI line (SO), while all other transfers of address, dummy cycles, mode bytes and read data take place on the 4-bit bus.

Quad Output SPI command

Quad Output SPI is a type of read command in which command, address and dummy cycles are all input on a single SPI line (SO) while the data read is output on the 4-bit bus.

Sector

Minimum erase unit size. All sectors are 4KB in length and 4KB aligned. The erase command used determines how many sectors may be erased. Individual sectors are also lockable.

XIP

Execute-in-place. Refers to the ability to execute code from memory. This terms is also sometimes used to describe an SPI feature that enables faster read performance from the memory. Please refer to command skip mode

Signal Descriptions

Describes the signals for the SPI Flash Memory Interface.

Table 10-1: Table 1: SPI Flash Signal Descriptions

Symbol	*Type*	*Description*
CLK	Input	***Clock:*** Provides the timing of the serial interface. Commands, addresses, or data present at serial data inputs are latched on the rising edge of the clock. Data is shifted out on the falling edge of the clock.
$\overline{CS}$	Input	***Chip select:*** When $\overline{CS}$ is high, the device is deselected and SO is at High-Z. When in extended SPI mode, with the device deselected, SO is three-stated. Unless an internal PROGRAM, ERASE, or WRITE STATUS REGISTER cycle is in progress, the device enters standby power mode (not deep power-down mode). Driving $\overline{CS}$ low enables the device, placing it in the active power mode. After power-up, a falling edge on $\overline{CS}$ is required prior to the start of any command.
SI (IO0)	I/O	***Serial Input for single bit data commands, IO0 for Dual and Quad commands.***
SO (IO1)	I/O	***Serial Output for single bit data commands, IO1 for Dual and Quad commands.***
IO2	I/O	***Quad data*** For Quad SPI read commands, IO2 functions as input/output signal for address and data transfers.
IO3	I/O	***Quad data:*** For Quad I/O and Quad output SPI read commands, IO3 functions as input/output signal for address and data transfers.

Architectural Concepts

To understand how to use the SPI flash, it is important to understand the memory organization, device protection features and SPI flash register definitions.

Memory Organization

The memory is configured as up to 2,097,152 bytes; consisting of up to 32 blocks (64 KB each); 512 sectors (4 KB each); and 8192 pages (256 bytes each).The memory is organized as 64 KB blocks that are further divided into 16 sectors each. Each sector comprises of 256 pages and each page is 256 bytes long. The memory can be erased one 4 KB sector at a time, one 64 KB block at a time, or the entire device can be erased. The device is sector, block, or bulk-erasable, but not page-erasable. Each page of memory can be individually programmed.

The memory can be write protected by software through volatile and non-volatile protection features, depending on the application needs.

The device has three regions of 256 bytes of memory with one-time programmable (OTP) protection that can be read, programmed and erased with the READ OTP ARRAY, PROGRAM OTP ARRAY and ERASE OTP ARRAY commands. The OTP array can be permanently locked by writing to the `STATUS2.OTPLOCK` bits.

The memory is configured as up to 2,097,152 bytes; consisting of up to 32 blocks (64 KB each); 512 sub sectors (4KB each); and 8192 pages (256 bytes each). The OTP arrays are located outside the main memory array.

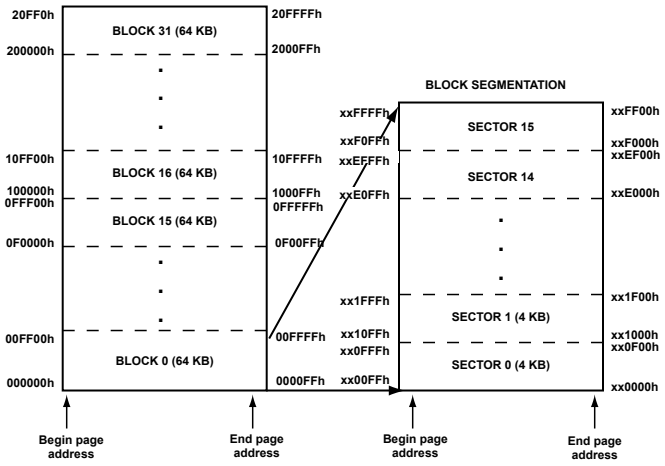

Figure 10-1: Flash Memory Organization

Flash Device Protection

The flash device and protocol provide a number of key features to help protect the contents of flash memory. These include the inability to erase a block with a single command and the ability to lock regions of the memory protecting them from erase and write operations.

Table 10-2: Data Protection Using Device Protocols

Protection by:	*Description*
Command execution check	Ensures that the number of clock pulses is a multiple of one byte before executing a PROGRAM or ERASE command, or any command that writes to the device registers.
WRITE ENABLE operation	Ensures that commands modifying device data must be preceded by a WRITE ENABLE command, which sets the write enable latch bit in the status register.

The Block Protect bits (`STATUS1.BP[4:2]`) provide Write Protection control and status. All, none or a portion of the memory array can be protected from Program and Erase commands. The default setting for the Block Protection bits is 0, where none of the array is protected.

The Top/Bottom bit (`STATUS1.TB`) controls whether the Block Protect bits (`STATUS1.BP[4:2]`) protect from the top (`STATUS1.TB` =0) or the bottom (`STATUS1.TB` =1) of the array.

The Sector/Block Protect bit (`STATUS1.SEC`) controls if the Block Protect bits (`STATUS1.BP[4:2]`) protect either 4-kB Sectors (`STATUS1.SEC`=1) or 64-kB blocks (`STATUS1.SEC`=0) in the top (`STATUS1.TB` =0) or the bottom (`STATUS1.BP` =1) of the array as shown in the ***Block Protection*** table and the ***Complement Block Protection*** table.

If the Complement Protect bit (`STATUS2.CMP`) =1, the previous array protection set by the `STATUS1.SEC`, `STATUS1.TB`, `STATUS1.BP[4:2]` bits is reversed. For instance, when `STATUS2.CMP` =0, a top 4-kB sector can be protected while the rest of the array is not. When `STATUS2.CMP` =1, the top 4-kB sector is unprotected while the rest of the array becomes read-only.

Table 10-3: Block Protection

STATUS1 Register Bit Field			**Block Protection (`STATUS2.CMP`=0)**		
SEC	TB	BP	Protected Blocks	Protected Addresses	Protected Density
X	X	000	None	None	None
0	0	001	31	0x1F0000- 0x1FFFFF	64 kB
0	0	010	30 and 31	0x1E0000- 0x1FFFFF	128 kB
0	0	011	28 through 31	0x1C0000- 0x1FFFFF	256 kB
0	0	100	24 through 31	0x180000- 0x1FFFFF	512 kB
0	0	101	16 through 31	0x100000- 0x1FFFFF	1 MB
0	1	001	0	0x000000- 0x00FFFF	64 kB
0	1	010	0 and 1	0x000000- 0x01FFFF	128 kB
0	1	011	0 through 3	0x000000- 0x03FFFF	256 kB
0	1	100	0 through 7	0x000000- 0x07FFFF	512 kB
0	1	101	0 through 15	0x000000- 0x0FFFFF	1 MB
X	X	11X	0 through 31	0x000000- 0x1FFFFF	2 MB

Table 10-3: Block Protection (Continued)

STATUS1 Register Bit Field			Block Protection (STATUS2.CMP=0)		
1	0	001	Partial 31	0x1FF000- 0x1FFFFF	4 kB
1	0	010	Partial 31	0x1FE000- 0x1FFFFF	8 kB
1	0	011	Partial 31	0x1FC000- 0x1FFFFF	16 kB
1	0	10X	Partial 31	0x1F8000- 0x1FFFFF	32 kB
1	1	001	Partial 0	0x000000- 0x000FFF	4 kB
1	1	010	Partial 0	0x000000- 0x001FFF	8 kB
1	1	011	Partial 0	0x000000- 0x003FFF	16 kB
1	1	10X	Partial 0	0x000000- 0x007FFF	32 kB

Table 10-4: Complement Block Protection

STATUS1 Register Bit Field			Block Protection (STATUS2.CMP=1)		
SEC	TB	BP	Protected Blocks	Protected Addresses	Protected Density
X	X	000	0 through 31	0x000000 - 0x1FFFFF	All
0	0	001	0 through 30	0x000000 - 0x1EFFFF	1,984 kB
0	0	010	0 through 29	0x000000- 0x1DFFFF	1,920 kB
0	0	011	0 through 27	0x000000- 0x1BFFFF	1,792 kB
0	0	100	0 through 23	0x000000- 0x17FFFF	1,536 kB
0	0	101	0 through 15	0x000000- 0x0FFFFF	1 MB
0	1	001	1 though 31	0x010000- 0x1FFFFF	1,984 kB
0	1	010	2 through 31	0x020000- 0x1FFFFF	1,920 kB
0	1	011	4 through 31	0x040000- 0x1FFFFF	1,792 kB
0	1	100	8 through 31	0x080000- 0x1FFFFF	1,536 kB
0	1	101	16 through 31	0x100000- 0x1FFFFF	1 MB
X	X	11X	None	None	None
1	0	001	0 through 31	0x000000- 0x1FEFFF	2,044 kB
1	0	010	0 through 31	0x000000- 0x1FDFFF	2, 040 kB
1	0	011	0 through 31	0x000000- 0x1FBFFF	2, 032 kB
1	0	10X	0 through 31	0x000000- 0x1F7FFF	2,016 kB
1	1	001	0 through 31	0x001000- 0x1FFFFF	2,044 kB
1	1	010	0 through 31	0x002000- 0x1FFFFF	2,040 kB
1	1	011	0 through 31	0x004000- 0x1FFFFF	2,032 kB

Table 10-4: Complement Block Protection (Continued)

STATUS1 Register Bit Field			Block Protection (`STATUS2.CMP`=1)		
1	1	10X	0 through 31	0x008000- 0x1FFFFF	2,016 kB

Operating Modes

The SPI flash memory supports many different operating modes and protocols. The modes and protocols can be used to configure the state of signals when no operations are taking place as well as during transfers, the bus width to be used at various stages in the transfer.

Interface Modes

The flash can be driven via the serial peripheral interface in one of two modes commonly referred to as Mode 0 and Mode 3.The difference between the two modes is the clock polarity when the bus master is in standby mode and not transferring data.

In both these interface modes, input data is latched in on the rising edge of the clock, and output data is available from the falling edge of the clock.

Table 10-5: SPI Modes

SPI Modes	*Clock Polarity*
CPOL = 0, CPHA = 0	Clock remains at 0 for (CPOL = 0, CPHA = 0)
CPOL = 1, CPHA = 1	Clock remains at 1 for (CPOL = 1, CPHA = 1)

Operating Protocols

SPI flash devices can support multiple operation protocols that can define the bus width used to issue a command to the device.

The memory can be operated with the following protocols:

1. ***Standard SPI Commands.*** Standard SPI commands use the SI input pin to serially write instructions, addresses or data to the device on the rising edge of CLK. The SO output pin is used to read data or status from the device on the falling edge CLK.
2. ***Dual SPI Commands.*** The SoC flash supports Dual SPI operation when using the Fast Read Dual Output and Fast Read Dual I/O commands. For Fast Read Dual Output read command, the command, address and dummy cycle inputs take place on the SI pin and the data from read operations is output on the SI and SO pins. For Fast Read Dual I/O read command, the command input takes place on the SI pin. The address, dummy cycles and mode bytes take place over the SI and SO pins and data is returned over the SI and SO pins.
3. ***Quad SPI Commands.*** The SoC flash supports Quad SPI operation when using the Fast Read Quad Output and Fast Read Quad I/O commands. Quad SPI commands require the nonvolatile or volatile Quad Enable bit in STATUS2 register to be set. For Fast Read Quad Output operation all the command, address and dummy cycle inputs take place on the SI pin and the data from read operations is output on the SI, SO, IO2 and IO3

pins. For Fast Read Quad I/O operation, the command input takes place on the SI pin. The address, dummy cycles and mode bytes take place over the SI, SO, IO2 and IO3 signals and data is returned over the SI,SO, IO2 and IO3 signals.

The following is the description of various features of flash.

- Command skip mode. This mode of operation allows improved read performance by eliminating the need to issue a read command for subsequent read operations.
- Configurable dummy cycle count for fast read operations allows to optimize the protocol for the required frequency of operation.

NOTE: Command skip mode will result in the memory not responding to any command sequences. Care is required to ensure the flash device is brought out of this mode of operation if anything other than a read operation is required.

- Each SPI operation begins with the assertion of the chip select signal $\overline{CS}$ low and ends with this signal going high.
- Operations begin with an 8-bit command sequence on the single bit `SI` signal with each bit transfered on each rising edge of the SPI clock. The command determines the operation to be performed.
- The command issued determines the bus with to be used for all transfers following the command.
- Some commands can be issued stand alone and others are required to be followed by a 24-bit address in order to select a memory location within the flash.
- Some commands support a mode configuration that follows the address sequence. This is most often used to notify the flash that the next command sequence to follow after the de-assertion of the $\overline{CS}$ signal should use the same command as the currently executing sequence. This allows for the entering of the command skip mode of operation.
- For a read operation dummy cycles may be required after the address and mode configuration before the data is returned from the flash.
    - Following the dummy cycles, data is returned from the flash on every SPI clock cycle until the de-assertion of the $\overline{CS}$ signal. $\overline{CS}$ de-assertion terminates the read operation.
- Write operations require the data to be written to follow the address cycles, no dummy cycles are required.
- For commands sequences that do not return data, the $\overline{CS}$ signal must only be driven high at the 8 bit boundary in order for the sequence to be accepted. If $\overline{CS}$ de-assertion occurs early or if additional clock cycles are detected prior to the $\overline{CS}$ de-assertion the operation will be rejected.
- All commands, addresses and mode transfers occur MSB first. Read and write data sequences also occur MSB first. All data is transferred with the lowest address byte transferred first.
- Flash array read operations during program or erase operations are ignored.

- A status read command is available to allow for verification on whether an operation is still ongoing or whether it completed successfully or not.

Command Skip Mode

Command skip mode allows for more efficient read accesses by eliminating the requirement to issue the read command for data read operations. This is especially beneficial when executing code from the SPI flash memory.

Command skip mode is controlled by the mode configuration field that follows the address cycles on the FAST READ DUAL I/O and FAST READ QUAD I/O operations.

The mode configuration field is the first byte to be transmitted after the address phase and is transmitted MSB first.

If the Continuous Read Mode bits M5-4 = (1,0), then the next Fast Read Dual I/O or Fast Read Quad I/O command after CS is lowered does not require the read command code. This reduces the command sequence by eight clocks and allows the Read address to be immediately entered after CS is asserted low. If the Continuous Read Mode bits M5-4 ≠ (1,0), the next command after CS is lowered requires the first byte command code, returning to normal operation.

NOTE: It is important to note that command skip mode of operation can result in loss of communication between the SPI controller and the memory device in the event of a system reset event. The SPI flash memory does not contain a reset pin and therefore a software procedure is required in order to ensure the SPI memory is brought back into the mode of operation in which the SPI controller can then re-establish communication and issue command cycles to the memory. Please refer to the Power Loss Recovery Sequence for further details.

Dummy Cycle Configuration

The number of dummy cycles used during FAST READ commands is configurable to allow for the optimization of the protocol for the required frequency of operation.

The number of dummy cycles is configurable via `STATUS3.DCYCLES`. The table below shows the supported configurations that can be used with each of the supported FAST READ commands. The Dummy cycle bits may be set to select a number of read cycles optimized for the frequency in use. If the number of dummy cycles is not sufficient for the operating frequency, invalid data is read. The table below shows the supported configurations that can be used with each of the supported FAST READ commands for different frequencies.

Table 10-6: Dummy Cycles for Fast Read Commands

`STAT3.DCYCLES`	Read Command Maximum Frequency (MHz)				
	FAST READ	DUAL OUTPUT FAST READ	DUAL I/O FAST READ	QUAD OUTPUT FAST READ	QUAD I/O FAST READ
0 (legacy dummy cycle)	100 (8 dummy cycles)	100 (8 dummy cycles)	88 (4 mode and 4 dummy cycles)	100 (8 dummy cycles)	78 (2 mode and 6 dummy cycles)

Table 10-6: Dummy Cycles for Fast Read Commands (Continued)

STAT3.DCYCLES	Read Command Maximum Frequency (MHz)				
	FAST READ	DUAL OUTPUT FAST READ	DUAL I/O FAST READ	QUAD OUTPUT FAST READ	QUAD I/O FAST READ
1					
2					59 (2 mode and 2 dummy cycles)
3					
4			100 (4 mode and 4 dummy cycles)		78 (2 mode and 4 dummy cycles)
5					
6					95 (2 mode and 6 dummy cycles)
7					
8	100 (8 dummy cycles)	100 (8 dummy cycles)	100 (4 mode and 8 dummy cycles)	100 (8 dummy cycles)	100 (2 mode and 8 dummy cycles)
9					
10					100 (2 mode and 10 dummy cycles)
11					
12			100 (4 mode and 12 dummy cycles)		100 (2 mode and 12 dummy cycles)
13					
14					100 (2 mode and 14 dummy cycles)
15					

NOTE: The DUAL I/O and the QUAD I/O FAST READ commands require a mode configuration byte after the address cycles. The `STAT3.DCYCLES` value that applies to the DUAL I/O and the QUAD I/O FAST READ commands in the above table does not take this mode control byte into account. Therefore if `DCYCLES` is configured for 8 dummy cycles and the operation is a QUAD I/O FAST READ, 2 cycles are

required to send the mode byte then 8 dummy cycles are required before the data. Thus 10 cycles are required between the end of the address and the beginning of the data.

Programming Model

The programming model for the SPI Flash includes:

- Command Definitions
- Identification Operations
- Read Operations
- Write Operations
- Register Operations
- Program Operations
- Erase Operations
- OTP Operations
- Power Loss Recovery Sequence

Command Definitions

Table 10-7: Command Set

Command	Code	Data Bytes
Identification Operations		
READ ID	9Fh	3
Read Operations		
READ	03h	1 to ∞
FAST READ	0Bh	1 to ∞
DUAL OUTPUT FAST READ	3Bh	1 to ∞
DUAL INPUT/OUTPUT FAST READ	BBh	1 to ∞
QUAD OUTPUT FAST READ	6Bh	1 to ∞
QUAD INPUT/OUTPUT FAST READ	EBh	1 to ∞
SET BURST WITH WRAP	77h	1 to ∞
Write Operations		
WRITE ENABLE	06h	0
WRITE ENABLE FOR VOLATILE STATUS REGISTER	50h	0

Table 10-7: Command Set (Continued)

Command	Code	Data Bytes
WRITE DISABLE	04h	0
Register Operations		
READ STATUS1 REGISTER	05h	1 to ∞
WRITE STATUS REGISTER	01h	
Program Operations		
PAGE PROGRAM	02h	1 to 256
Erase Operations		
SECTOR ERASE	20h	0
BLOCK ERASE	D8h	0
BULK ERASE	C7h	0
OTP Operations		
READ OTP ARRAY		
PROGRAM OTP ARRAY	42h	
Continuous Mode Reset Operation		
CONTINUOUS MODE RESET	0xFFh or 0xFFFFh	0

Identification Operations

The following sections describe the Flash ID operations

READ ID

The READ ID command allows for the reading of the JEDEC industry standard ID.

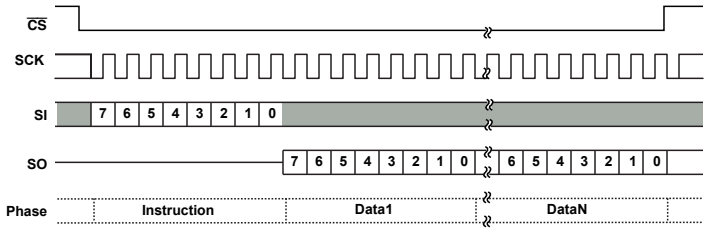

Figure 10-2: Read JEDEC ID Command Sequence

To execute the command, $\overline{CS}$ is driven low and the command code is input on `SI`. The device outputs the information shown below. If an erase or program operation is in progress when the command is executed, the command is not decoded and the command cycle in progress is not affected. When $\overline{CS}$ is driven high, the device goes to standby. The operation is terminated by driving $\overline{CS}$ high at any time during data output.

Table 10-8: Read ID Data Out

Size (Bytes)	Name	Content Value	
3	Device ID	0x154001	

Read Operations

The following sections describe the Flash read operations.

READ

The READ command is used to read the contents of the flash array using the single bit bus.

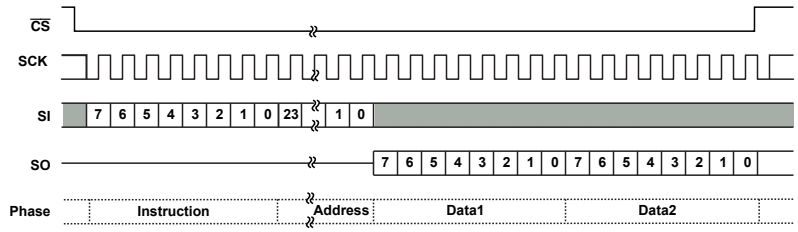

Figure 10-3: Read Data Command

To execute the command, $\overline{CS}$ is driven low. The command code is input on SI, followed by the input on SI of three address bytes. Each address bit is latched in during the rising edge of the clock. After the address has been received, data is shifted out on the SO signal on the falling edge of the clock MSB first. The operation is terminated by driving $\overline{CS}$ high at any time during data output.

The addressed byte can be at any location, and the address automatically increments to the next address after each byte of data is shifted out allowing for the entire memory array to be read with a single command.

FAST READ

The FAST READ command is used to read the contents of the flash array using the single bit bus. It allows for faster operating frequency due to the use of dummy cycles.

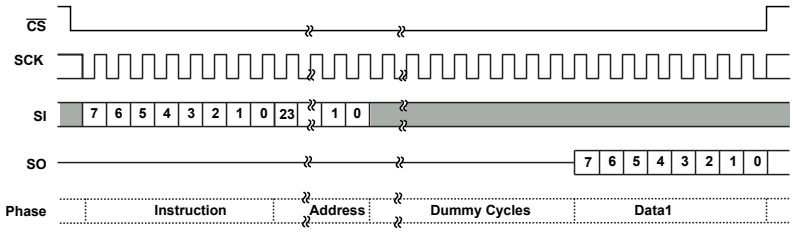

Figure 10-4: Fast Read Command Sequence

To execute the command, $\overline{CS}$ is driven low. The command code is input on SI, followed by the input on SI of three address bytes and a single dummy byte (8 dummy cycles). Each address bit and dummy bit is latched in during the rising edge of the clock. After the required number of dummy cycles have been received, data is shifted out

on the SO signal on the falling edge of the clock MSB first. The operation is terminated by driving $\overline{CS}$ high at any time during data output.

The addressed byte can be at any location, and the address automatically increments to the next address after each byte of data is shifted out allowing for the entire memory array to be read with a single command.

NOTE: If variable dummy cycles is enabled, the number of dummy cycles required is determined by the SR3.DUMMY_CYCLES parameter.

DUAL OUTPUT FAST READ

The DUAL OUTPUT FAST READ command is used to read the contents of the flash array using the dual bit bus for the data cycles. It allows for faster operating frequency due to the use of dummy cycles.

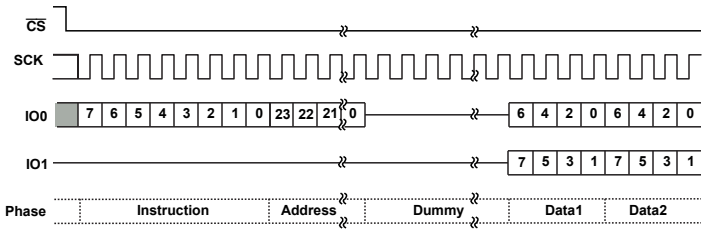

Figure 10-5: Dual Output Fast Read Command Sequence

To execute the command, $\overline{CS}$ is driven low. The command code is input on SI, followed by the input on SI of three address bytes and a single dummy byte (8 dummy cycles). Each address bit and dummy bit is latched in during the rising edge of the clock. After the required number of dummy cycles have been received, data is shifted out on the SO and SI signals on the falling edge of the clock MSB first. The operation is terminated by driving $\overline{CS}$ high at any time during data output.

The addressed byte can be at any location, and the address automatically increments to the next address after each byte of data is shifted out allowing for the entire memory array to be read with a single command.

NOTE: If variable dummy cycles is enabled, the number of dummy cycles required is determined by the .DUMMY_CYCLES parameter.

DUAL I/O FAST READ

The DUAL I/O FAST READ command is used to read the contents of the flash array using the dual bit bus for the address dummy and data cycles. It allows for faster operating frequency due to the use of dummy cycles.

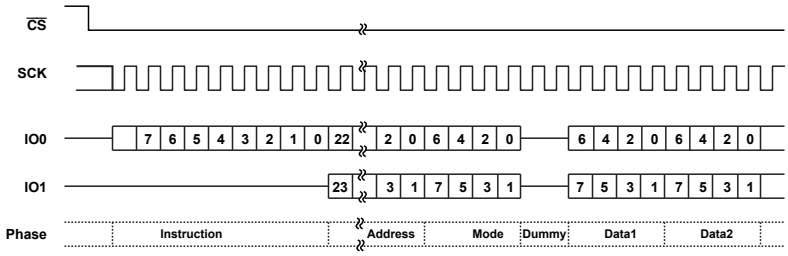

Figure 10-6: Fast Read Dual I/O Command Sequence (Initial command or previous M5-4 ≠ 10)

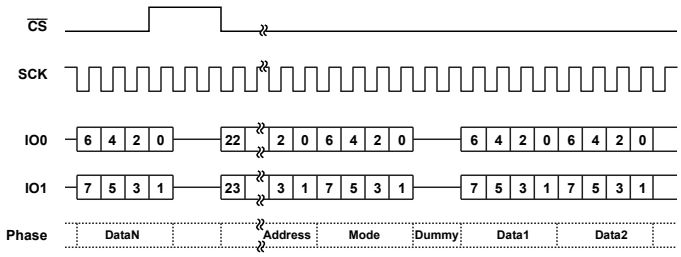

Figure 10-7: Fast Read Dual I/O Command Sequence (Previous command set M5-4 = 10)

To execute the command, $\overline{CS}$ is driven low. The command code is input on SI, followed by the input of three address bytes and a single dummy byte (4 dummy cycles). The address and dummy bytes are input on the SO and SI signals. Each address bit and dummy bit is latched in during the rising edge of the clock. After the required number of dummy cycles have been received, data is shifted out on the SO and SI signals on the falling edge of the clock MSB first. The operation is terminated by driving $\overline{CS}$ high at any time during data output.

The addressed byte can be at any location, and the address automatically increments to the next address after each byte of data is shifted out allowing for the entire memory array to be read with a single command. To set command skip mode, the mode bytes are configured to be M5-4 = (1,0), then the next Dual I/O fast read command does not require command code to be sent.

NOTE: If variable dummy cycles is enabled, the number of dummy cycles required is determined by the .DCYCLES parameter.

QUAD OUTPUT FAST READ

The QUAD OUTPUT FAST READ command is used to read the contents of the flash array using the quad bit bus for the data cycles. It allows for faster operating frequency due to the use of dummy cycles.

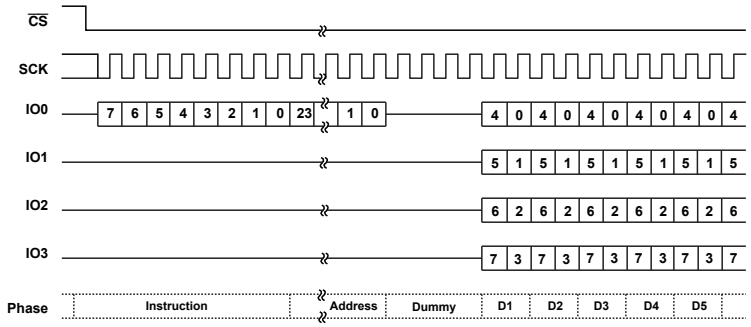

Figure 10-8: Fast Read Quad Output Command Sequence

To execute the command, $\overline{CS}$ is driven low. The command code is input on SI, followed by the input on SI of three address bytes and a single dummy byte (8 dummy cycles). Each address bit and dummy bit is latched in during the rising edge of the clock. After the required number of dummy cycles have been received, data is shifted out on the IO3,IO2,SO and SI signals on the falling edge of the clock MSB first. The operation is terminated by driving $\overline{CS}$ high at any time during data output.

The addressed byte can be at any location, and the address automatically increments to the next address after each byte of data is shifted out allowing for the entire memory array to be read with a single command.

NOTE: If variable dummy cycles is enabled, the number of dummy cycles required is determined by the .DUMMY_CYCLES parameter.

QUAD I/O FAST READ

The QUAD I/O FAST READ command is used to read the contents of the flash array using the quad bit bus for the address, dummy and data cycles. It allows for faster operating frequency due to the use of dummy cycles.

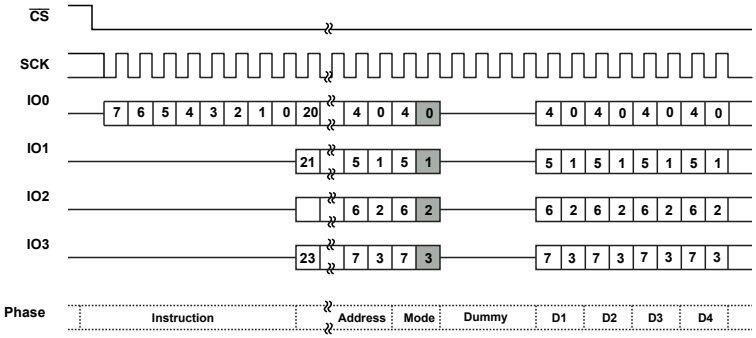

Figure 10-9: Fast Read Quad I/O Command Sequence (Initial command or previous M5-4 ≠ 10)

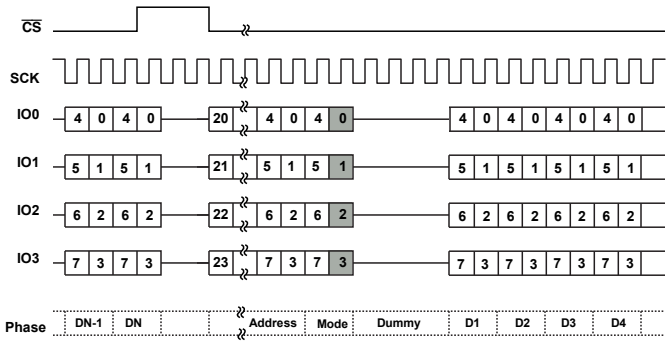

Figure 10-10: Fast Read Quad I/O Command Sequence (Previous command set M5-4 = 10)

To execute the command, $\overline{CS}$ is driven low. The command code is input on SI, followed by the input of three address bytes and 3 dummy bytes (6 dummy cycles). The address and dummy bytes are input on the IO3, IO2,SO and SI signals.Each address bit and dummy bit is latched in during the rising edge of the clock. After the required number of dummy cycles have been received, data is shifted out on the SO and SI signals on the falling edge of the clock MSB first. The operation is terminated by driving $\overline{CS}$ high at any time during data output.

The addressed byte can be at any location, and the address automatically increments to the next address after each byte of data is shifted out allowing for the entire memory array to be read with a single command. To set command skip mode, the mode bytes are configured to be M5-4 = (1,0), then the next Quad I/O fast read command does not require command code to be sent.

NOTE: If variable dummy cycles is enabled, the number of dummy cycles required is determined by the .DUMMY_CYCLES parameter.

SET BURST WITH WRAP

The Set Burst with Wrap command is used in conjunction with QUAD I/O FAST READ command to access a fixed length of 8/16/32 byte section within a 256-byte page. Depending upon application this feature can improve overall system code execution performance.

To execute the command, CS is driven low. The command code is input on SI, followed by 3 dummy bytes (6 dummy cycles) and a wrap byte. The dummy bytes and wrap byte are input on the IO3, IO2,SO and SI signals. Each dummy bit and wrap bit is latched in during the rising edge of the clock. Refer to figure for the command.

Wrap bit W4 is used to enable wrap and wrap bits W[6:5] is used to define wrap length. Following table depicts configuration for wrap bits.

Table 10-9: Wrap Byte Configuration

W6, W5	W4 = 0		W4 = 1 (Default)	
	Wrap Around	Wrap Length	Wrap Around	Wrap Length
00	Yes	8-Byte	No	N/A
01	Yes	16-Byte	No	N/A

Table 10-9: Wrap Byte Configuration (Continued)

W6, W5	W4 = 0		W4 = 1 (Default)	
	Wrap Around	Wrap Length	Wrap Around	Wrap Length
10	Yes	32-Byte	No	N/A

Once Set Burst with Wrap command is executed, all the following QUAD I/O FAST READ command will use wrap bits configuration to access the 8/16/32-byte section within any page. To exit the Wrap Around function and return to normal read operation, another Set Burst with Wrap command should be issued to set wrap bit, W4 = 1.

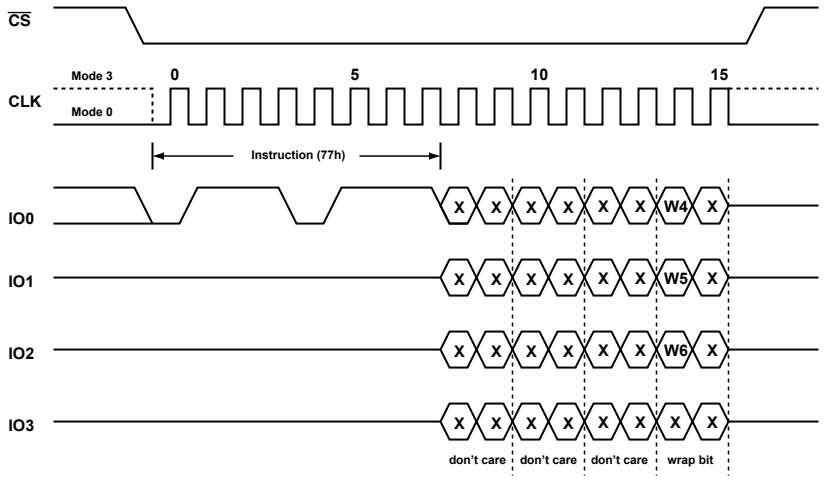

Figure 10-11: Set Burst with Wrap Instruction Sequence

Write Operations

The following sections describe the Flash write operations.

WRITE ENABLE

The WRITE ENABLE command is used to set the Write Enable Latch bit (`STATUS1.WEL`) that is required in order for program erase or register write operations to occur.

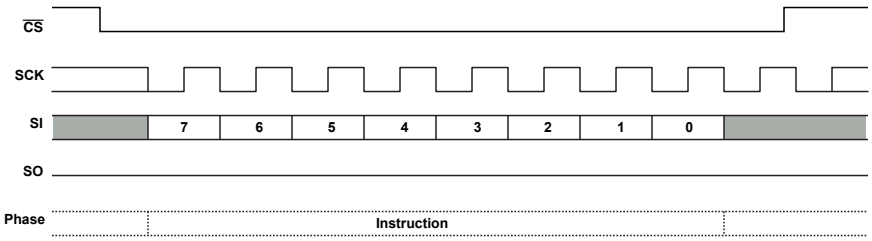

Figure 10-12: Write Enable Command Sequence

To execute the command, $\overline{CS}$ is driven low. The command code is input on `SI`. The operation is terminated by driving $\overline{CS}$ high after the eighth clock cycle.

NOTE: The STATUS1.WELbit is automatically cleared after completion of a program, erase or write operation.

WRITE ENABLE FOR VOLATILE STATUS REGISTER

The Status Register bits described in SPI Flash Register section can also be written to as volatile bits. During power up reset, the non-volatile Status Register bits are copied to a volatile version of the Status Register that is used during device operation. The value of volatile version of status register bits will be retained till it is powered up again. Write to Status Register as a volatile version will be faster as compared to non-volatile write.

To write the volatile version of the Status Register bits, the Write Enable for Volatile Status Register command must be issued and immediately followed by the Write Status Registers command. The Write Enable for Volatile Status Register command does not set the STATUS1.WEL bit, it is changed by the Write Status Registers command.

WRITE_DISABLE

The WRITE DISABLE command is used to reset the Write Enable Latch bit (STATUS1.WEL) to 0.

To execute the command, $\overline{CS}$ is driven low. The command code is input on SI. The operation is terminated by driving $\overline{CS}$ high after the eighth clock cycle.

NOTE: The STATUS1.WELbit is automatically cleared after completion of a program erase or write operation.

Register Operations

The following commands are used for flash status register operations.

READ STATUS1 REGISTER

The READ STATUS1 REGISTER command allows for the reading of the 8-bit STATUS1 register.

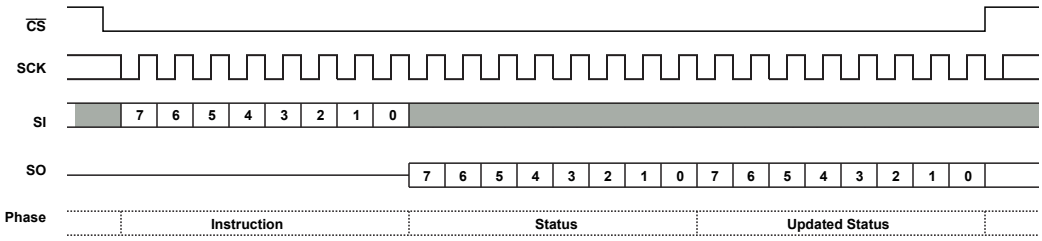

Figure 10-13: Read Status Register Command Sequence

To execute the command, $\overline{CS}$ is driven low. The command code is input on SI. After the command has been received, the contents of the status register are shifted out on the SO signal on the falling edge of the clock MSB first. The operation is terminated by driving $\overline{CS}$ high at any time during data output.

The command may be used at any time such as when program, erase or write operations are in progress. By continuing to hold $\overline{CS}$ low, the updated status register contents are output on the following data byte. This allows software to poll on the various status bits without needing to continually issue the command cycle as long as $\overline{CS}$ is asserted low for the duration of the polling period.

READ STATUS2 REGISTER

The READ STATUS2 REGISTER command allows for the reading of the 8-bit STATUS2 register.

To execute the command, $\overline{CS}$ is driven low. The command code is input on SI. After the command has been received, the contents of the status register are shifted out on the SO signal on the falling edge of the clock MSB first. The operation is terminated by driving $\overline{CS}$ high at any time during data output.

By continuing to hold $\overline{CS}$ low, the updated status register contents will be output on the following data byte. This allows software to poll on the various status bits without needing to continually issue the command cycle as long as $\overline{CS}$ is asserted low for the duration of the polling period.

NOTE: Unlike the READ STATUS1 REGISTER command. The command may not be used at any time such as when program, erase or write operations are in progress.

READ STATUS3 REGISTER

The READ STATUS3 REGISTER command allows for the reading of the 8-bit STATUS3 register.

To execute the command, $\overline{CS}$ is driven low. The command code is input on SI. After the command has been received, the contents of the status register are shifted out on the SO signal on the falling edge of the clock MSB first. The operation is terminated by driving $\overline{CS}$ high at any time during data output.

By continuing to hold $\overline{CS}$ low, the updated status register contents will be output on the following data byte. This allows software to poll on the various status bits without needing to continually issue the command cycle as long as $\overline{CS}$ is asserted low for the duration of the polling period.

NOTE: Unlike the READ STATUS1 REGISTER command. The command may not be used at any time such as when program, erase or write operations are in progress.

WRITE STATUS REGISTER

The WRITE STATUS REGISTER command is used to write to the status registers.

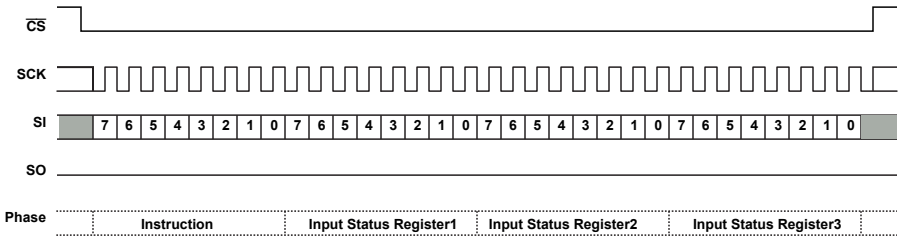

Figure 10-14: Write Status Register Command Sequence

To execute the command, $\overline{CS}$ is driven low. The command code is input on SI, followed by the 3 bytes of input to be written to the STATUS1, STATUS2 and STATUS3 registers. Each input bit is latched in during the rising edge of the clock. The operation is completed by driving $\overline{CS}$ high after the eighth bit of a data value is clocked in the 8-bit boundary of the data input.

While the WRITE STATUS REGISTER operation is in progress, the READ STATUS REGISTER command can be used to check the status of the write operation via the `STATUS1.BUSY` bit. After the operation has completed the `STATUS1.WEL` bit is cleared.

To write non-volatile Status Register bits, a Write Enable command must previously have been executed for the device to accept the Write Status Registers command.

To write volatile Status Register bits, a Write Enable for Volatile Status Register command must previously have been executed for the device to accept the Write Status Registers command.

Program Operations

Flash can be programmed using PAGE PROGRAM command. Below is the description of program command.

PAGE PROGRAM

The PAGE PROGRAM command allows up to 256 bytes of data to be programmed to erased memory locations.

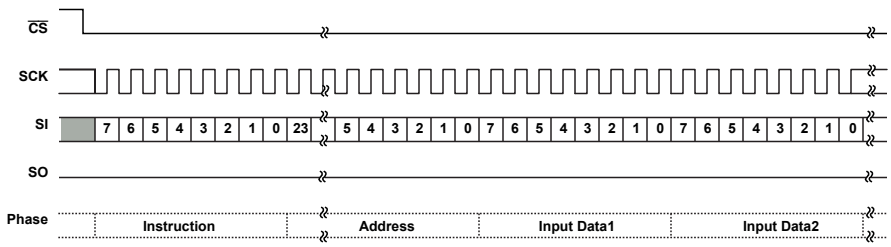

Figure 10-15: Page Program Command Sequence

The command is initiated by first executing the WRITE ENABLE command to set the write enable latch bit to 1. $\overline{CS}$ is then driven low and held low until the eighth bit of the last data byte has been latched in, after which it must be driven high. The command code is input on `SI`, followed by input on `SI` of address bytes and at least one data byte. Each address bit is latched in during the rising edge of the clock. When $\overline{CS}$ is driven high, the operation, which is self-timed, is initiated.

If the 8 least significant address bits, (the starting address of the page) are not all zero, all data transmitted beyond the end of the page is programmed from the starting address of the same page. This results in data wrapping in the page and may not be desirable.

If the number of bytes sent to the device exceed the maximum page size, previously latched data is discarded and only the last page size number of bytes are guaranteed to be programmed correctly within the page. If the number of bytes sent to the device is less than the maximum page size, and the number of bytes sent do not exceed the remaining page length, they are correctly programmed at the specified addresses without any effect on the other bytes of the same page.

When the operation is in progress, the the `STATUS1.BUSY` bit is set to 1. The write enable latch bit is cleared to 0, whether the operation is successful or not.

The STATUS register can be polled for the operation status. If $\overline{CS}$ is not driven high, the command is not executed and the write enable latch remains set to 1. When a command is applied to a protected block, the command is not executed and the write enable latch bit remains set to 1.

Erase Operations

The Flash erase commands can be issued to erase either sector wise, block wise or a bulk erase. It cannot be used to erase page wise.

SECTOR ERASE

The SECTOR ERASE command erases a 4KB flash sector resulting in all bytes being set to 0xFF.

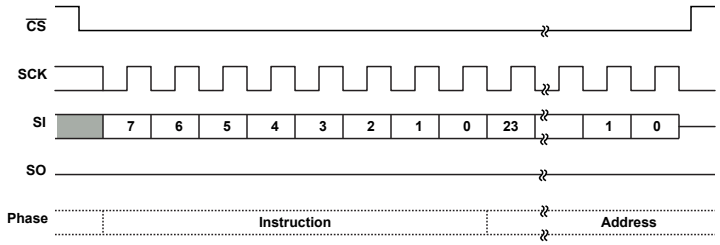

Figure 10-16: Sector Erase Command Sequence

To execute the command the WRITE ENABLE command must be issued to set the write enable latch bit to 1. $\overline{CS}$ is driven low and held low until the eighth bit of the last address byte has been latched in, after which it must be driven high. The command code is input on SI, followed by three address bytes; any address within the sector is valid. Each bit is latched in during the rising edge of the clock. When $\overline{CS}$ is driven high, the operation, which is self-timed, is initiated. The operation is terminated by driving $\overline{CS}$ high.

When the operation is in progress, the write in progress bit is set to 1. The write enable latch bit is cleared to 0, whether the operation is successful or not.

BLOCK ERASE

The BLOCK ERASE command erases a 64KB flash block resulting in all bytes being set to 0xFF.

To execute the command the WRITE ENABLE command must be issued to set the write enable latch bit to 1. $\overline{CS}$ is driven low and held low until the eighth bit of the last address byte has been latched in, after which it must be driven high. The command code is input on SI, followed by three address bytes; any address within the block is valid. Each bit is latched in during the rising edge of the clock. When $\overline{CS}$ is driven high, the operation, which is self-timed, is initiated. The operation is terminated by driving $\overline{CS}$ high.

The status register can be polled for the operation status. When the operation completes, the STATUS1.BUSY bit is cleared to 0.

If the operation times out, the write enable latch bit is reset. If $\overline{CS}$ is not driven high, the command is not executed and the write enable latch remains set to 1. When a command is applied to a protected sector, the command is not executed.

BULK ERASE

The BULK ERASE command erases the entire flash memory contents resulting in all bytes being set to 0xFF.

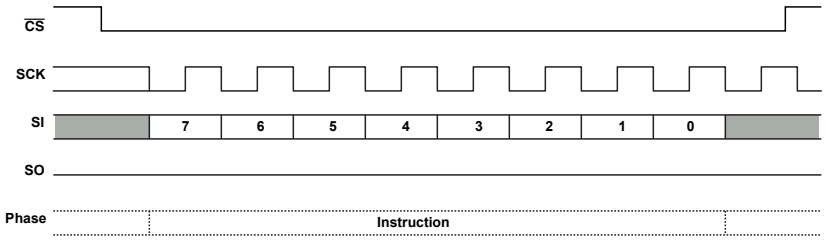

Figure 10-17: Bulk Erase Command Sequence

To execute the command the WRITE ENABLE command must be issued to set the write enable latch bit to 1. $\overline{CS}$ is driven low and held low until the eighth bit of the last data byte has been latched in, after which it must be driven high. The command code is input on `SI`, followed by three address bytes; any address within the block is valid. Each bit is latched in during the rising edge of the clock. When $\overline{CS}$ is driven high, the operation, which is self-timed, is initiated. The operation is terminated by driving $\overline{CS}$ high.

The status register can be polled for the operation status. When the operation completes, the `STATUS1.BUSY` bit is cleared to 0

If the operation times out, the write enable latch bit is reset. If $\overline{CS}$ is not driven high, the command is not executed and the write enable latch remains set to 1. When a command is applied to a protected sector, the command is not executed.

OTP Operations

There are three OTP arrays in flash memory location each of 256 bytes, which can be locked permanently. The OTP arrays in the flash can be programmed, erased and read using different commands. The OTP array location in flash can be permanently locked by writing to `STATUS2.OTPLOCK` bits. Once the OTP lock bits are set the OTP arrays become permanently locked with only read operation valid and erase and program commands will be ignored.

READ OTP ARRAY

The READ OTP ARRAY command allows one or more data bytes to be read sequentially from the OTP area,

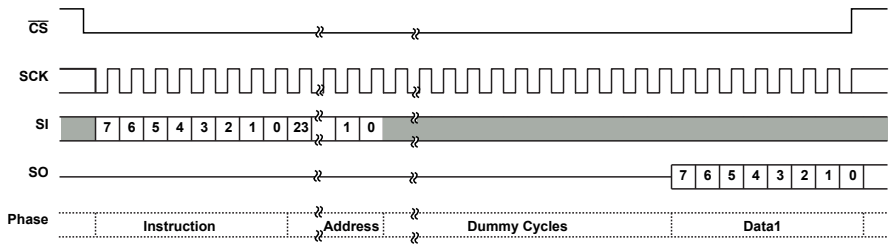

Figure 10-18: Read OTP Array Command Sequence

To initiate the command, $\overline{CS}$ is driven low. The command code is input on SI, followed by three address bytes and eight dummy clock cycles. Each bit is latched in during the rising edge of the clock. Data is shifted out on SO, beginning from the specified address on the falling edge of the clock. The address increments automatically to the next address after each byte of data is shifted out.

If read continuously, the internal address counter rolls over after location 0xFF. It is reset to 0x00 then continues to increment, outputting the data from that address. The operation is terminated by driving $\overline{CS}$ high at any time during data output.

Table 10-10: READ OTP Addresses

Address	A23-16	A15-8	A7-0
OTP Array 1	0x00	0x10	0x00-0xFF
OTP Array 2	0x00	0x20	0x00-0xFF
OTP Array 3	0x00	0x30	0x00-0xFF

PROGRAM OTP ARRAY

The PROGRAM OTP ARRAY command allows one or more bytes to be programmed to the OTP array.

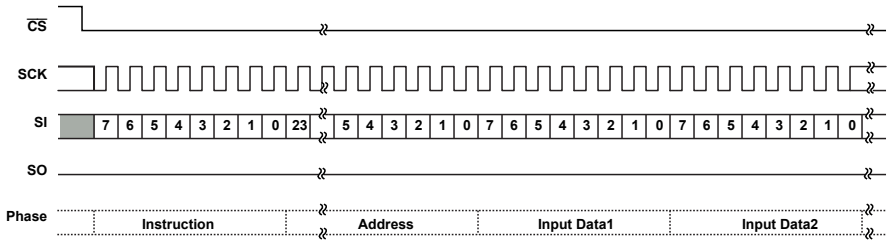

Figure 10-19: Program OTP Array Command Sequence

To initiate the command, the WRITE ENABLE command must be issued to set the write enable latch bit to 1; otherwise, the command is ignored.

To initiate the command, $\overline{CS}$ is driven low. The command code is input on SI, followed by three address bytes and at least a single data byte. Each bit is latched in during the rising edge of the clock. $\overline{CS}$ is driven low and held low until the eighth bit of the last data byte has been latched in, after which it must be driven high. When $\overline{CS}$ is driven high, the operation, which is self-timed, is initiated.

The OTP arrays can be protected via the OTP lock bits in the STATUS2 register. Once a lock bit is set to 1 the OTP array is permanently locked. If the command addresses a locked OTP array the command is not executed and the write enable latch remains set to 1.

ERASE OTP ARRAY

The ERASE OTP ARRAY command allows for the erasing a non protected OTP array, setting all the bytes in the array to 0xFF

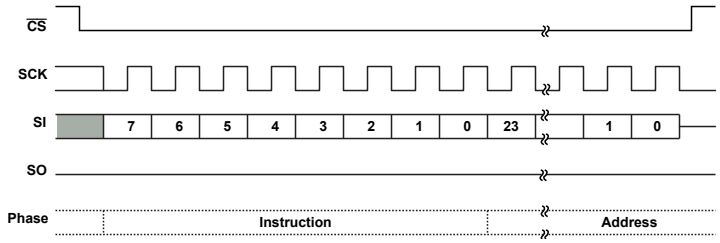

Figure 10-20: Erase Security Registers Command Sequence

To execute the command, the WRITE ENABLE command must be issued to set the write enable latch bit to 1; otherwise, the command is ignored.

$\overline{CS}$ is driven low and held low until the eighth bit of the last address byte has been latched in, after which it must be driven high. The command code is input on `SI`, followed by three address bytes; any address within the OTP array is valid. Each bit is latched in during the rising edge of the clock. When $\overline{CS}$ is driven high, the operation, which is self-timed, is initiated. The operation is terminated by driving $\overline{CS}$ high.

If the write enable latch bit is not set, the device ignores the BLOCK ERASE command.

When the operation is in progress, the write in progress bit is set to 1. The write enable latch bit is cleared to 0, whether the operation is successful or not.

After $\overline{CS}$ is driven high, the status register can be polled for the operation status. When the operation completes, the `STATUS1.BUSY` bit is cleared to 0.

If the operation times out, the write enable latch bit is reset. If $\overline{CS}$ is not driven high, the command is not executed and the write enable latch remains set to 1. When a command is applied to a protected sector, the command is not executed.

Table 10-11: ERASE OTP Addresses

Address	A23-16	A15-8	A7-0
OTP Array 1	0x00	0x10	Don't Care
OTP Array 2	0x00	0x20	Don't Care
OTP Array 3	0x00	0x30	Don't Care

Power Loss Recovery Sequence

In order to bring the SPI flash device back into a mode of operation that allows the SPI controller to re-establish communication with the memory, a Continuous mode reset command is issued.

Continuous Read Mode Reset

The mode configuration bits responsible for controlling the command skip mode are used in conjunction with the Dual I/O Fast Read and Quad I/O Fast Read commands. Issuing a 0xFFFF or 0xFF command on the single bit bus configuration thus driving `SI` high while $\overline{CS}$ is driven low is sufficient to bring the device out of the command skip mode of operation regardless of command that was used to enter the mode in the first instance.

In addition, if the burst wrap mode is being used it is also recommended to issue the Set Burst with Wrap command to set the wrapping back to 8 bits to allowing for standard sequential reads to occur from the memory array. Issuing these commands after a system reset event ensure the device operation is consistent with the power on device operation.

It is recommended that this procedures be executed after every system or hard reset event in order to ensure communication between the SPI controller and the flash memory can be re-established.

NOTE: When the processor is configured for SPI Master boot mode, the above sequence of events is executed in the boot code before attempting to boot or execute code from the on-chip SPI flash memory.

SPI Flash Registers

The SPI Flash registers configure flash operations and report flash status.

STATUS1 Register

Table 10-12: STATUS1 Register Bit Definitions

Bit No.	Bit Name	Description/Enumeration	
7	Reserved	Must be set to 0 (default)	
6	Reserved	Must be set to 0 (default)	
5	BPEN	Block Protect Enable. Non Volatile Determines whether the protected memory area defined by the BP bits are enabled.	
		0 (default)	Disabled
		1	Enabled
4:2	BP	Block Protect. Non-Volatile Determines memory to be software protected against program and erase operations. These only become applicable when `BPEN` is set	
		000 (default)	No blocks protected
		001	Block 0 protected
		010	Blocks 0 and 1 protected
		011	Blocks 0 through 3 protected
		100	Blocks 0 through 7 protected
		101	Blocks 0 through 15 protected
		110	Blocks 0 through 32 protected
		111	Blocks 0 through 32 protected

Table 10-12: STATUS1 Register Bit Definitions (Continued)

Bit No.	Bit Name	Description/Enumeration	
1	WEL	Write Enable Latch. Non-Volatile Read Only When set allows for the execution of program erase and write operations. This bit is set by executing the WRITE ENABLE command	
		0 (default)	Disabled
		1	Enabled
0	BUSY	Device Busy. Non-Volatile Read Only Indicates if the device is busy performing a program, erase or status register write operation	
		0 (default)	Not Busy
		1	Busy

Table 10-13: STATUS1 Register Bit Definitions

Bit No.	Bit Name	Description/Enumeration	
7	Reserved	Must be set to 0 (default)	
6	SEC	Block/Sector Protect Determines whether BP field protects 64kB blocks or 4kB sectors. There is a volatile and non-volatile version available	
		0 (default)	Block Protection
		1	Sector Protection
5	TB	Top/Bottom Protect Controls whether the BP bits protect from the top or the bottom of the memory array. There is a volatile and non-volatile version available	
		0 (default)	Protect from Top down
		1	Protect from Bottom up
4:2	BP	Block Protect Determines memory to be software protected against program and erase operations. These only become applicable when BPEN is set. Refer to the block protection table below. There are volatile and non volatile versions available.	
1	WEL	Write Enable Latch. Volatile and Read Only When set allows for the execution of program erase and write operations. This bit is set by executing the WRITE ENABLE command	
		0 (default)	Disabled
		1	Enabled

Table 10-13: STATUS1 Register Bit Definitions (Continued)

Bit No.	Bit Name	Description/Enumeration	
0	BUSY	Device Busy. Volatile and Read Only Indicates if the device is busy performing a program, erase or status register write operation	
		0 (default)	Not Busy
		1	Busy

STATUS2 Register

Table 10-14: STATUS2 Register Bit Definitions

Bit No.	Bit Name	Description/Enumeration	
7	Reserved	Must be set to 0 (default)	
6	CMP	Complement Protect. Non-Volatile and volatile version Used in conjunction with the `STATUS1.SEC`, `STATUS1.TB` and `STATUS1.BP` bits. It provides more flexibility to the array protection. When set, the array protection is reversed. Protected regions when CMP is 0 become unprotected with CMP enabled and unprotected become protected.	
		0 (default)	Normal Protection Map
		1	Complement Protection Map
5:3	OTPLOCK	OTP Lock Bits Each bit is used to lock a 256 byte region of the OTP memory array. Once set the lock is permanent.	
		000 (default)	OTP Lock Disabled
		001	OTP Array 1 Locked
		010	OTP Array 2 Locked
		011	OTP Array 1 and 2 Locked
		100	OTP Array 3 Locked
		101	OTP Array 1 and 3 Locked
		110	OTP Array 2 and 3 Locked
		111	OTP Array 1, 2 and 3 Locked
2	Reserved	Reserved	
1	QEN	Quad Enable. Non-Volatile and volatile version Enables the `IO2` and `IO3` functionality for quad transfers and disable the $\overline{\text{WP}}$ and $\overline{\text{HOLD}}$ functionality.	
		0 (default)	Disabled
		1	Enabled

Table 10-14: STATUS2 Register Bit Definitions (Continued)

Bit No.	Bit Name	Description/Enumeration	
0	SRP1	Status Register Protect 1. Non-Volatile and volatile version Enables protection on the `STATUS1` and `STATUS2` registers.	
		0 (default)	Status registers can be written after the WRITE ENABLE command
		1	Status registers are protected and cannot be written again until the next power-up cycle

STATUS3 Register

Table 10-15: STATUS3 Register Bit Definitions

Bit No.	Bit Name	Description/Enumeration	
7	Reserved	Must be set to 0 (default)	
6:5	WRAP	Wrap Mode. Volatile version Selects the alignment boundary at which reading will wrap to perform a cache line fill. Reading begins at the initial byte address of a FAST READ QUAD I/O command and sequential bytes are read until the boundary is reached. Reading then wraps to the beginning of the selected boundary. This allows critical word first cache line fills.	
		0 (default)	8-byte Wrap
		1	16-byte Wrap
		1	32-byte Wrap
		1	64-byte Wrap
4	WRAPDIS	Wrap Mode Disable. Volatile version Disables wrap mode such that all reads are sequential.	
		0	Wrap Enabled
		1 (default)	Wrap Disabled
3:0	DCYCLES	Dummy Cycles. Volatile version Sets the number of dummy clock cycles subsequent to all FAST READ commands. Supported values are from 1 through 15 dummy clocks. The default value of 0 enables the legacy standard settings. Refer to Dummy Cycle Configuration	

11 Cyclic Redundancy Check (CRC)

The CRC peripheral performs the cyclic redundancy check (CRC) of the block of data that is presented to the peripheral. The peripheral provides a means to verify periodically the integrity of the system memory, the contents of memory-mapped registers (MMRs), or communication message objects. It is based on a CRC32 engine that computes the signature of 32-bit data presented to the peripheral.

The dedicated hardware compares the calculated signature of the operation to a pre-loaded expected signature. If the two signatures fail to match, the peripheral generates an error.

The source channel of the memory-to-memory DMA channels can provide data. The CRC optionally forwards data to memory through the destination DMA channel. Alternatively, the peripheral supports data presented by core write transactions.

The CRC peripheral implements a reduced table-lookup algorithm to compute the signature of the data. The CRC uses a programmable 32-bit CRC polynomial to generate the lookup table (LUT) contents automatically.

More CRC peripheral modes allow for initializing large memory sections with a constant value, or for verifying that sections of memory are equal to a constant value.

CRC Features

The CRC peripheral supports a number of key features.

- Memory scan modes for memory verification
- Memory transfer modes for on-the-fly CRC calculations while transferring data from one memory to another
- A programmable 32-bit CRC polynomial with automatic LUT generation
- Data mirroring options

The CRC module also includes the following features.

- CRC checksum computation and comparison modes
- 32-bit programmable CRC polynomial with bit reverse option
- Automatic lookup table (LUT) generation
- Data mirroring options for endian and reflected polynomial cases

- Automatic clear and preset of results
- Fault and error interrupt reporting
- DMA and MMR based operation

Because the CRC module is closely tied to memory-to-memory DMA (MDMA) channel pairs, the use cases include the following features.

- Memory scan mode with CRC compute or compare
- Memory transfer mode with CRC compute or compare
- Memory fill operation with 32-bit data patterns
- Memory verify operation
- MMR write access to FIFO of destination DMA
- MMR read access to FIFO of source DMA
- Profiting from advanced DMA features, like descriptor mode and bandwidth control or monitor

CRC Functional Description

The CRC peripheral supports a number of modes of operation that allow for the initialization and verification of regions of memory. The peripheral supports efficient memory-fill and verification operations on regions of memory with or against a constant value. These modes of operation do not require the CRC engine to calculate a signature. Other modes of operation allow for the calculation of CRC signature and verification for a memory region. The modes allow for on-the-fly CRC calculation when performing memory-to-memory DMA transfers from one memory region to another.

To minimize the need for core accesses, the peripheral interfaces with one or more (depending on processor features) memory-to memory DMA (MDMA) channels. This connectivity permits flexible configuration, in which data can be written-to or read-from the peripheral using DMA transactions, core transactions, or a combination of both.

The CRC module supports one DMA channel, providing both a data input and data output. CRC0 connects to the MDMA0 channel pair.

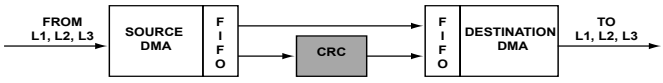

Figure 11-1: Memory Flow

ADSP-CM40x CRC Register List

The Cyclic Redundancy Check (CRC) unit includes the data comparison, polynomial operation, and look up table generation features needed for CRC operation. The CRC provides CRC protection as specified by many functional safety requirements. This unit facilitates the system software's ability to periodically check the correctness of the

code/data available in memory. A set of registers govern CRC operations. For more information on CRC functionality, see the CRC register descriptions.

Table 11-1: ADSP-CM40x CRC Register List

Name	Description
CRC_COMP	Data Compare Register
CRC_CTL	Control Register
CRC_DCNT	Data Word Count Register
CRC_DCNTCAP	Data Count Capture Register
CRC_DCNTRLD	Data Word Count Reload Register
CRC_DFIFO	Data FIFO Register
CRC_FILLVAL	Fill Value Register
CRC_INEN	Interrupt Enable Register
CRC_INEN_CLR	Interrupt Enable Clear Register
CRC_INEN_SET	Interrupt Enable Set Register
CRC_POLY	Polynomial Register
CRC_RESULT_CUR	CRC Current Result Register
CRC_RESULT_FIN	CRC Final Result Register
CRC_STAT	Status Register

ADSP-CM40x CRC Interrupt List

Table 11-2: ADSP-CM40x CRC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
100	CRC0_DCNTEXP	CRC0 Datacount expiration	Level	
101	CRC0_ERR	CRC0 Error	Level	

CRC Definitions

To make the best use of the CRC, it is useful to understand the following terms.

CRC

Acronym for Cyclic Redundancy Check. An error detection code that can detect changes within a block of data.

CRC Polynomial

The 32-bit polynomial used by the CRC engine to generate the Look-Up-Table required for the CRC implementation

LUT

Acronym for the Look-Up-Table. The Look-Up-Table is automatically generated from the supplied 32-bit CRC polynomial.

DMA

Acronym for Direct Memory Access. Used to describe a data transfer that takes place through a DMA channel allowing data distribution around a system without intervention from the core.

MDMA

Acronym for Memory-To-Memory DMA transfer that often requires the use of two DMA channels to transfer data from one memory region to another memory region. One DMA channel is configured as a source channel and the second as a destination channel.

CRC Block Diagram

The ***CRC Block Diagram*** shows the functional block diagram of the CRC. The following sections describe the blocks.

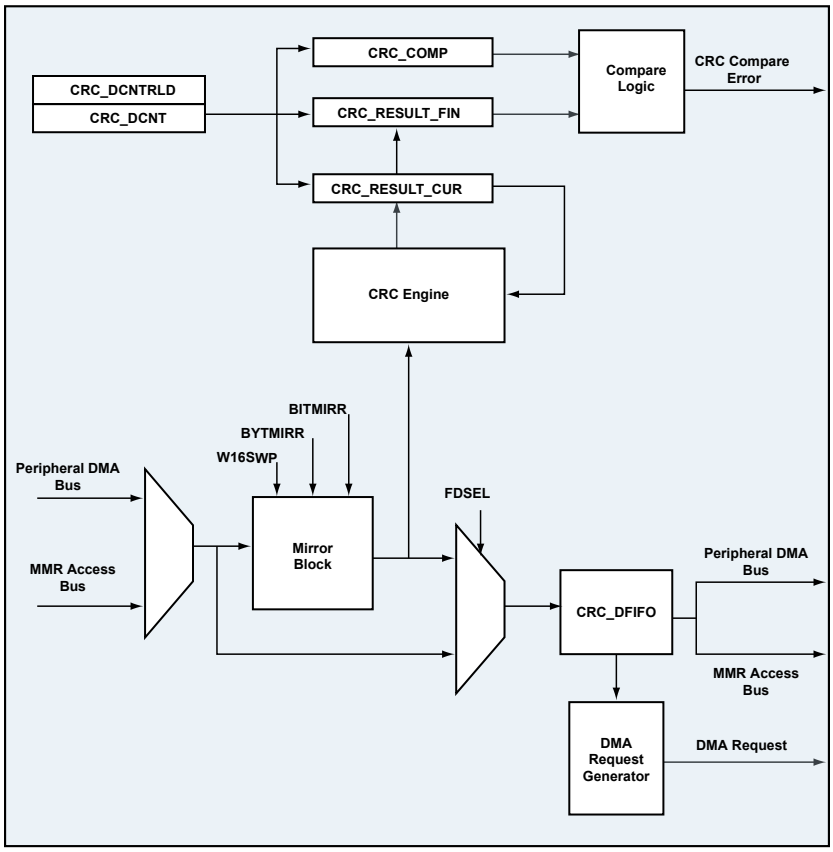

Figure 11-2: CRC Block Diagram

Peripheral DMA Bus

The CRC peripheral provides both an incoming and outgoing datapath to the peripheral DMA bus. The MDMA source channel is interfaced to the incoming datapath providing data to the CRC peripheral. For memory transfer and data fill modes, the CRC uses the MDMA destination channel to either output the data from the CRC FIFO or use the data for the fill operation.

MMR Access Bus

The core uses the MMR access bus to access all the memory-mapped registers of the peripheral for configuration, status, and debug purposes. The core can also use the MMR access bus to feed data to the CRC peripheral or read data from the FIFO of the CRC peripheral. The CRC operation is an alternative to the DMA channel operation to read data from the FIFO.

Data received by MMR-writes can transfer to destination DMA. Similarly, data received by source DMA can be output through the MMR interface. Optionally, intermediate results can be made available to the MMR interface.

Mirror Block

The mirror block individually controls bit-reversing of the polynomial, the computation results, and the expected result. Bit mirroring, byte mirroring, word swapping, and any combination of these operations can control endian and the reflection of processed data.

Data FIFO

The CRC data FIFO is a 32-bit-wide 4-entry FIFO. The FIFO is accessible to both the peripheral DMA bus and the MMR access bus. The FIFO status is accessible from the CRC_STAT register.

DMA Request Generator

The DMA request generator is responsible for granting incoming DMA requests from the source DMA channel and issuing outgoing DMA requests to the destination DMA channel.

CRC Engine

The CRC engine is a 32-bit CRC engine that implements the reduced table lookup scheme. The CRC engine provides support for a user-programmable 32-bit polynomial that the CRC uses to load the lookup table parameters required for the CRC calculation. The CRC engine is a 2-cycle implementation operating on 16 bits of data per cycle.

Compare Logic

The compare logic takes the final CRC signature and compares it to the expected CRC signature, generating a CRC compare error when the signatures do not match. A compare error can flag a system fault.

CRC Architectural Concepts

A 32-bit polynomial is required before calculation of the CRC signature can occur. The CRC uses the polynomial to generate the contents of an internal lookup table that the reduced table lookup implementation requires. The lookup table is automatically generated when the polynomial is written. It must be initialized prior to any operation that requires the use of the CRC engine.

The mirror block logic can manipulate the data presented to the CRC engine before the CRC uses the data in the calculation of the CRC signature. The data mirror operation is configurable to allow for bit reversing, byte reversing, and 16-bit word swapping operations on the incoming data. For memory transfer compute-and-compare operations, programs can configure the peripheral to output the data in the same form in which it was received. Or, the operation can output the mirrored data in the same manner that it is presented to the CRC engine.

While the CRC peripheral is in operation, the status of the FIFO is continually updated and reflected in the CRC_STAT register. The FIFO status is required for core-based accesses to the CRC peripheral. The status indicates when:

- The CRC peripheral can receive data

- Data is available for reading from the FIFO
- The result of the CRC_RESULT_CUR register has been updated

The status of the CRC_RESULT_CUR register indicates that the current CRC calculation has completed and the result is available.

Lookup Table

The lookup table consists of a set of 16 32-bit registers that hardware populates automatically when a write access takes place to the CRC_POLY register. 16 clock cycles are required to generate all 16 lookup table entries. The status of the process for generating the lookup table is reflected in CRC_STAT.LUTDONE allowing for software to poll on the completion of the event or for generation of an interrupt.

NOTE: Hardware must populate the lookup table before any operation using the CRC peripheral can take place, even if the operation does not use the CRC engine. The peripheral does not issue any data requests until the table generation process is complete. In addition, the CRC_STAT.IBR field, that indicates the input buffer status as required for core-based transfers, is only valid upon completion of the process for generating the lookup table.

Data Mirroring

The data mirror block can be configured to manipulate the incoming data before the data passes to the CRC engine and, optionally, to the FIFO. This configuration allows the peripheral to handle various forms of endianness and to function with reflected polynomials.

There are three configuration bits that control the data mirroring process: CRC_CTL.BITMIRR, CRC_CTL.BYTMIRR, and CRC_CTL.W16SWP. The ***Data Mirroring Options*** table details how these options affect the incoming data and the output generated by the mirror block.

Table 11-3: Data Mirroring Options

W16SWP	BYTMIRR	BITMIRR	Output Data
0	0	0	Dout[31:0] = Din[31:0]
0	0	1	Dout[31:0] = Din[24:31],Din[16:23],Din[8:15],Din[0:7]
0	1	0	Dout[31:0] = Din[7:0],Din[15:8],Din[23:16],Din[31:24]
0	1	1	Dout[31:0] = Din[0:7],Din[8:15],Din[16:23],Din[24:31]
1	0	0	Dout[31:0] = Din[15:0], D[31:16]
1	0	1	Dout[31:0] = Din[8:15],Din[0:7], Din[24:31],Din[16:23]
1	1	0	Dout[31:0] = Din[23:16],Din[31:24], Din[7:0],Din[15:8]
1	1	1	Dout[31:0] = Din[16:23],Din[24:31], Din[0:7],Din[8:15]

When the CRC is configured to operate in the memory transfer compute-and-compare mode, the bit-reversed output data can be written to the FIFO. This feature is controlled through the CRC_CTL.FDSEL field.

In addition to providing bit swapping and mirror options to the incoming data, the CRC peripheral also supports bit mirroring on the following registers.

- CRC_RESULT_CUR and CRC_RESULT_FIN, controlled through the CRC_CTL.RSLTMIRR field. When mirroring is enabled, the values to be written to these registers are fully bit-reversed before the write operation occurs.
- CRC_POLY, controlled through the CRC_CTL.POLYMIRR field. When mirroring is enabled, the 32-bit polynomial is fully bit-reversed before the write operation to the register occurs.
- CRC_COMP, controlled through the CRC_CTL.CMPMIRR field. When mirroring is enabled, the contents to be loaded to this register are fully bit-reversed before the write operation occurs.

FIFO Status and Data Requests

The CRC peripheral provides indication of the input and output buffer status through CRC_STAT.IBR and CRC_STAT.OBR respectively. For core-based operations, software must monitor these status fields prior to writing to or reading from the CRC FIFO. No write to the CRC FIFO can occur while CRC_STAT.IBR indicates that the buffer is not ready to accept data. Similarly, the CRC FIFO cannot be read until CRC_STAT.OBR indicates that data is available.

The memory scan modes of operation only require the monitoring of the input buffer status. The memory transfer, compute-and-compare mode uses both input and output buffer status. If the current result of the CRC computation is required, then software must verify that the current operation has completed and that the intermediate result is ready. The CRC_STAT.IRR indicates the status.

NOTE: The memory transfer fill mode of operation requires the use of a DMA channel. The CRC does not support core reads from the CRC FIFO for this mode of operation.

Memory transfer, compute-and-compare mode uses burst transactions to make the most efficient use of the available resources. In this mode, when the FIFO is initially empty and the peripheral is enabled, the CRC_STAT.IBR bit indicates that the CRC is ready to accept data. The peripheral generates data requests to the source DMA channel (if CRC uses DMA). While the number of words remaining in the CRC_DCNT register is greater than the FIFO depth, the peripheral issues data requests or accepts incoming data in bursts. The peripheral continues until the CRC FIFO becomes full.

Once full, the CRC_STAT.IBR and CRC_STAT.OBR bits are updated, and then the CRC issues outgoing data requests. Only when the FIFO is empty can the peripheral accept further incoming data, and the CRC_STAT.IBR and CRC_STAT.OBR bits are updated once again.

Once CRC_DCNT is decremented such that the number of words waiting for processing is less than the number required to fill the FIFO, the burst mode of operation is disabled. Incoming data is accepted as long as the FIFO is not full. Outgoing data is available as long the FIFO is not empty. Therefore, there are no restrictions requiring the word count to be a multiple of the FIFO depth.

All other CRC modes of operation indicate that incoming data can be accepted as long as the FIFO is not full. Outgoing data is available as long the FIFO is not empty.

The CRC_CTL.OBRSTALL and CRC_CTL.IRRSTALL bit configurations also influence how the CRC generates data requests and status bits. The following list describes the bits.

- The CRC_CTL.OBRSTALL bit can be configured such that the CRC peripheral stalls as soon as there is output data available in the FIFO. Use this mode of operation only in memory transfer, compute-and-compare mode. This mode results in the processing of one 32-bit word at a time. The peripheral does not request or accept incoming data until the current value being processed is read from the peripheral.
- The CRC_CTL.IRRSTALL bit can be configured so that the CRC peripheral stalls all further incoming data requests until the CRC_RESULT_CUR register is read after being updated. Use this mode of operation for CRC signature generation. It is not applicable to memory transfer data fill mode or memory scan data verify mode of operation.

CRC Operating Modes

The following sections describe the various operating modes of the CRC interface.

Data Transfer Modes

The CRC peripheral supports two main categories of operation involving data transfers:

- Memory scan mode
- Memory transfer mode

Memory scan modes are read-only operations that allow the contents of memory to be read into the peripheral and verified for correctness. There are two forms of memory scan mode:

- CRC compute-and-compare performs a CRC calculation on data presented to the peripheral and compares the CRC result with a pre-determined and pre-loaded result. An error is generated when the results differ.
- Data verify compares each 32-bit data word presented to the CRC peripheral to a pre-loaded 32-bit value and generates an error when the data differs.

Both of these modes of operation require, at the most, a single DMA channel to read the data from memory into the peripheral. No data is forwarded to data output or destination DMA. The CRC can also use core-driven transfers for either of these modes of operation.

The memory transfer modes involve memory write or memory read-and-write operations allowing for memory to be initialized or transferred from one region of memory to another. There are two forms of memory transfer mode:

- CRC compute-and-compare performs a full data transfer from one memory region to another memory region. The CRC generates a signature on the data presented and compares it with a pre-determined and pre-loaded result. An error is generated when the results differ.
- Data fill initializes a region of memory with a pre-loaded 32-bit constant value.

The CRC compute-and-compare mode of operation requires both incoming and outgoing data channels. The operation occurs either using DMA channels, using core-driven write or read operations to and from the FIFO or using

a combination of both. The data fill mode of operation requires only a memory write DMA destination channel—this mode does not support core driven operations.

Memory Scan Compute and Compare

In this mode of operation, the CRC engine of the peripheral is enabled. The mode is configured through the CRC_CTL.OPMODE field and the CRC engine performs a 32-bit CRC operation on the incoming data stream.

The length of the data stream is configured through the CRC_DCNT register. The accumulated result of the CRC operation is contained in the CRC_RESULT_CUR register. As the CRC engine processes each 32-bit word, the CRC_DCNT register is decremented and CRC_RESULT_CUR is updated.

Once CRC_DCNT decrements to zero, the contents of the CRC_RESULT_CUR register are copied to CRC_RESULT_FIN and CRC_STAT.DCNTEXP is updated accordingly. The CRC uses the CRC_COMP register to store the expected result of the operation. After the CRC calculation, CRC_COMP is compared with CRC_RESULT_FIN and CRC_STAT.CMPERR is updated to reflect the status of the compare operation. CRC_STAT.CMPERR must be cleared before the next CRC operation is performed.

The CRC peripheral also contains CRC_DCNTRLD register. The CRC uses this register to reload CRC_DCNT upon completion of the CRC operation in preparation for the next transfer.

The initial seed of the CRC computation can be configured through CRC_CTL.AUTOCLRZ and CRC_CTL.AUTOCLRF. This configuration provides a way to reset CRC_RESULT_CUR to 0x00000000, 0xFFFFFFFF or to leave the current register contents untouched for the next operation.

The peripheral can be configured to allow for the compare error and data expiration events to generate an interrupt.

Memory Scan Data Verify

In this mode of operation, the CRC engine of the peripheral is not required. The mode is enabled through the CRC_CTL.OPMODE field. Each 32-bit word of the data stream is compared with a constant value that is stored in the CRC_COMP register. The CRC_DCNT register contains the number of words for comparison. The CRC_DCNT register is decremented upon receiving a new 32-bit word from the data stream. If the compare operation fails, the CRC_STAT.CMPERR bit is updated and the contents of CRC_DCNT are captured in the CRC_DCNTCAP register. This information can be used to identify the location in the data stream where the error occurred. Clear the CRC_STAT.CMPERR field to reenable capturing of further errors.

Once CRC_DCNT decrements to zero, CRC_STAT.DCNTEXP is updated accordingly to signal the end of the operation. The peripheral can be configured to allow for the compare error and data expiration events to generate an interrupt.

Memory Transfer Compute and Compare

In this mode of operation, the CRC engine of the peripheral is enabled. The mode is configured through the CRC_CTL.OPMODE field and the CRC engine performs a 32-bit CRC operation on the incoming data stream.

The length of the data stream is configured through the CRC_DCNT register. The accumulated result of the CRC operation is contained in the CRC_RESULT_CUR register. As the CRC engine processes each 32-bit word, the CRC_DCNT register is decremented and CRC_RESULT_CUR is updated.

Once CRC_DCNT decrements to zero, the contents of the CRC_RESULT_CUR register are copied to CRC_RESULT_FIN and CRC_STAT.DCNTEXP is updated accordingly. The CRC uses the CRC_COMP register to store the expected result of the operation. Upon completion of the CRC calculation, CRC_COMP is compared with CRC_RESULT_FIN and CRC_STAT.CMPERR is updated to reflect the status of the compare operation. Clear CRC_STAT.CMPERR before the next CRC operation is performed.

The CRC peripheral also contains CRC_DCNTRLD register. The CRC uses this register to reload CRC_DCNT upon completion of the CRC operation in preparation for the next transfer.

The initial seed of the CRC computation can be configured through CRC_CTL.AUTOCLRZ and CRC_CTL.AUTOCLRF. This configuration provides a means to reset CRC_RESULT_CUR to 0x00000000, 0xFFFFFFFF or to leave the current register contents untouched for the next operation.

The peripheral can be configured to allow for the compare error and data expiration events to generate an interrupt.

Memory Transfer Data Fill Mode

In this mode of operation, the CRC engine of the peripheral is not required. The mode is enabled through the CRC_CTL.OPMODE field. The CRC_FILLVAL register is written with a 32-bit value. The CRC uses this value to initialize a block memory through the memory-to-memory DMA destination channel. When the CRC peripheral and the DMA destination channel are enabled, the contents of the CRC_FILLVAL register is written to the DMA channel to initialize the memory region. The CRC_DCNT register contains the number of words for the write operation.

Once CRC_DCNT decrements to zero, CRC_STAT.DCNTEXP is updated accordingly to signal the end of the operation. The peripheral can be configured to allow for the data expiration event to generate an interrupt.

CRC Event Control

The CRC peripheral can enable certain CRC status operations to generate an interrupt event to the system event controller. There, a CRC error can be qualified as a system fault.

Interrupt Signals

The CRC peripheral can generate two interrupts that are optionally enabled within the system event controller. One is a CRC status interrupt and the other is a CRC error interrupt.

The CRC_STAT.CMPERR status bit can be configured as an interrupt and is signaled through the CRC error interrupt signal. The CRC_STAT.CMPERR status field is set whenever the CRC peripheral performs a compare operation that fails. This status can be the result of a failed memory scan data-verify operation that compares the contents of a memory range with a constant 32-bit value. Or, it can be the result of a CRC signature calculated for a memory region that does not match the expected pre-programmed result for a memory-compare operation.

The CRC_STAT.DCNTEXP status bit is set when the CRC_DCNT register has decremented to zero. The status indicates that the CRC peripheral has now processed all the data requested for the current CRC operation. The CRC can also use this signal to generate an interrupt. The interrupt is signaled on the CRC status interrupt signal.

Both these status bits can be configured to generate and interrupt through the CRC_INEN register. The CRC_INEN register also has bit set, CRC_INEN_SET, and bit clear CRC_INEN_CLR equivalent registers that the CRC uses for the enabling and disabling of these interrupt sources.

The CRC_STAT register has two write one to clear (W1C) fields for clearing the two interrupt sources.

NOTE: Disabling the CRC peripheral through the CRC_CTL.BLKEN bit does not result in the clearing of interrupt sources. Clear the interrupt sources using a W1C operation to the CRC_STAT register.

CRC Programming Model

It is important to note the following restrictions when using the CRC peripheral with the DMA channels:

1. When enabling the CRC peripheral and the DMA channels, enable the CRC peripheral prior to enabling the DMA channels.
2. When disabling the CRC peripheral and the DMA channels, disable the DMA channels prior to disabling the CRC peripheral.

CRC Mode Configuration

Describes a number of tasks showing the various operation modes of the CRC peripheral.

- Look-up Table Generation
- Core Driven Memory Scan Compute and Compare Mode
- DMA Driven Memory Scan Compute and Compare Mode
- Core Driven Memory Scan Data Verify Mode
- DMA Driven Memory Scan Data Verify Mode
- Core Driven Memory Transfer Compute and Compare Mode
- DMA Driven Memory Transfer Compute and Compare Mode
- DMA Driven Memory Transfer Data Fill Mode

Look-up Table Generation

Describes the steps required to initialize the CRC peripheral LUT.

1. Write the 32-bit CRC polynomial of choice to the CRC_POLY register.

   ADDITIONAL INFORMATION: This operation results in the CRC peripheral starting the LUT initialization process. The CRC_STAT.LUTDONE bit is updated to reflect the operation is in progress.

2. Poll the CRC_STAT.LUTDONE bit until the status bit indicates that the operation is completed.

The CRC peripheral has completed initialization of all the LUT registers and is now ready for data operations. The CRC_STAT.LUTDONE bit remains in the current state until the CRC_POLY register is written again, or the peripheral or processor are reset.

Core Driven Memory Scan Compute and Compare Mode

Performs CRC signature calculation and verification for a region of memory using core transactions. The CRC peripheral is configured such that it operates in burst mode due to the stalling options configured through disabling the CRC_CTL register.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per CRC_CTL.BLKEN

1. Initialize the CRC_DCNT register.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the CRC_DCNTRLD register.

   ADDITIONAL INFORMATION: This value is used to reload the CRC_DCNT register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the CRC_RESULT_CUR register.

   ADDITIONAL INFORMATION: This register can be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize the CRC_COMP register.

   ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that the software uses in the final compare operation.

5. Initialize the CRC_INEN register.

   ADDITIONAL INFORMATION: The CRC uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts. If enabled, ensure that the corresponding interrupt handlers are also configured.

6. Initialize CRC_CTL register with the CRC_CTL.OPMODE bit set to memory scan compute-and-compare mode and the CRC_CTL.BLKEN bit configured to enable the CRC peripheral.

   - Disable the CRC_CTL.OBRSTALL and CRC_CTL.IRRSTALL bit options for this task example.

- Configure all mirroring and bit reversal options.
- Configure CRC auto clear options.

The CRC peripheral is now enabled and ready for the core or DMA channel to write data.

7. Write memory region data to the CRC peripheral.
   a. While CRC_STAT.IBR bit indicates that the input buffer is ready, write the CRC_DFIFO register with 32-bit data.

      ADDITIONAL INFORMATION: Repeat this step until all data has been written.
8. Poll the CRC_STAT.DCNTEXP bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures that all the data has been processed.
9. Poll the CRC_STAT.CMPERR bit if the interrupt was disabled to check for a compare error.

   ADDITIONAL INFORMATION: Perform this step only if the compare error interrupt is not enabled.
10. Write the CRC_STAT register to clear both CRC_STAT.DCNTEXP and CRC_STAT.CMPERR bits.

    ADDITIONAL INFORMATION: If interrupts were enabled, then clear of these status bits within the interrupt handlers for the respective interrupts.

    The CRC compute-and-compare operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation.

The integrity check of the memory through the expected CRC signature has completed. The final result is indicated through the CRC_STAT.CMPERR bit and the corresponding interrupt when enabled.

Clear any W1C CRC status bits before performing more CRC operations.

DMA Driven Memory Scan Compute and Compare Mode

Performs CRC signature calculation and verification for a region of memory using DMA transactions. The CRC peripheral is configured such that it operates in the burst mode of operation due to the stalling options configured through disabling CRC_CTL.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per the CRC_CTL.BLKEN bit.

1. Initialize the CRC_DCNT register.

ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the CRC_DCNTRLD register.

   ADDITIONAL INFORMATION: This value is used to reload the CRC_DCNT register upon completion of current operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the CRC_RESULT_CUR register.

   ADDITIONAL INFORMATION: This register can be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize the CRC_COMP register.

   ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that the software uses in the final operation.

5. Initialize the CRC_INEN register.

   ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts, as needed. If enabled, ensure that the corresponding interrupt handlers are also configured.

6. Initialize the CRC_CTL register with the CRC_CTL.OPMODE bit set to memory scan compute compare mode and the CRC_CTL.BLKEN bit configured to enable the CRC peripheral.

   - Disable the CRC_CTL.OBRSTALL and CRC_CTL.IRRSTALL bit options for this task example.
   - Configure all mirroring and bit reversal options.
   - Configure all CRC auto clear options.

   The CRC peripheral is now enabled and ready for the core or DMA channel to write data.

7. Configure and enable the memory-to-memory source DMA channel for memory read STOP mode.

   ADDITIONAL INFORMATION: This step starts the data transfer from the memory region and writes the data to the CRC peripheral.

8. Poll the CRC_STAT.DCNTEXP bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if the counter expired interrupt is disabled. Polling ensures all the data has been processed.

9. Poll the CRC_STAT.CMPERR bit if the interrupt was disabled to check for a compare error.

   ADDITIONAL INFORMATION: Perform this step only if the compare error interrupt is not enabled.

10. Write the CRC_STAT register to clear both the CRC_STAT.DCNTEXP and CRC_STAT.CMPERR bits.

    ADDITIONAL INFORMATION: If interrupts were enabled, then clear these status bits within the interrupt handlers for the respective interrupts.

The CRC compute-and-compare operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation.

The integrity check of the memory through the expected CRC signature has completed and the final result indicated is through CRC_STAT.CMPERR and the corresponding interrupt, when enabled.

Clear any W1C CRC status bits before performing a further CRC operation. Clear any W1C status bits of the memory-to-memory source DMA channel before the next CRC operation.

Core Driven Memory Scan Data Verify Mode

Reads a region of memory using core transactions and performs a compare operation on each 32-bit word against a single pre-loaded 32-bit constant. The compare error interrupt is enabled to capture and log the location of any compare errors.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per CRC_CTL.BLKEN

The interrupt service routine for the compare error interrupt reads and stores the contents of CRC_DCNTCAP register to a buffer before clearing the compare error interrupt.

1. Initialize the CRC_DCNT register.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the CRC_DCNTRLD register.

   ADDITIONAL INFORMATION: This value is used to reload the CRC_DCNT register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the CRC_COMP register.

   ADDITIONAL INFORMATION: This register contains the 32-bit constant that the memory region is expected to be filled with. Each 32 bit of data presented to the peripheral is compared with this value.

4. Initialize the CRC_INEN register.

   ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts. If enabled, ensure that the corresponding interrupt handlers are also configured.

5. Initialize the CRC_CTL register with the CRC_CTL.OPMODE bit set to memory scan data verify mode and the CRC_CTL.BLKEN bit configured to enable the CRC peripheral.

The CRC peripheral is now enabled and ready for the core or DMA channel to write data.

6. Write memory region data to the CRC peripheral.
   a. Poll the CRC_STAT.IBR bit until input buffer is ready.
   b. Write the CRC_DFIFO register with 32-bit data.

      ADDITIONAL INFORMATION: Repeat these two steps until the entire memory region has been written to the CRC peripheral.
7. Poll the CRC_INEN_SET.DCNTEXP bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures all the data has been processed.
8. Check if the buffer used to capture the CRC_DCNTCAP register upon a compare error has any new entries.

   ADDITIONAL INFORMATION: The values captures in the buffer provide a means to locate where in the memory region the failures occurred.
9. Write CRC_STAT to clear both the CRC_INEN_SET.DCNTEXP and CRC_INEN.CMPERR bits.

   ADDITIONAL INFORMATION: If interrupts were enabled, the clear these status bits within the interrupt handlers for the respective interrupts.

   The CRC memory scan-verify operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation.

The result of the integrity check of the memory with the 32-bit constant is indicated through the CRC_INEN.CMPERR bit and the corresponding interrupt, when enabled. Each comparison error is traceable due to the logging of CRC_DCNTCAP from within the compare error interrupt handler.

Clear any W1C CRC status bits before performing a further CRC operation.

DMA Driven Memory Scan Data Verify Mode

Reads a region of memory using DMA transactions and performs a compare operation on each 32-bit word against a single pre-loaded 32-bit constant. The compare error interrupt is enabled to capture and log the location of any compare errors.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per the CRC_CTL.BLKEN bit

The interrupt service routine for the compare error interrupt reads and stores the contents of the CRC_DCNTCAP register to a buffer before clearing the compare error interrupt.

1. Initialize the CRC_DCNT register.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the CRC_DCNTRLD register.

   ADDITIONAL INFORMATION: The CRC module uses this register to reload the CRC_DCNT register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the CRC_COMP register.

   ADDITIONAL INFORMATION: This register contains the 32-bit constant that the memory region is expected to be filled with. Each 32 bit of data presented to the peripheral is compared with this value.

4. Initialize the CRC_INEN register.

   ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts, as needed. If enabled, ensure that the corresponding interrupt handlers are also configured.

5. Initialize the CRC_CTL register with the CRC_CTL.OPMODE bit set to memory scan data verify mode and CRC_CTL.BLKEN configured to enable the CRC peripheral.

   The CRC peripheral is now enabled and ready for the core or DMA channel to write the data.

6. Configure and enable the memory-to-memory source DMA channel for memory read STOP mode.

   ADDITIONAL INFORMATION: This step starts the data transfer from the memory region and writes the data to the CRC peripheral.

7. Poll the CRC_STAT.DCNTEXP bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures all the data has been processed.

8. Check if the buffer used to capture the CRC_DCNTCAP register upon a compare error has any new entries.

   ADDITIONAL INFORMATION: The values captures in the buffer provide a means to locate where in the memory region the failures occurred.

9. Write the CRC_STAT register to clear both the CRC_STAT.DCNTEXP and CRC_STAT.CMPERR bits.

   ADDITIONAL INFORMATION: If interrupts were enabled, then clear these status bits within the interrupt handlers for the respective interrupts.

   The CRC memory scan-verify operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation.

The result of the integrity check of the memory with the 32-bit constant is indicated through the CRC_STAT.CMPERR bit and the corresponding interrupt when enabled. Each comparison error is traceable due to the logging of the CRC_DCNTCAP register from within the compare error interrupt handler.

Clear any W1C CRC status bits and DMA status bits before performing a further CRC operation.

Core Driven Memory Transfer Compute and Compare Mode

Performs CRC signature calculation and verification for a region of memory using core transactions while copying the contents to another memory region. The CRC peripheral is configured such that it operates in the burst mode of operation due to the stalling options configured through disabling the CRC_CTL register.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per the CRC_CTL.BLKEN bit

1. Initialize the CRC_DCNT register.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the CRC_DCNTRLD register.

   ADDITIONAL INFORMATION: This value is used to reload the CRC_DCNT register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the CRC_RESULT_CUR register.

   ADDITIONAL INFORMATION: This register can be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize the CRC_COMP register.

   ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that the software uses in the final compare operation.

5. Initialize the CRC_INEN register.

   ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts, as needed. If enabled, ensure that the corresponding interrupt handlers are also configured.

6. Initialize the CRC_CTL register with the CRC_CTL.OPMODE bit set to memory scan compute-and-compare mode and the CRC_CTL.BLKEN bit configured to enable the CRC peripheral.

   a. Disable the CRC_CTL.OBRSTALL bit and the CRC_CTL.IRRSTALL bit options for this task example.

b. Configure all mirroring and bit reversal options.

c. Configure CRC auto clear options

The CRC peripheral is now enabled and ready for the core or DMA channel to write data.

7. Write memory region data to the CRC peripheral and read it back to the new destination.

   a. While the CRC_STAT.IBR bit indicates that the input buffer is ready, write the CRC_DFIFO register with 32-bit data.

   b. While the CRC_STAT.OBR bit indicates that the output buffer is ready, read the CRC_DFIFO register and store data to new destination.

   ADDITIONAL INFORMATION: Repeat these two steps until all required data has been processed through the CRC peripheral and copied to the new destination.

8. Poll the CRC_STAT.DCNTEXP bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if the counter expired interrupt is disabled. Polling ensures all the data has been processed.

9. Poll the CRC_STAT.CMPERR bit if the interrupt was disabled to check for a compare error.

   ADDITIONAL INFORMATION: Perform this step only if the compare error interrupt is not enabled.

10. Write the CRC_STAT register to clear both CRC_STAT.DCNTEXP and CRC_STAT.CMPERR bits.

    ADDITIONAL INFORMATION: If interrupts were enabled, then clear these status bits within the interrupt handlers for the respective interrupts.

    The CRC compute-and-compare operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation. The memory region has also been copied to its new destination.

The memory region has been copied to a new location and an integrity check of the memory through the expected CRC signature has also completed. The final result is indicated through the CRC_STAT.CMPERR bit and the corresponding interrupt when enabled.

Clear any W1C CRC status bits before performing a further CRC operation.

DMA Driven Memory Transfer Compute and Compare Mode

Performs CRC signature calculation and verification for a region of memory using DMA transactions. The memory region is also copied to another memory region using memory-to-memory DMA transfers. The CRC peripheral is configured such that it operates in burst mode due to the stalling options configured through disabling CRC_CTL.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)

- The CRC block is disabled per the CRC_CTL.BLKEN register.

1. Initialize the CRC_DCNT register.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the CRC_DCNTRLD register.

   ADDITIONAL INFORMATION: This value is used to reload the CRC_DCNT register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the CRC_RESULT_CUR register.

   ADDITIONAL INFORMATION: This register can be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize the CRC_COMP register.

   ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that the software uses in the final compare operation.

5. Initialize the CRC_INEN register.

   ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts, as needed. If enabled, ensure that the corresponding interrupt handlers are also configured.

6. Initialize the CRC_CTL register with the CRC_CTL.OPMODE bit set to memory scan compute compare mode and CRC_CTL.BLKEN configured to enable the CRC peripheral.

   a. Disable the CRC_CTL.OBRSTALL and the CRC_CTL.IRRSTALL bit options for this task example.

   b. Configure all mirroring and bit reversal options

   c. Configure CRC auto clear options

   The CRC peripheral is now enabled and ready for the core or DMA channel to write data.

7. Configure and enable the memory-to-memory source DMA channel for memory read STOP mode and destination DMA channel for memory write STOP mode.

   ADDITIONAL INFORMATION: This step starts the data transfer from one memory region to another through the memory-to-memory DMA channels and the CRC peripheral.

8. Poll the CRC_STAT.DCNTEXP bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures all the data has been processed.

9. Poll the CRC_STAT.CMPERR bit if the interrupt was disabled to check for a compare error.

   ADDITIONAL INFORMATION: Perform this step only if the compare error interrupt is not enabled.

10. Write the CRC_STAT register to clear both the CRC_STAT.DCNTEXP and the CRC_STAT.CMPERR bits.

    ADDITIONAL INFORMATION: If interrupts were enabled, then clear these status bits within the interrupt handlers for the respective interrupts.

    The CRC compute-and-compare operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation. The memory region has also been copied to its new destination.

The integrity check of the memory through the expected CRC signature has completed and the final result is indicated through the CRC_STAT.CMPERR bit and the corresponding interrupt when enabled. The memory region has also been copied to its final destination.

Clear any W1C CRC status bits before performing a further CRC operation. Also, clear any W1C status bits of the memory-to-memory source and destination DMA channels before the next CRC operation.

DMA Driven Memory Transfer Data Fill Mode

Initializes a region of memory to a constant 32-bit value using DMA transactions.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per the CRC_CTL.BLKEN bit

1. Initialize the CRC_DCNT register.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the CRC_DCNTRLD register.

   ADDITIONAL INFORMATION: This value is used to reload the CRC_DCNT register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the CRC_FILLVAL register.

   ADDITIONAL INFORMATION: This register contains the 32-bit constant that the CRC module uses to fill the memory region.

4. Initialize the CRC_INEN register.

   ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of block completion. Configure these interrupts as required. If enabled, ensure that the corresponding interrupt handlers are also configured.

5. Initialize the CRC_CTL register with the CRC_CTL.OPMODE bit set to memory transfer fill mode and the CRC_CTL.BLKEN bit configured to enable the CRC peripheral.

The CRC peripheral is now enabled and is ready for the DMA channel to write data.

6. Configure and enable the memory-to-memory destination DMA channel for memory write STOP mode.

   ADDITIONAL INFORMATION: This step starts the data transfer taking the constant 32-bit value from the CRC peripheral and writing the data to the DMA channel.

7. Poll the `CRC_STAT.DCNTEXP` bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures that all the data has been processed.

8. Write the `CRC_STAT` register to clear the `CRC_STAT.DCNTEXP` bit.

   ADDITIONAL INFORMATION: If interrupts were enabled, then clear this status bit within the interrupt handlers for the respective interrupts.

   The CRC memory transfer fill operation is now complete and the CRC peripheral is ready to be configured for the next CRC operation.

The memory region is now filled with the constant data and the CRC peripheral is ready to be configured for a new operation.

Clear any W1C CRC status bits and DMA status bits before performing a further CRC operation.

ADSP-CM40x CRC Peripheral and DMA Channel List

Table 11-4: CRC DMA Channels

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/Monitor Support
DMA17	CRC0 Receive	128	Yes
DMA18	CRC0 Transmit	64	Yes

Table 11-5: CRC DMA Channels (Continued)

DMA Channel	Memory Bus Width	Peripheral Bus Width	Max Outstanding Reads	Max Outstanding Writes
DMA17	32-bit	32-bit	8	7
DMA18	32-bit	32-bit	8	4

ADSP-CM40x CRC Register Descriptions

Cyclic Redundancy Check Unit (CRC) contains the following registers.

Table 11-6: ADSP-CM40x CRC Register List

Name	Description
	Data Compare Register

Table 11-6: ADSP-CM40x CRC Register List (Continued)

Name	Description
CRC_COMP	
CRC_CTL	Control Register
CRC_DCNT	Data Word Count Register
CRC_DCNTCAP	Data Count Capture Register
CRC_DCNTRLD	Data Word Count Reload Register
CRC_DFIFO	Data FIFO Register
CRC_FILLVAL	Fill Value Register
CRC_INEN	Interrupt Enable Register
CRC_INEN_CLR	Interrupt Enable Clear Register
CRC_INEN_SET	Interrupt Enable Set Register
CRC_POLY	Polynomial Register
CRC_RESULT_CUR	CRC Current Result Register
CRC_RESULT_FIN	CRC Final Result Register
CRC_STAT	Status Register

Data Compare Register

The CRC_COMP contains the value corresponding to the expected CRC result or signature for the current data stream. At the end of the operation, the content of this register is used to compare against the result produced by the CRC operation. In data verify mode, each incoming data value is compared with the content of this register.

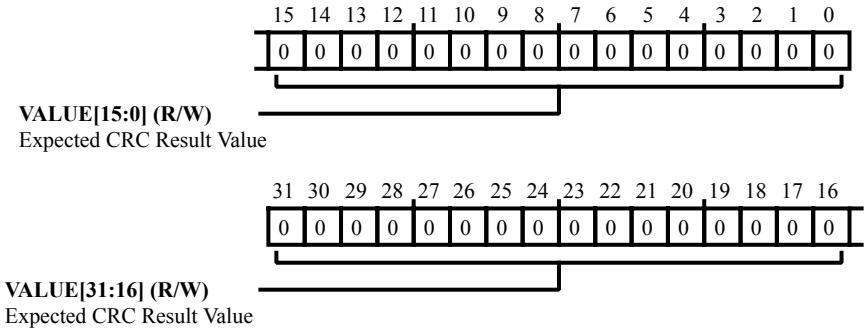

Figure 11-3: CRC_COMP Register Diagram

Table 11-7: CRC_COMP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Expected CRC Result Value. The CRC_COMP.VALUE bit field contains the value corresponding to the expected CRC result or signature for the current data stream.

Control Register

The CRC_CTL configures the operation modes and settings for the CRC.

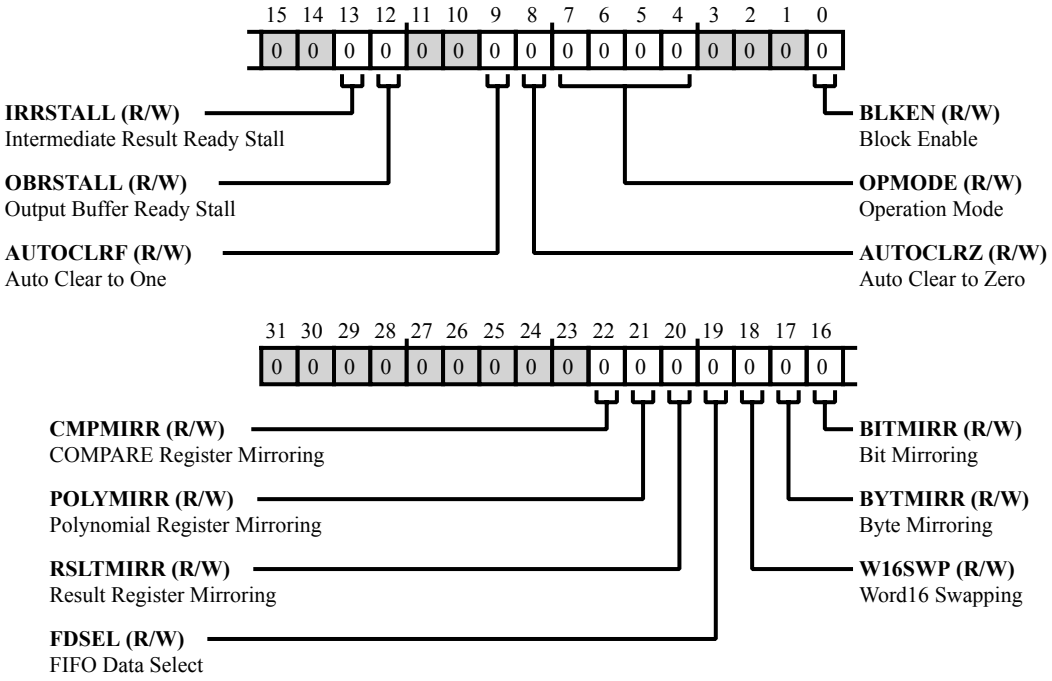

Figure 11-4: CRC_CTL Register Diagram

Table 11-8: CRC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
22 (R/W)	CMPMIRR	COMPARE Register Mirroring. The CRC_CTL.CMPMIRR enables data mirroring for the CRC_COMP compare register. When enabled, the 32-bit value in this register is fully bit mirrored (reversed). The bit-reversed value is used for comparison with the CRC_RESULT_FIN register.	
		0	Disable compare mirroring
		1	Enable compare mirroring
21 (R/W)	POLYMIRR	Polynomial Register Mirroring. The CRC_CTL.POLYMIRR enables data mirroring for the CRC_POLY polynomial register. When enabled, the 32-bit value in this register is fully bit mirrored (reversed). The bit-reversed value is used for CRC computations.	
		0	Disable polynomial mirroring
		1	Enable polynomial mirroring

Table 11-8: CRC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/W)	RSLTMIRR	Result Register Mirroring. The CRC_CTL.RSLTMIRR enables data mirroring for the CRC_RESULT_CUR and CRC_RESULT_FIN result registers. When enabled, the 32-bit values is these registers are fully bit mirrored (reversed).	
		0	Disable result mirroring
		1	Enable result mirroring
19 (R/W)	FDSEL	FIFO Data Select. The CRC_CTL.FDSEL selects whether the CRC writes modified or unmodified data to the FIFO in memory transfer mode. If enabled, the data written is affected by the state of the data mirroring selections (CRC_CTL.BITMIRR, CRC_CTL.BYTMIRR, and CRC_CTL.W16SWP) before being written to the FIFO.	
		0	Write unmodified data to FIFO
		1	Write modified data to FIFO
18 (R/W)	W16SWP	Word16 Swapping. The CRC_CTL.W16SWP enables the CRC's data mirror block to swap the upper and lower 16-bit words within the 32-bit input data, before further processing.	
		0	Disable word16 swapping
		1	Enable word16 swapping
17 (R/W)	BYTMIRR	Byte Mirroring. The CRC_CTL.BYTMIRR enables the CRC's data mirror block to mirror the bytes within the 32-bit input data, before further processing.	
		0	Disable byte mirroring
		1	Enable byte mirroring
16 (R/W)	BITMIRR	Bit Mirroring. The CRC_CTL.BITMIRR enables the CRC's data mirror block to mirror the bits within each byte of the 32-bit input data, before further processing.	
		0	Disable bit mirroring
		1	Enable bit mirroring
13 (R/W)	IRRSTALL	Intermediate Result Ready Stall. The CRC_CTL.IRRSTALL enables stalling the state machine for input data when there is a valid intermediate result to be read in CRC_RESULT_CUR. This feature should be used only in CRC computation modes (for example, CRC_CTL.OPMODE =1 or =3).	
		0	Do not stall
		1	Stall on IRR

Table 11-8: CRC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	OBRSTALL	Output Buffer Ready Stall. The CRC_CTL.OBRSTALL enables stalling the state machine for input data when there is a valid data in the output buffer. This feature should be used only in memory-to-memory transfer modes (for example, CRC_CTL.OPMODE =1).	
		0	Do not stall
		1	Stall on OBR
9 (R/W)	AUTOCLRF	Auto Clear to One. The CRC_CTL.AUTOCLRF enables auto clear to one when the CRC is in intermediate results ready stall mode (CRC_CTL.IRRSTALL=1) and the CRC data count expires (CRC_DCNT=0). Note that CRC_CTL.AUTOCLRZ must be disabled, or the CRC_CTL.AUTOCLRF has no effect.	
		0	No auto clear
		1	Auto clear
8 (R/W)	AUTOCLRZ	Auto Clear to Zero. The CRC_CTL.AUTOCLRZ enables auto clear to zero when the CRC is in intermediate results ready stall mode (CRC_CTL.IRRSTALL=1) and the CRC data count expires (CRC_DCNT=0). Note that CRC_CTL.AUTOCLRF must be disabled, or the CRC_CTL.AUTOCLRZ has no effect.	
		0	No auto clear
		1	Auto clear
7:4 (R/W)	OPMODE	Operation Mode. The CRC_CTL.OPMODE selects the memory transfer or scan mode.	
		0	Reserved
		1	CRC compute/compare memory transfer
		2	Data fill memory transfer
		3	CRC compute/compare memory scan
		4	Data verify memory scan
0 (R/W)	BLKEN	Block Enable. The CRC_CTL.BLKEN enables/disables CRC operation.	
		0	Disable
		1	Enable

Data Word Count Register

The CRC_DCNT holds the word count that is used for the CRC operation. On transfer of every 32-bit word, the CRC decrements by 1 the content of this register. When the count decrements to zero, this event triggers a CRC compare action, and CRC_DCNT is automatically loaded from the CRC_DCNTRLD for the next CRC operation. Note that the initial value programmed into CRC_DCNT may be different from what is programmed in the CRC_DCNTRLD.

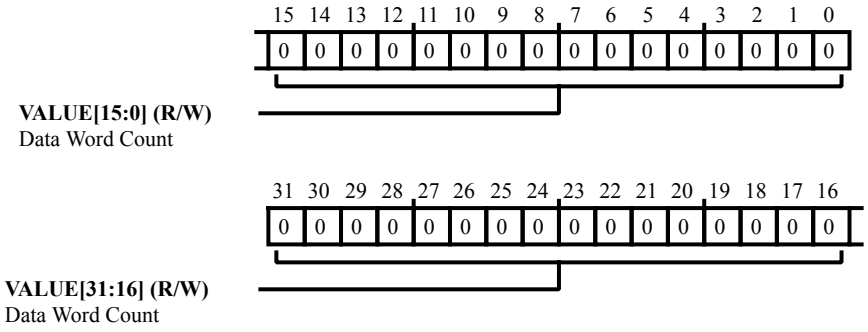

Figure 11-5: CRC_DCNT Register Diagram

Table 11-9: CRC_DCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Data Word Count. The CRC_DCNT.VALUE bit field holds the word count that is used for the CRC operation.

Data Count Capture Register

The CRC_DCNTCAP captures the CRC_DCNT value when a compare operation fails in data verify mode. This capture can be used to track the position of error in the data stream. Capture operation is enabled only if the CRC_STAT.CMPERR indicates no compare error. After an error occurs and data count is captured, no further errors are logged until the CRC_STAT.CMPERR bit is cleared. To obtain the position of error in the data stream, subtract the CRC_DCNTCAP value from the initial CRC_DCNT.

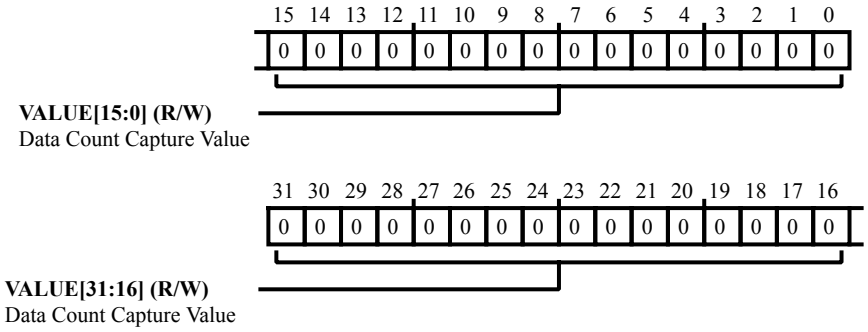

Figure 11-6: CRC_DCNTCAP Register Diagram

Table 11-10: CRC_DCNTCAP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Data Count Capture Value.

Data Word Count Reload Register

The CRC_DCNTRLD holds the value that the CRC automatically loads into CRC_DCNT when the CRC_DCNT decrements to 0. At startup, the value programmed in CRC_DCNT and CRC_DCNTRLD could be different. So, for the first iteration, the CRC operation happens for the count initially programmed in the CRC_DCNT register. While for subsequent CRC operations, the count is taken from the CRC_DCNTRLD register.

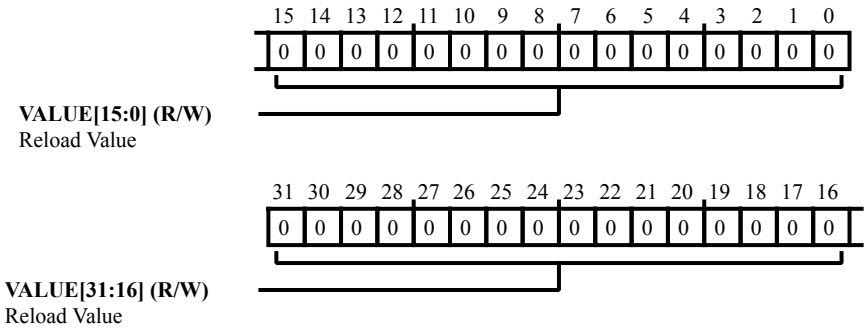

Figure 11-7: CRC_DCNTRLD Register Diagram

Table 11-11: CRC_DCNTRLD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Reload Value. The CRC_DCNTRLD.VALUE bit field holds the value that automatically loads into CRC_DCNT when the CRC_DCNT decrements to 0.

Data FIFO Register

In memory transfer mode (non-data fill mode), the data from the DMA or processor core buses is written into the CRC_DFIFO on each input data grant (DMA grant or core write). Data is read from this FIFO on each output data grant (DMA grant or core read). FIFO status information is available in the CRC_STAT register. Whenever, the FIFO has valid data, output data requests are generated.

Note that---in non-memory transfer mode and in data fill mode---the input data actually does not get written into this FIFO. So, this register should not be read in these modes.

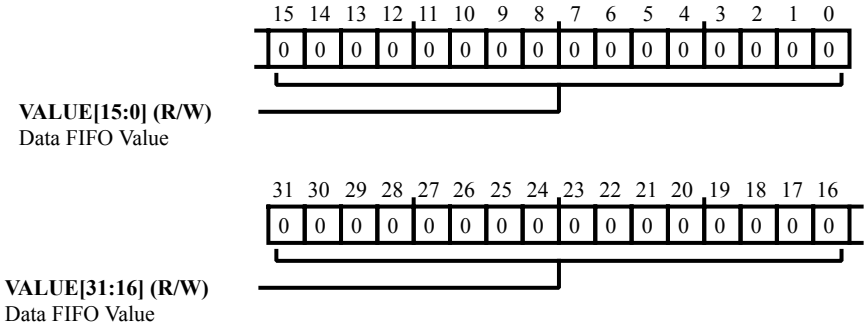

Figure 11-8: CRC_DFIFO Register Diagram

Table 11-12: CRC_DFIFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Data FIFO Value. The CRC_DFIFO.VALUE bit field is the data from the DMA or processor core buses.

Fill Value Register

The CRC_FILLVAL holds the value that the CRC uses for the memory fill operation. In data fill mode, the value programmed in this register is used for the memory fill operation.

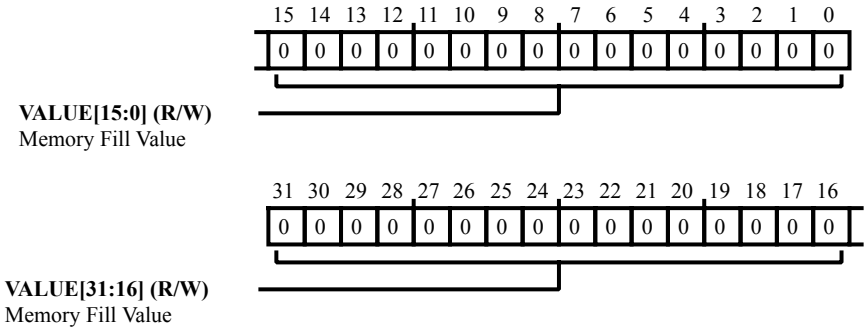

Figure 11-9: CRC_FILLVAL Register Diagram

Table 11-13: CRC_FILLVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Memory Fill Value. The CRC_FILLVAL.VALUE bit field holds the value that the CRC uses for the memory fill operation.

Interrupt Enable Register

The CRC_INEN unmasks (enables) or masks (disables) interrupt requests generated in the CRC from going to the processor core. Note that CRC interrupts are not disabled when the CRC is disabled (CRC_CTL.BLKEN =0).

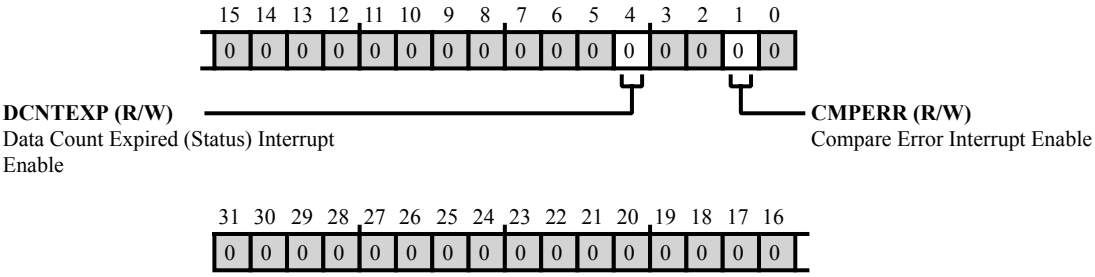

Figure 11-10: CRC_INEN Register Diagram

Table 11-14: CRC_INEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	DCNTEXP	Data Count Expired (Status) Interrupt Enable. The CRC_INEN.DCNTEXP enables (unmasks) the data count expired (CRC status) interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
1 (R/W)	CMPERR	Compare Error Interrupt Enable. The CRC_INEN.CMPERR enables (unmasks) the data compare interrupt, which is generated when CRC data comparison fails.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt

Interrupt Enable Clear Register

The CRC_INEN_CLR permits clearing individual bits in the CRC_INEN register without affecting other bits in the register.

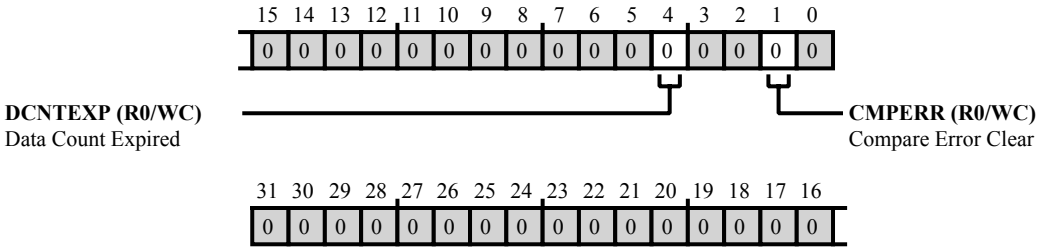

Figure 11-11: CRC_INEN_CLR Register Diagram

Table 11-15: CRC_INEN_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R0/WC)	DCNTEXP	Data Count Expired. The CRC_INEN_CLR.DCNTEXP bit clears the Data Count Expired (Status) interrupt.	
		0	No Effect
		1	Clear Bit
1 (R0/WC)	CMPERR	Compare Error Clear. The CRC_INEN_CLR.CMPERR bit clears the Compare Error interrupt.	
		0	No Effect
		1	Clear Bit

Interrupt Enable Set Register

The CRC_INEN_SET permits setting individual bits in the CRC_INEN register without affecting other bits in the register.

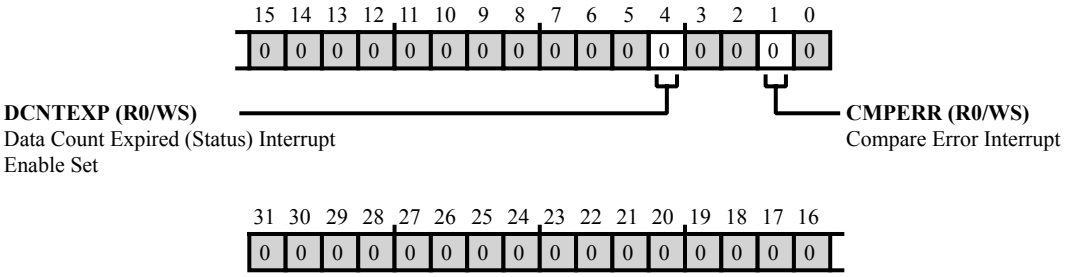

Figure 11-12: CRC_INEN_SET Register Diagram

Table 11-16: CRC_INEN_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R0/WS)	DCNTEXP	Data Count Expired (Status) Interrupt Enable Set. The CRC_INEN_SET.DCNTEXP bit sets the Data Count Expired (Status) interrupt.	
		0	No Effect
		1	Set Bit
1 (R0/WS)	CMPERR	Compare Error Interrupt. The CRC_INEN_SET.CMPERR bit sets the Compare Error interrupt.	
		0	No Effect
		1	Set Bit

Polynomial Register

The CRC_POLY holds a 32-bit polynomial for CRC operations. Bit 31 corresponds to coefficient of x31 of the CRC polynomial, bit 30 corresponds to coefficient of x30, and so on through to bit 0. Coefficient of x32 is assumed to be "1" for any polynomial that is selected. Based on the polynomial in CRC_POLY, the CRC generates a look-up table (LUT), which is used to compute the CRC of the incoming data stream.

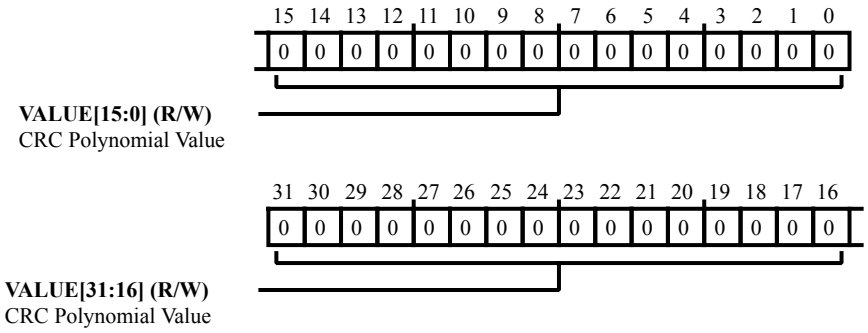

Figure 11-13: CRC_POLY Register Diagram

Table 11-17: CRC_POLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	CRC Polynomial Value. The CRC_POLY.VALUE bit field holds the 32-bit polynomial for CRC operations.

CRC Current Result Register

The CRC_RESULT_CUR holds the current or intermediate CRC result and is updated when new data is written into the CRC. Each time the CRC_DCNT expires, the CRC loads the value from this register into the CRC_RESULT_FIN. The CRC_RESULT_CUR may be set to auto clear to zero or auto clear to ones when CRC_DCNT expires by configuring the CRC_CTL.AUTOCLRZ and CRC_CTL.AUTOCLRF bits. Before starting a CRC operation, the CRC_RESULT_CUR should be programmed to the desired value. Note that this register can be read by the processor core at any time.

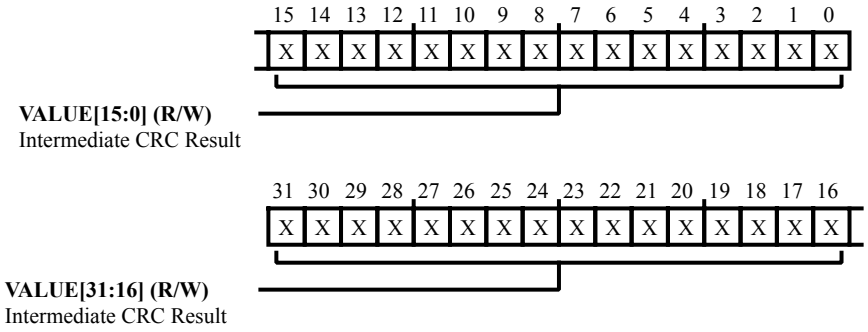

Figure 11-14: CRC_RESULT_CUR Register Diagram

Table 11-18: CRC_RESULT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Intermediate CRC Result. The CRC_RESULT_CUR.VALUE bit field holds the current or intermediate CRC result.

CRC Final Result Register

The CRC_RESULT_FIN holds the final CRC computed for a data stream. A data stream is a DMA of CRC_DCNT number of words into the CRC. When CRC_DCNT decrements to zero for each datastream, the CRC loads CRC_RESULT_FIN with the value from CRC_RESULT_CUR.

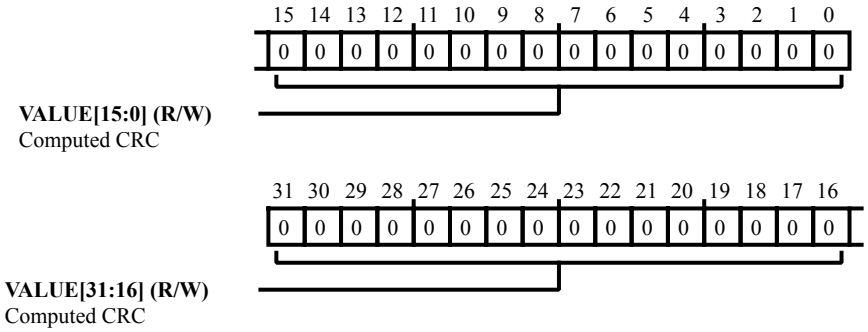

Figure 11-15: CRC_RESULT_FIN Register Diagram

Table 11-19: CRC_RESULT_FIN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Computed CRC. The CRC_RESULT_FIN.VALUE bit field holds the final CRC computed for a data stream.

Status Register

The CRC_STAT indicates status for CRC operations and interrupt generation.

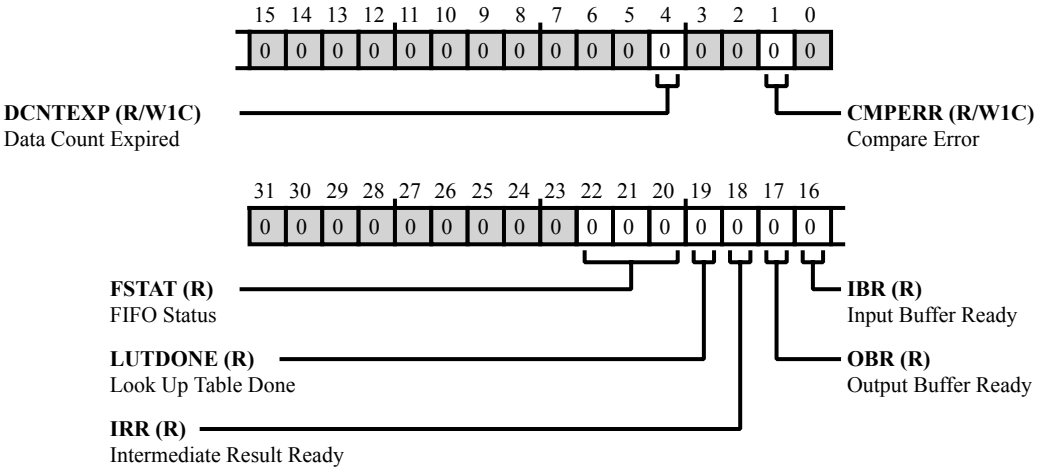

Figure 11-16: CRC_STAT Register Diagram

Table 11-20: CRC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
22:20 (R/NW)	FSTAT	FIFO Status. The CRC_STAT.FSTAT indicates the current FIFO status. This field is read-only.	
		0	FIFO Empty
		1	FIFO has 1 data
		2	FIFO has 2 data
		3	FIFO has 3 data
		4	FIFO has 4 data (Full)
19 (R/NW)	LUTDONE	Look Up Table Done. The CRC_STAT.LUTDONE indicates that the CRC has generated the look up table for the current polynomial. This read-only bit is cleared at reset and cleared when the CRC_POLY is written.	
		0	No Status
		1	LUT Generation Done

Table 11-20: CRC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/NW)	IRR	Intermediate Result Ready. The CRC_STAT.IRR indicates that the CRC has updated the CRC_RESULT_CUR register with intermediate CRC results for the new data written to the CRC. The processor core should read from the CRC_RESULT_CUR register only after detecting CRC_STAT.IRR =1. This read-only bit is cleared by CRC hardware and is valid when CRC_CTL.IRRSTALL is enabled.	
		0	No Status
		1	Intermediate Results Ready
17 (R/NW)	OBR	Output Buffer Ready. The CRC_STAT.OBR indicates that the CRC has data ready for the processor core to read. The processor core should read from the CRC only after detecting CRC_STAT.OBR =1. This read-only bit is cleared by CRC hardware.	
		0	No Status
		1	Output Buffer Ready
16 (R/NW)	IBR	Input Buffer Ready. The CRC_STAT.IBR indicates that the CRC is ready to accept a processor core write. The processor core should write to the input register only after detecting that CRC_STAT.IBR =1. This read-only bit is cleared by CRC hardware.	
		0	No Status
		1	Input Buffer Ready
4 (R/W1C)	DCNTEXP	Data Count Expired. The CRC_STAT.DCNTEXP indicates that the CRC_DCNT has expired. This W1C bit is not automatically cleared when the CRC is disabled (CRC_CTL.BLKEN =0). When the CRC sets this bit on CRC_DCNT expiry, the CRC generates the CRC_INEN.DCNTEXP interrupt.	
		0	No Status
		1	Data Counter Expired
1 (R/W1C)	CMPERR	Compare Error. The CRC_STAT.CMPERR indicates that a CRC mismatch or data mismatch has been detected. This W1C bit is not automatically cleared when the CRC is disabled (CRC_CTL.BLKEN =0). When the CRC sets this bit on detecting a mismatch, the CRC generates the CRC_INEN.CMPERR interrupt. While this bit is set, the CRC_DCNTCAP is disabled from capturing the data count values.	
		0	No Status
		1	Compare Error

12 Direct Memory Access (DMA)

The processor architecture distributes the DMA channels throughout the infrastructure. Often, the channels cluster together through system crossbars (SCB), sharing a single interface with the main system crossbar.

The DMA channels can perform transfers between memory and a peripheral or between one memory and another memory. Memory-to-memory DMA transfers (MDMA) require two DMA channels. One channel is the source channel, and the second, the destination channel.

All DMA channels can transport data to and from virtually all on-chip and off-chip memories.

DMA transfers on the processor use either a descriptor-based method or register-based method. Register-based DMA allows the processor directly to program DMA controller registers to initiate a DMA transfer. On completion, the controller registers can automatically update with their original setup values for continuous transfer, if needed. Descriptor-based DMA transfers require a set of parameters stored within memory to initiate a DMA sequence. Descriptor-based transfers allow the chaining together of multiple DMA sequences. In descriptor-based DMA operations, DMA channel programming can automatically set up and start another DMA transfer after the current sequence completes.

The DMA channel does not connect external memories and devices directly. Rather, data passes through an external-memory interface port. DMA operations can access any device the external memory interface supports. These interfaces typically include:

- flash memory
- SRAM
- FIFOs
- memory-mapped peripheral devices
- Dynamic Memory (if present)

DMA Channel Features

The processor uses Direct Memory Access (DMA) to transfer data within memory spaces or between a memory space and a peripheral. The processor can specify data transfer operations and return to normal processing while the fully integrated DMA channel carries out the data transfers independent of processor activity. The DMA channels are dispersed throughout the infrastructure and interface with the system crossbar unit (SCB).

The following is a list of DMA interface features.

- Supports integer byte strides including byte strides of 0 and negative byte strides
- Register based configuration
  - Core writes DMA configuration
  - Supports automatic reloading for continuous operation
- Flexible descriptor-based configuration
  - DMA descriptors are fetched from memory
  - Support for variable descriptor sizes
- Flexible flow control – Transitions between the various descriptor-based modes and for DMA termination
- Orthogonal transfers
  - Support for three transfer dimensions
  - 1D and 2D transfers supported per descriptor set
  - 3D support provided by chained descriptor sets
- Configurable memory and peripheral-transfer word sizes
  - Memory interface supports 8-bit, 16-bit, 32-bit, 64-bit, 128-bit, and 256-bit transfers
  - Peripheral interface supports for 8, 16, and 32-bit transfers
- Interrupt notification
  - Row or work unit completion
  - Error conditions
- Incoming and outgoing trigger support
  - Trigger generation for row or work unit completion
  - Work unit can wait for incoming trigger
- MMR access bus – Provides access to memory mapped registers for configuration, monitoring, and debug
- SCB crossbar interface connects the DMA channel to the system crossbar
- Peripheral DMA bus – Interfaces the DMA channel to a peripheral or another DMA channel
- Peripheral data-request interrupt support
- Bandwidth monitoring and limiting for MDMA channels

DMA Channel Functional Description

This section provides a functional description of the DMA channel interface.

ADSP-CM40x DMA Register List

The DMA channel controller (DMA) supports data transfers within memory spaces or between a memory space and a peripheral. The processor can specify data transfer operations and return to normal processing while the fully integrated DMA channel carries out the data transfers independent of processor activity. The DMA channels are dispersed throughout the infrastructure, as DMAn's. A set of registers governs DMA operations. For more information on DMA functionality, see the DMA register descriptions.

Table 12-1: ADSP-CM40x DMA Register List

Name	Description
DMA_ADDRSTART	Start Address of Current Buffer Register
DMA_ADDR_CUR	Current Address Register
DMA_BWLCNT	Bandwidth Limit Count Register
DMA_BWLCNT_CUR	Bandwidth Limit Count Current Register
DMA_BWMCNT	Bandwidth Monitor Count Register
DMA_BWMCNT_CUR	Bandwidth Monitor Count Current Register
DMA_CFG	Configuration Register
DMA_DSCPTR_CUR	Current Descriptor Pointer Register
DMA_DSCPTR_NXT	Pointer to Next Initial Descriptor Register
DMA_DSCPTR_PRV	Previous Initial Descriptor Pointer Register
DMA_STAT	Status Register
DMA_XCNT	Inner Loop Count Start Value Register
DMA_XCNT_CUR	Current Count(1D) or intra-row XCNT (2D) Register
DMA_XMOD	Inner Loop Address Increment Register
DMA_YCNT	Outer Loop Count Start Value (2D only) Register
DMA_YCNT_CUR	Current Row Count (2D only) Register
DMA_YMOD	Outer Loop Address Increment (2D only) Register

ADSP-CM40x DMA Channel List

Table 12-2: ADSP-CM40x DMA Channel List

DMA ID	DMA Channel Name	Description
DMA0	SPORT0_A_DMA	SPORT0 Channel A DMA
DMA1	SPORT0_B_DMA	SPORT0 Channel B DMA

Table 12-2: ADSP-CM40x DMA Channel List (Continued)

DMA ID	DMA Channel Name	Description
DMA2	SPI0_TXDMA	SPI0 TX DMA Channel
DMA3	SPI0_RXDMA	SPI0 RX DMA Channel
DMA4	UART0_TXDMA	UART0 TX DMA Channel
DMA5	UART0_RXDMA	UART0 RX DMA Channel
DMA6	SPORT1_A_DMA	SPORT1 Channel A DMA
DMA7	SPORT1_B_DMA	SPORT1 Channel B DMA
DMA8	SPI1_TXDMA	SPI1 TX DMA Channel
DMA9	SPI1_RXDMA	SPI1 RX DMA Channel
DMA10	UART1_TXDMA	UART1 TX DMA Channel
DMA11	UART1_RXDMA	UART1 RX DMA Channel
DMA12	UART2_TXDMA	UART2 TX DMA Channel
DMA13	UART2_RXDMA	UART2 RX DMA Channel
DMA14	HAE0_RXDMA_CH0	HAE0 RX DMA Channel 0
DMA15	HAE0_RXDMA_CH1	HAE0 RX DMA Channel 1
DMA16	HAE0_TXDMA	HAE0 TX DMA Channel
DMA17	MDMA0_SRC	MDMA SRC Channel
DMA18	MDMA0_DST	MDMA DST Channel
DMA19	MDMA1_SRC	MDMA SRC Channel
DMA20	MDMA1_DST	MDMA DST Channel

DMA Definitions

To make the best use of the DMA controller, it is useful to understand the following terms.

Descriptor

An individual configuration fetched from memory that maps to a single register within a DMA channel.

Descriptor Fetch

The action of retrieving descriptors from memory through memory read operations and loading then into the DMA channel registers upon their read return.

Descriptor Set

A group of descriptors associated with a single work unit.

Disabled State

The channel is disabled because the enable bit = 0 or as a result of an error.

DMAC

An acronym used for a DMA cluster.

DMA Channel

A single DMA engine that has all the capabilities and registers as defined for a given processor. A DMA channel or engine is connected to a single peripheral.

DMA Cluster

A grouping of multiple DMA channels with a shared SCB crossbar interface, controller, and arbiter. Also known as a DMAC.

Initial Descriptor

The first descriptor in the descriptor set.

MDMA

Memory-to-Memory DMA Data transfer. Two DMA channels are paired to perform a memory read from one address location and a memory write of that data to another address location.

Stop State

A time where the channel is enabled but not currently programmed to perform a data transfer. Programming the flow to STOP causes the channel to enter Stop State at the end of the work unit.

User

Any person, debug, emulator, software routine, or action taken by the core that accesses the MMR registers of the DMA channel or peripherals, or sets up data and descriptors in memory.

Wait State

If instructed to wait for a trigger, the channel enters this state once it has completed a work unit. The channel remains in this state until a trigger occurs. If a trigger came in before reaching the wait state, the channel skips over the wait state upon completion of the work unit.

Work Unit

A single data transaction or series of data transactions performed based on the configuration of the DMA channel. For autobuffer mode, a new work unit is defined at the time all current count registers are initialized to start values. Once all the current count registers count down to zero, the work unit has completed.

Work Unit Chain

A single work unit or a series or work units separated by a `STOP` or disabled state. The work units in the chain are programmed to another descriptor flow. The last work unit in the chain is programmed to a flow of `STOP` or `AUTO`. `STOP` stops the state at the end of that work unit. `AUTO` is required to be disabled by disabling the DMA channel. A work unit chain is also known as a descriptor chain.

Block Diagram

The *DMA Channel Block Diagram* shows the functional blocks within the DMA interface.

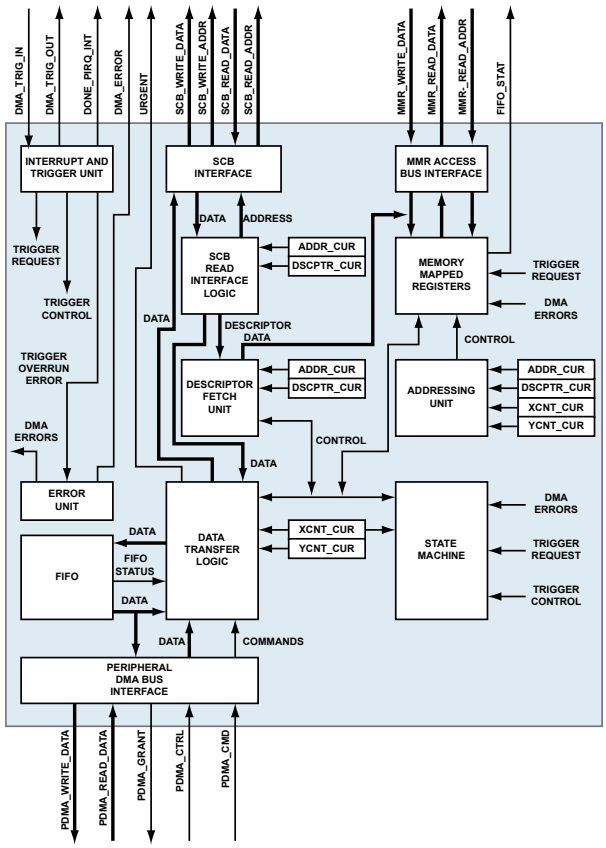

Figure 12-1: DMA Channel Block Diagram

For more information on the interfaces shown in the block diagram, see:

- DMA Channel Peripheral DMA Bus

- DMA Channel MMR Access Bus
- DMA Channel Event Control
- DMA Channel SCB Interface

Architectural Concepts

The DMA channel provides a method to transfer data between memory spaces or between memory and a peripheral using a number of system interfaces. The DMA channel provides an efficient method of distributing data throughout the system, freeing up the processor core for other operations. Each peripheral that supports DMA transfers has its own dedicated DMA channel or channels with its own register set. The register set configures and controls the operating modes of the DMA transfers.

DMA Channel SCB Interface

The SCB interface connects the DMA channel to the SCB crossbar allowing for transfers to and from the processors internal memory and other suitable system resources.

The DMA channel connects to the system interconnect through the SCB interface. This connection lets the DMA channel perform work-unit data transfers with memories such as L1, L2 (internal), and L3 (external). In addition to work unit data transfers, the SCB interface also is used for fetching descriptor sets for all the descriptor-based transfer modes.

The DMA channel can support data bus widths of 16, 32, 64, or 128 bits. The data bus widths for a given DMA channel on a specific processor can vary and are not configurable. Read the `DMA_STAT.MBWID` field to determine the assigned bus widths.

SCB Interface Signals

The DMA channel operates at *SCLK* frequency as does the SCB interface. The SCB crossbar handles the internal arbitration of the transfer requests of all the masters interfaced to the SCB crossbar instance as shown in the ***SCB Interface Signals*** table.

Table 12-3: SCB Interface Signals

Signal	Width (bits)	Description
SCB_WRITE_DATA	16, 32, 64, or 128	Data bus used for write operations. The width of the bus can be determined from `DMA_STAT.MBWID`
SCB_WRITE_ADDRESS	32	Write address bus. Provides the address of the first transfer in a burst transaction
SCB_READ_DATA	16, 32, 64, or 128	Data bus used for read operations. The width of the bus can be determined from `DMA_STAT.MBWID`
SCB_READ_ADDRESS	32	Read address bus. Provides the address of the first transfer in a burst transaction

SCB Burst Transfers

The SCB interface supports burst transfers for memory read and write operations. The burst length is a function of the configurable memory size of the DMA channel for the work unit and the fixed bus width of the SCB data bus of the DMA channel.

- If the DMA channel configuration selects a memory transfer size is less than or equal to the DMA channels bus width, the burst length is always 1.
- If the configured memory size is greater than the SCB interface bus width, the burst length is sufficient to transfer a transaction as specified by the configured memory size.

Table 12-4: DMA Channel SCB Burst Lengths

Configured Memory Size	Burst Length			
	16-bit Bus	*32-bit Bus*	*64-Bit Bus*	*128-bit Bus*
1 Byte	1	1	1	1
2 Bytes	1	1	1	1
4 Bytes	2	1	1	1
8 Bytes	4	2	1	1
16 Bytes	8	4	2	1
32 Bytes	16	8	4	2

Data Address Alignment

To prevent addressing errors and to maximize bandwidth of the SCB interface to the DMA channel, data addresses align with a multiple of the programmable memory size of the DMA channels configuration. These configuration options appear in the Descriptor Set Address Alignment table.

There are situations in which entire work units may not transfer at the maximum configurable memory size. In this case, the entire work unit may transfer by reducing the configured memory size at the expense of bus bandwidth using descriptor sets as follows:

- The first descriptor set can be configured to transfer data until the larger memory size alignments are met.
- A second descriptor set with a larger memory size configuration then can be used to transfer the bulk of the data in the work unit.
- Finally, a third descriptor set can be used with a smaller memory size to complete any final data transfers that cannot meet the alignment requirements of the previous descriptor set configuration.

Table 12-5: DMA Channel Address Alignment Requirements

Configured Memory Size	Address Restriction
1 Byte	No restriction

Table 12-5: DMA Channel Address Alignment Requirements (Continued)

Configured Memory Size	Address Restriction
2 Bytes	ADDR[0] == 0
4 Bytes	ADDR[1:0] == 0
8 Bytes	ADDR[2:0] == 0
16 Bytes	ADDR[3:0] == 0
32 Bytes	ADDR[4:0] == 0

Descriptor Set Address Alignment

All descriptor set addresses and descriptors within a descriptor set must align to a 32-bit address. For descriptor set fetches, the DMA engine ignores the memory-size configuration of the DMA channel. This feature avoids the need to align descriptor sets based on the memory width configuration of the previous descriptor set.

For descriptor sets containing only a single descriptor the transfer takes place as a single 32-bit transfer. For descriptor sets containing multiple descriptors, the DMA engine fetches each 32-bit descriptor individually and treated as multiple 32-bit transfers.

DMA Channel Peripheral DMA Bus

The peripheral DMA bus connects the DMA channel to a peripheral or another DMA channel.

The DMA channel connects to peripherals or other DMA channels through the peripheral DMA bus. This bus is a dedicated point-to-point interface supporting data bus widths of 8, 16, 32, or 64 bits. The data bus widths for a given DMA channel on a particular processor can vary and are not configurable. Reading the `DMA_STAT.PBWID` field permits determining the assigned bus width.

The DMA channel operates at *SCLK* frequency as does the peripheral DMA bus. The following table provides descriptions of the peripheral DMA bus signals.

Table 12-6: Peripheral DMA Bus Signals

Signal	Width (bits)	Description
PDMA_WRITE_DATA	8, 16, 32, or 64	Data bus used for write operations. The width of the bus can be determined from `DMA_STAT.PBWID`
PDMA_READ_DATA	8, 16, 32, or 64	Data bus used for read operations. The width of the bus can be determined from `DMA_STAT.PBWID`
PDMA_DMA_GRANT		Control signals to indicate that data is valid for DMA channel read operations (peripheral transmit). These signals indicate that the DMA channel is ready to receive data for write operations (peripheral receive)
PDMA_CMD	3	Used by the peripheral for issuing DMA channel control commands
PDMA_CTRL		The control signals used by the peripheral to send various commands to the DMA channel and control the direction of flow

Peripheral Control Commands

The peripheral DMA bus of the DMA channel provides a means for peripherals on the processor to issue commands to the DMA channel. These commands provide greater control over the DMA channel operation. This control improves real-time performance and relieves control and interrupt demands on the core. Peripherals can send commands to the DMA controller over the 3-bit PERI_CMD bus. The DMA control commands extend the set of operations available to the peripheral beyond the simple "request data" command used by peripherals in general. Refer to the appropriate peripheral chapter for a description on how that peripheral uses DMA control commands.

These DMA control commands (see the ***PDMA_CMD Peripheral DMA Control Commands*** table) are not visible to or controlled by the program. But, their use by a peripheral has implications for the structure of the DMA transfers that the peripheral can support. It is important to write application software such that it complies with certain restrictions, regarding work units and descriptor chains. Complying with this guideline makes the peripheral operate properly whenever it issues DMA control commands.

The following table describes the commands, which the DMA controller issues. The following sections describe these commands in more detail.

Table 12-7: PDMA_CMD Peripheral DMA Control Commands

Command	Name	Description
b#000	NOP	No operation
b#001	Restart	Restarts the current work unit from the beginning
b#010	Finish	Finishes the current work unit and starts the next
b#011	Interrupt	Immediately sets the DMA completion interrupt in the DMA channel
b#100	Request Data	Typical DMA data request
b#101	Request Data Urgent	Urgent DMA data request
b#110	Reserved	Reserved
b#111	Reserved	Reserved

Idle Command

The DMA channel drives this command when the enabled peripheral has no data requests required.

Restart Command

This command causes the current work unit to interrupt processing and start again, using the addresses and count values from the `DMA_ADDRSTART`, `DMA_XCNT`, and `DMA_YCNT` registers. The DMA controller does not signal an interrupt when the work unit terminates.

If a channel programmed to transmit (memory read) receives a restart command, the channel momentarily pauses, permitting any pending memory reads initiated before the Restart command to complete. During this period, the channel does not grant DMA requests. After all pending reads flush from the pipelines of the channel, the channel resets its counters and FIFO, then starts pre-fetch reads from memory. The DMA controller grants data requests

from the peripheral as soon as new prefetched data is available in the DMA FIFO. In this case, the peripheral can use the Restart command to reattempt a failed transmission of a work unit.

If a channel programmed to receive (memory write) receives a restart command, the channel stops writing to memory, discards any data held in its DMA FIFO, and resets its counters and FIFO. As soon as this initialization is complete, the channel again grants DMA write requests from the peripheral. In this case, the peripheral can use the restart command to abort the transfer of received data into a work unit, and reuse the memory buffer for a later data transfer.

The request from the restart control command is not granted or acknowledged. The DMA controller always accepts the request.

Finish Command

The finish command causes the current work unit to terminate processing and move on to the next work unit. If enabled within the DMA_CFG register, the DMA channel signals an interrupt or a trigger event. The peripheral can then use the finish command to partition the DMA stream into work units on its own. This partitioning occurs---perhaps as a result of parsing the data currently passing though its supported communication channel---without direct real-time control by the processor.

For a DMA channel programmed to transmit (memory read) operation and receives a finish command, the channel momentarily pauses for the completion of any pending memory reads, which were initiated prior to the finish command. During this time, the channel does not grant DMA requests. After the flush of all pending reads from the pipelines of the channel, the channel signals an interrupt or a trigger (if enabled) and begins fetching the next descriptor (if any). DMA data requests from the peripheral are granted as soon as new prefetched data is available in the DMA FIFO.

If a channel programmed to receive (memory write) receives a finish command, the channel stops granting new DMA requests while it drains its FIFO. The channel writes to memory any DMA data received by the DMA channel prior to the finish command. When the FIFO reaches an empty state, the channel signals an interrupt or a trigger (if enabled) and begins fetching the next descriptor (if any). After fetching the next descriptor, the channel initializes its FIFO, then resumes granting DMA requests from the peripheral.

The finish command request is not granted or acknowledged. The request is always accepted by the DMA channel.

Interrupt Command

The interrupt command causes the DMA channel to generate an interrupt. When programming the channel to support this command, configure the DMA_CFG.INT bit field to PIRQ mode. This configuration directs the channel not to generate interrupts based on work unit state. Instead, the channel generates interrupts only when it receives the interrupt command from the peripheral. When the channel receives an interrupt command, the DMA_STAT.PIRQ bit indicates the event under the following conditions:

- The DMA_CFG.EN bit enables the DMA channel.
- The DMA channel is in the stop state.
- The interrupt in DMA_CFG.INT is configured for PIRQ mode.

The peripheral only issues the interrupt command in response to receiving the last grant command from the DMA channel, indicating that the transfer is the last transfer in the work unit.

Request-Data Command

The request data command is a request for data transfers between the DMA channel and the peripheral. The request is held by the peripheral until granted or acknowledged by the DMA channel.

Request-Data Urgent Command

The request-data urgent command behaves identically to the request data command, except that---during the commands assertion---the DMA channel performs its memory accesses with urgent priority. This priority includes both data and descriptor fetch memory accesses. For example, a DMA management capable peripheral can use this control command if an internal FIFO approaches a critical condition.

The request is held by the peripheral until granted or acknowledged by the DMA channel.

Peripheral-Control Command Restrictions

The proper operation of the DMA channel FIFO leads to certain restrictions in the sequence of DMA peripheral control commands issued by a peripheral. The following sections describe these restrictions.

Transmit-Restart or Transmit-Finish Command

A peripheral only can issue a restart or finish control command to a channel configured for memory read under the following conditions:

- The peripheral has already performed at least one DMA transfer in the current work unit.
- The current work unit has (`FIFO_SIZE`/`DMA_CFG.MSIZE`) + 1 memory transfers remaining.

The first item ensures that the work unit has started. The second item ensures that the work unit has not completed. The second item is sufficiently large that it is always at least five more than the maximum data count before any restart or finish command. If using restart or finish commands to manage a work unit, this requirement implies that the work unit must have `DMA_XCNT_CUR` and `DMA_YCNT_CUR` register values representing at least five data items.

To satisfy the second item, ensure that the number of memory transfers described by the descriptor is (`FIFO_SIZE`/`DMA_CFG.MSIZE`) + 1 larger than the maximum number of memory transfers expected.

Receive-Restart or Receive-Finish Commands

A peripheral only can issue a restart or finish control command to a channel configured for memory write under the following conditions:

- The number of peripheral transfers completed is less than (`DMA_CFG.MSIZE`/`DMA_CFG.PSIZE`) × (transfers described by descriptor)
- In addition to the previous condition, one of the following conditions also must apply:

- A finish command terminated the previous work unit, *and* the peripheral has done at least one transfer in the current work unit.
- The peripheral has done (`FIFO_SIZE`/`DMA_CFG.PSIZE`) + 1 transfers in the current work unit.

The first condition ensures that the descriptor is still active. The second set of conditions ensures that data from the previous descriptor has left the FIFO and that the current descriptor has started.

Memory DMA and Triggering

A memory DMA (MDMA) channel provides a means of doing memory-to-memory DMA transfers among the various memory spaces that have DMA support.

The DMA controller implements memory DMA (MDMA) channels by interfacing two DMA channels through the peripheral DMA bus interface. One DMA channel serves for memory read operations, and the second channel servers for memory writes. Depending on the processor, a memory DMA channel can have an additional peripheral, such as a CRC peripheral. The additional peripheral is inserted into the peripheral DMA bus that optionally can be enabled.

MDMA channel configurations that do not involve an additional peripheral impose no restrictions on which of the DMA channels is to use for the read operation or the write operation. But, the configuration of both channels cannot have the same transfer direction. For MDMA channel configurations that enable a peripheral between the read and write channels, be aware of possible restrictions imposed on which channel can use a given transfer direction.

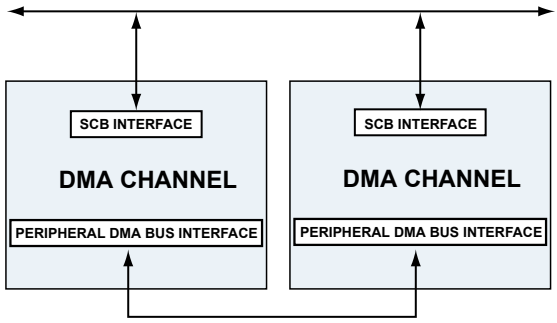

Figure 12-2: MDMA Channel Dedicated Pair

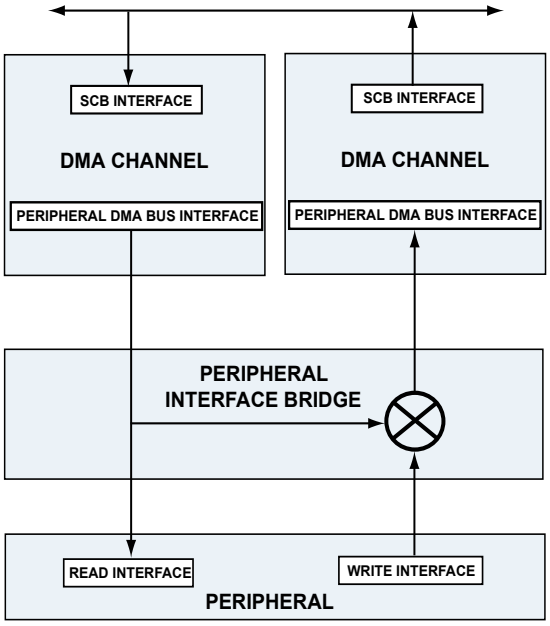

Figure 12-3: MDMA Channel Pair with Peripheral

A memory-to-memory transfer always requires enabled source and destination channels. Because the channels interface through the peripheral DMA bus and can have an additional peripheral inserted into the peripheral DMA bus, programs must make sure to set the same values in the `DMA_CFG.PSIZE` of both the source and destination channels.

The memory DMA channels support the full range of `DMA_CFG.MSIZE` options for the DMA transfers to and from the memories.

As the MDMA channel consists of two DMA channels, the entire MDMA channel has two sets of FIFOs, one in the read channel and one in the write channel. This FIFO usage allows for more efficient bursting of both read and write transactions to use the available bandwidth. While the `DMA_CFG.PSIZE` configuration must be identical for both source and destination DMA channels, this restriction does not apply for the `DMA_CFG.MSIZE` configuration.

Configure the `DMA_CFG.PSIZE` bits to a value no larger than the supported bus width of the peripheral DMA bus.

The independent source and destination DMA channels also have their own dedicated interrupt and trigger events. While it is normal practice to have only event generation performed at destination DMA completion, programs also can use other means of interrupt generation.

Configuration of an MDMA transfer is done in a similar manner to peripheral DMA transfers, except for writing two DMA channel registers instead of one.

To control the pace of data transfers, use triggers on either the memory read or the memory write channel pair used in an MDMA operation. Enabling `DMA_CFG.TWAIT` in the memory read channel prevents both channels from transferring data before the system is ready. However, only configuring the memory write channel to wait for a trigger allows for data fetch from the memory in anticipation of the memory write operation.

DMA Channel MMR Access Bus

The MMR access bus provides access to all the DMA channels memory-mapped registers for DMA channel configuration, monitoring, and debug. The interface has a fixed 32-bit data bus for read and write accesses.

The ***MMR Access Bus Signals*** table provides descriptions of the MMR access bus signals.

Table 12-8: MMR Access Bus Signals

Signal	Width (bits)	Description
MMR_WRITE_DATA	32	Data bus used for write operations to the MMRs from the core.
MMR_READ_DATA	32	Data bus used to return read data from the MMRs
MMR_READ_ADDR	7	Address used to select the MMR to access

DMA Channel Operation Flow

A detailed description of the flow of operation of the DMA channel appears in the following topics:

- Startup Flow
- Refresh Flow
- DMA Operating Modes
- Stop Mode
- DMA Channel Errors

Startup Flow

Enabling a DMA operation on a given channel first requires directly writing some or all of the DMA parameter registers. The minimum set of register required to be initialized depends on the desired mode of operation as described in the following sections.

Startup Minimum-Enable Requirements

To start a DMA operation on a given channel, some or all of the DMA parameter registers must first be initialized and configured to the DMA channels desired operating mode.

- For descriptor array-based flow modes, at minimum, write the `DMA_DSCPTR_CUR` register prior to writing to the `DMA_CFG` register, which is the special action required to start the DMA channel.
- For descriptor list-based flow modes, at minimum, write the `DMA_DSCPTR_NXT` register prior to writing to the `DMA_CFG` register, which is the special action required to start the DMA channel.
- For non-descriptor-based flow modes, write the `DMA_ADDRSTART`, `DMA_XCNT`, and `DMA_XMOD` registers prior to writing the `DMA_CFG` register.

Programs can write other registers that can remain static throughout the course of the DMA activity. The write to the `DMA_CFG` register begins the DMA operation.

ATTENTION: When software directly writes the DMA_CFG register, the DMA controller recognizes this action as the special startup condition. This condition occurs when starting DMA for the first time on this channel or occurs after the DMA channel stops. It is possible for the channel to flag a DMA error condition regardless of the DMA_CFG.EN bit setting.

Startup Operation

When software directly writes the DMA_CFG register, the DMA channel recognizes this operation as the special start-up condition. This condition occurs when starting DMA for the first time on this channel or occurs after the channel has entered to the stop state.

When the descriptor fetch is complete and the DMA channel is enabled, the DMA_CFG descriptor element in the DMA_CFG register assumes control. Before this point, the direct write to the DMA_CFG register had control.

At startup, the selected flow mode and the descriptor size determine the course of the DMA initialization process. The DMA_CFG.FLOW field determines whether to load more current registers from descriptor sets in memory. The DMA_CFG.NDSIZE field details how many descriptor elements to fetch before starting the DMA operation. This process does not affect DMA registers that are not in the descriptor; no modifications are made to their prior values.

For descriptor-list flow modes, the channel copies the DMA_DSCPTR_NXT register value into the DMA_DSCPTR_CUR register. Then, the channel fetches of new descriptor elements from memory. The DMA_DSCPTR_CUR register indexes each fetch, and the channel increments the index after each fetch. After completion of the descriptor fetch, the DMA_DSCPTR_CUR register points to the next 32-bit word in memory past the end of the descriptor.

If the descriptor fetch is for a descriptor-array mode transfer, the channel does *not* copy the DMA_DSCPTR_NXT register into the DMA_DSCPTR_CUR register. *Instead*, the descriptor fetch indexing begins with the value in the DMA_DSCPTR_CUR register.

If DMA_CFG is not part of the fetched descriptor set, the previous value (originally as written on startup) controls the work unit operation. If the DMA_CFG register is part of the fetched descriptor set, the value programmed by the MMR access controls only the loading of the first descriptor fetched from memory. The configuration of the DMA_CFG register controls the subsequent DMA work units of the fetched descriptor set.

After the descriptor fetch is complete or if the flow configuration was originally for one of the register-based flow modes, the DMA operation begins. The DMA channel immediately fills its FIFO. For a memory-write operation, the DMA channel begins accepting data from the peripheral. For a memory-read operation, the DMA channel begins memory reads when the SCB bus grants access to the DMA channel.

When the DMA channel performs its first data-memory access, its address and count computations take their input operands from the start registers. These registers can include DMA_ADDRSTART, DMA_XCNT, and DMA_YCNT, if necessary. The channel writes results back to the current registers. These registers include DMA_ADDR_CUR, DMA_XCNT_CUR, and DMA_YCNT_CUR. Note also that the current registers are not valid until the channel performs the first memory access, which can be some time after the write to the DMA_CFG register starts the channel. Once started, the channel automatically loads the current registers from the appropriate descriptor elements, overwriting their previous contents. These automatic-load operations include:

- The channel copies the DMA_ADDRSTART value to DMA_ADDR_CUR.

- The channel copies the DMA_XCNT value to DMA_XCNT_CUR.
- The channel copies the DMA_YCNT to DMA_YCNT_CUR.

Refresh Flow

When the channel completes processing of a work unit, the DMA channel performs the following operations:

- Completes the transfer of all data between memory and the DMA channel
- Performs a synchronized transition (if the DMA channel configuration is a memory read operation with the DMA_CFG.SYNC bit enabled) *and* transfers all data to the peripheral before continuing
- Forwards the signals from the DMA channel (if interrupts or triggers are enabled) *and* updates the DMA_STAT register to indicate the interrupt or trigger events
- Clears the DMA_STAT.RUN bit field to stop DMA operation (if the flow was set to stop mode) *and* transfers any remaining data in the FIFO of the DMA channel to the peripheral
- Loads a new descriptor from memory into the DMA registers by way of the contents of the DMA_DSCPTR_CUR register (for descriptor-array mode) *and* increments the DMA_DSCPTR_CUR register

  The channel takes the descriptor size from the DMA_CFG.NDSIZE value before the fetch.
- Copies the DMA_DSCPTR_NXT register into the DMA_DSCPTR_CUR register (for descriptor-list mode), fetches the descriptor from the new contents of the DMA_DSCPTR_CUR register, *and* places these contents into the DMA registers while incrementing the DMA_DSCPTR_CUR register
- Checks for detection of an incoming trigger event (for descriptor-on-demand array mode):
  - If the channel detects a trigger event, the DMA channel loads a new descriptor from memory into the DMA registers from the contents of the DMA_DSCPTR_CUR register, while incrementing the DMA_DSCPTR_CUR register. The channel takes the descriptor size from the DMA_CFG.NDSIZE value before the fetch.
  - If the channel detects no trigger event, the DMA channel begins the next work unit by reloading the current registers.
- Checks for detection of an incoming trigger event (for descriptor-on-demand list mode):
  - If the channel detects a trigger event, the DMA channel copies the DMA_DSCPTR_NXT register value to the DMA_DSCPTR_CUR register, fetches the descriptor memory from the DMA_DSCPTR_CUR register, *and* places the contents into the DMA registers while incrementing the DMA_DSCPTR_CUR register.
  - If the channel detects no trigger event, the DMA channel begins the next work unit by reloading the current registers as described in the next step.
- Begins the next work unit (if flow configuration is anything other than stop mode) by reloading the current registers (DMA_ADDR_CUR, DMA_XCNT_CUR, and DMA_YCNT_CUR) from their descriptor registers (DMA_ADDRSTART, DMA_XCNT, and DMA_YCNT)

Work Unit Transition Flow

The DMA_CFG.SYNC bit controls transitions from one work unit to the next work unit. In general, continuous transitions have lower latency at the cost of restrictions on changes of data format or addressed memory space in the two work units. These latency gains and data restrictions arise from the way the channel handles the DMA FIFO while fetching the next descriptor.

In continuous transitions, with disabled synchronization, the DMA FIFO pipeline continues to transfer data to and from the peripheral or destination memory. These transfers continue during the descriptor fetch and during the DMA channel pause between descriptor chains. By comparison, synchronized transitions provide better real-time synchronization of interrupts and triggers with a given peripheral state. Synchronized transitions also provide greater flexibility in the data formats and memory spaces of the two work units. This flexibility comes at the cost of higher latency in the transition. In synchronized transitions, the DMA FIFO pipeline drains to the destination or flushes (received data discarded) between work units.

NOTE: The DMA_CFG.SYNC bit of the MDMA source channel controls work unit transitions for MDMA streams. Clear this reserved bit of the MDMA destination channel, placing it in the disabled state. In transmit (memory read) channels, the DMA_CFG.SYNC bit of the last descriptor before the transition controls the transition behavior. In contrast, in receive channels, the DMA_CFG.SYNC bit of the first descriptor of the next descriptor chain controls the transition.

Work Unit Transmit and MDMA Source Transitions

In DMA transmit (memory read) and MDMA source channels, the DMA_CFG.SYNC bit controls the interrupt timing at the end of the work unit. This bit also controls the handling of the DMA FIFO between the current and the next work unit.

If the DMA_CFG.SYNC bit configuration disables synchronization, the DMA channel operates in continuous transition. In a continuous transition, just after reading the last data item from memory, the DMA channel starts all of the following operations parallel:

- Signals the interrupt or trigger
- Updates the DMA_STAT register to indicate DMA completion status
- Begins fetching the next descriptor
- Delivers the final data items from the DMA FIFO to the destination memory or peripheral

This process lets the DMA channel provide data from the FIFO to the peripheral continuously during the descriptor fetch latency period.

If the configuration disables synchronization, the final interrupt or trigger (if enabled) occurs when the channel reads the last data from memory. This event occurs at the earliest time that the channel safely can modify the output memory buffer without affecting the previous data transmission. There can be a number of data items remaining in the FIFO and not yet at the peripheral. This number depends on the FIFO depth of the DMA channel. In this configuration, do not use the DMA interrupt as the sole means of synchronizing the shutdown or re configuration of the peripheral following a transmission.

NOTE: If the configuration selects continuous transition on a transmit (memory read) descriptor, the next descriptor must have the same:

- Peripheral transfer size (DMA_CFG.PSIZE)
- Read or write direction
- Source memory (internal versus external) as the current descriptor

It is possible to disable synchronization by selecting continuous transition on a work unit with configuration for stop-flow mode and with enabled interrupts or triggers. This approach can result in execution of the event service routine while draining of the final data is ongoing from the FIFO to the peripheral. If data transfers are in progress, the FIFO is not yet empty. The DMA_STAT.RUN bits of the DMA channels indicate this status. Do not start a new work unit with different peripheral transfer size or direction while data transfers are in progress.

CAUTION: Disabling the channel with the DMA_CFG.EN bit while data transfers are in progress causes the loss of the data in the FIFO.

A synchronized transition configuration directs the channel to drain the DMA FIFO to the destination memory or peripheral. This FIFO operation occurs before the channel signals any interrupt and before the channel fetches any subsequent descriptor or data. This operation incurs greater latency, but provides direct synchronization between the DMA interrupt and the state of the data at the peripheral.

If the configuration enables synchronization and enables interrupts, on the last descriptor in a work unit, the interrupt occurs when the channel transfers the final data to the peripheral. This event allows the service routine to switch properly to non-DMA transmit operation. When the event vectors to the interrupt service routine, the DMA channel FIFO is empty, and the DMA channel is no longer running (indicated by the DMA_STAT.RUN bits).

A synchronized transition also allows greater flexibility in the format of the DMA descriptor chain. When enabled, the next descriptor can have any DMA_CFG.PSIZE configuration or read/write direction supported by the peripheral and can come from either memory space (internal as opposed to external). This feature can be useful in managing MDMA work unit queues, since it is no longer necessary to interrupt the queue between dissimilar work units.

Work Unit Receive and MDMA Destination Transitions

In DMA receive channels (memory write operations), the DMA_CFG.SYNC bit controls the handling of the DMA FIFO between descriptor chains (not individual descriptor sets), during the DMA channel pause. The DMA channel pauses after descriptor sets configured with stop flow mode complete. Restart the channel (for example, after an interrupt) by writing the DMA_CFG register of the channel with a value that enables the DMA channel. If the configuration disables synchronization in the DMA_CFG value of the new work unit, the configuration selects a continuous transition. In this mode, the DMA FIFO retains any data items received during the channel pause, and they are the first items written to memory in the new work unit. This mode of operation provides lower latency at work unit transitions and ensures no dropping of data items during a DMA pause. The channel provides this operation at the cost of certain restrictions on the DMA descriptors.

NOTE: If the DMA_CFG.SYNC bit disables synchronization on the first descriptor of a chain after a DMA pause, do not change the configuration of the DMA_CFG.PSIZE field of the new chain from the previous

descriptor chain (active before the pause). This restriction applies unless the DMA channel is reset between chains by disabling and then re-enabling the DMA channel.

If the DMA_CFG.SYNC bit configuration enables synchronization, the channel uses a synchronized transition. In this mode, only the data that the DMA channel receives from the peripheral after the write to the DMA_CFG register gets to memory. The channel discards any prior data items transferred from the peripheral to the DMA FIFO before this register write occurs. This operation provides direct synchronization between the data stream received from the peripheral and the timing of the channel restart, which occurs on the write to the DMA_CFG register.

For receive DMA operations, the synchronization has no effect in transitions between work units in the same descriptor chain. When the flow mode of previous descriptor was not stop, the DMA channel did not pause.

If a descriptor chain begins with synchronization enabled, there is no restriction on the DMA_CFG.PSIZE of the new chain in comparison to the previous chain.

NOTE: The peripheral transfer size (DMA_CFG.PSIZE) must not change between one descriptor and the next in any DMA receive (memory write) channel within a single descriptor chain, regardless of the DMA_CFG.SYNC bit setting. In other words, all memory write descriptor sets in a descriptor chain must have the same DMA_CFG.PSIZE value. For any DMA receive channel (memory write operation), there is no restriction on changes of peripheral transfer size (internal versus external) between descriptors or descriptor chains.

Transfer Termination and Shutdown Flow

This section describes channel transfer termination and shutdown in stop flow mode and in autobuffer flow mode.

Stop Flow Mode

In stop flow mode, the DMA channel stops automatically after the work unit is complete. If using a list or array of descriptors to control DMA transfers and if every descriptor contains a DMA_CFG descriptor element, configure the flow of the final DMA_CFG descriptor element to stop mode, stopping the channel gracefully. After completion, the DMA channel remains in the stop state. Do not confuse this state with the disabled state, which either occurs due to a DMA error or occurs through disabling the DMA channel by configuring the DMA_CFG.EN bit.

The intention of disabling the DMA channel through a write to the DMA_CFG.EN bit is to shut down the DMA channel and to enter the disabled state. All memory and peripheral data transfers cease, and only peripheral interrupts pass through the DMA channels interrupt signals. However, the DMA channel maintains the DMA_STAT.RUN bits. For a write to memory, the outstanding memory transaction counter tracks returning memory write acknowledgments and updates as required.

For memory reads, the outstanding memory-transaction count also tracks returning memory reads. The channel does not write the memory reads into the FIFO. The channel updates the counter to reflect the completion of the transaction, but the channel ignores the data. The DMA_STAT.RUN bits remain in the *waiting for write ACK or FIFO drain to peripheral* state and do not change to *stop or idle state* until the return of all outstanding transactions.

When the DMA_CFG.EN bit again enables the DMA channel, the channel performs a full reset and clears all counters. If an outstanding memory transaction returns an acknowledgment or read data after this event, a memory

transaction error occurred, which generates an error event. Programs must ensure that all outstanding memory transactions complete before reconfiguring the DMA channel. One method programs can use is to poll the DMA_STAT.RUN bits to return to the stop or idle state before proceeding.

Autobuffer Flow Mode

In this mode, the flow does not use any descriptors in stored memory. Instead, the channel performs DMA in a continuous circular buffer fashion, based on user programmed DMA register settings. On completion of the work unit, the channel reloads the parameter registers into the current registers, and DMA resumes immediately with zero overhead. Consider this mode as a succession of automatically restarted work units.

For auto-buffer-flow modes, the only way to cease operations is to disable the DMA channel through the DMA_CFG.EN bit. One method of changing to a new work unit would be to:

- Disable the DMA channel
- Set up all the registers (and descriptors in memory, if used) except for DMA_CFG
- Poll DMA_STAT.RUN to wait for the status to reflect stop or idle state, and
- Write DMA_CFG to the new configuration to begin the next work unit

In auto-buffer-flow mode or for a list or array of descriptor sets without DMA_CFG descriptors, use an MMR write to the DMA_CFG register to terminate the DMA transfer process. Configure the value of the DMA_CFG.EN bit in this register to disable the DMA channel.

CAUTION: When the configuration disables a DMA channel, the DMA controller disables interrupt logic that is based on work unit transitions. Be aware of the system environment and current actions, so that additional interrupts are not required from the DMA channel.

CAUTION: If disabled through DMA_CFG.EN in the middle of a transaction, the DMA channel completes any transactions that have begun and avoids generating bus errors. However, the channel considers the action of re-enabling the DMA as a hard reset for all internal DMA channel components. Therefore, pay special attention to that particular action to avoid unexpected results.

DMA Channel Errors

When an error occurs, the DMA channel maintains all the state and register values which allows programs to diagnose error causes more thoroughly. The greatest benefit to the programmer is to know exactly what operational state the DMA channel was in at the exact moment the error occurred.

Take special care to address the root cause of the error, whether or not the problem originated in the DMA channel. If not properly resolved, the error could result in an additional error shortly after operations resume. The problem can cause other errors elsewhere in the DMA channel or associated modules and circuitry, so take care also to address those potential problems. Ensure that all outstanding memory reads and writes are complete or cleared, before resuming DMA channel operation.

After addressing all issues and neutralizing all side effects of any errors, clear the DMA_STAT.ERRC status field and restart the DMA channel by disabling then re-enabling the DMA channel through the DMA_CFG.EN bit.

The following sections describe the error types.

Status and Debug

DMA channel error conditions can cause the DMA process to end abnormally. The DMA channel provides error detection as a tool for system development and debug, helping to identify DMA-related programming errors. When the DMA channel detects an error, the channel immediately stops and discards any returned memory-read transactions. The DMA_STAT.RUN field of the DMA channel indicates the idle state after acknowledging all outstanding memory transactions. In addition, the channel asserts an error interrupt and updates theDMA_STAT.IRQERR. Also the channel updates the DMA_STAT.ERRC field, indicating the error cause of the first detected error. Unless the error occurs at the exact moment that modification of register values occurs, the registers contain the error values.

All the DMA error interrupts are combined into a single shared interrupt. Combined error signals require reading the DMA_STAT register of each DMA channel associated with a combined error interrupt to determine the DMA channel responsible for the generation of the interrupt.

The DMA channel error interrupt handler performs the following actions:

- Read the DMA_STAT register of each DMA channel, seeking a channel with the DMA_STAT.IRQERR set to indicate an error.
- Read the DMA_STAT.ERRC field of each DMA channel, determining the cause of the error.
- Clear the problem with the DMA channel. For example, fix the register values.
- Clear the error in the DMA channel through a write-1-to-clear operation to the DMA_STAT.IRQERR bit.

If the channel flags any uncleared error other than a bandwidth monitor error, the channel reports no other error. If the channel reports an uncleared bandwidth monitor error, the channel reports any newly detected error through updating the DMA_STAT.ERRC field.

DMA Configuration Register Errors

The channel only flags these configuration errors when the DMA_CFG.EN bit enables the DMA channel. Error flagging occurs when the configuration:

- Uses a reserved setting
- Enables DMA_CFG.TWAIT in descriptor on-demand flow mode
- Uses an illegal DMA_CFG.NDSIZE
- Uses an illegal DMA_CFG.MSIZE
- Configures DMA_XCNT = 0. Or when DMA_YCNT = 0 in 2D DMA mode
- Uses non-zero value in DMA_CFG.NDSIZE when DMA is configured in stop mode or auto mode

- Enables interrupt or outgoing triggers on DMA_YCNT when DMA is configured in 1D mode
- Use a DMA_CFG.MSIZE that exceeds the FIFO size of the DMA channel
- Uses an illegal DMA_CFG.PSIZE
- Uses a DMA_CFG.PSIZE that exceeds the FIFO size
- Uses a DMA_CFG.PSIZE that exceeds the bus width
- Attempts to change from a transmit operation (memory read) to receive operation without properly synching in the previous work unit or if first work unit in a new chain
- Attempts to change DMA_CFG.PSIZE of a transmit operation (memory read) without properly synching in previous work unit or if first work unit in a new chain
- Attempts to change from receive operation (memory write) to transmit during a descriptor chain.

  The channel only can change from receive to transmit if the new transmit is synchronized and is the first work unit.
- Attempts to change DMA_CFG.PSIZE of a receive operation (memory write) when the operation was not the first work unit (with DMA_CFG.SYNC enabled)

Illegal Register Write During Run

The channel generates an error when a write occurs to writable registers of an enabled, running DMA channel. The channel blocks the write. The DMA_STAT, DMA_BWLCNT, and DMA_BWMCNT registers are exempt from this behavior.

Address Alignment Error

The channel generates an address alignment error when any of the following apply:

- Alignment of a descriptor address is not on a 32-bit boundary.
- The current DMA_CFG.MSIZE configuration contains an unaligned transfer address. DMA_ADDRSTART is not aligned according to DMA_CFG.MSIZE field.

Memory Access Error

The channel generates a memory access error when DMA process:

- attempts to access an unpopulated address,
- attempts to access an address defined as cache, or
- attempts to access a location that provokes a security violation.

The error returned from the memory triggers the memory access error.

Trigger Overrun Error

A trigger overrun error is generated when a new trigger input occurred while an outstanding trigger is waiting. This error is only generated if `DMA_CFG.TOVEN` is enabled.

Bandwidth-Monitor Error

The channel generates this error when the bandwidth-monitor count expires. This error is not fatal, and the DMA channel continues operation.

Control Interface Error

The channel reports control-interface errors as bus errors to the bus master. This error can result from:

- An address error
- A register write error (write to a read-only register)

DMA Operating Modes

The DMA channel supports a number of different flow modes that control how the DMA channel progresses from one work unit to the next.

The flow mode of a DMA channel is not a global setting. A DMA descriptor set can include the descriptor responsible for configuring the flow of the work unit. There is no restriction, limiting the flow configuration to be the same for the entire descriptor chain. If the descriptor chain is not endless, the last descriptor set configures the flow to stop mode, which results in termination of the descriptor chain after work unit completion. Another example for mixing flow modes is to create an endless descriptor array. The configuration of the last descriptor set in the array selects list mode. The next descriptor pointer in this set of descriptors points to the first descriptor in the array.

Register Based Flow Modes

Register-based DMA operations require configuration by directly writing to the memory-mapped registers of the DMA channel.

Register-based DMA is the traditional method of DMA operation. Software writes all of the configuration of the DMA channel into the memory-mapped registers. This configuration includes information such as the source or destination address and length of the data in the transfer. The DMA controller then starts channel operation. The DMA channel supports the following register-based flow modes.

- Stop Mode
- Autobuffer Mode

The DMA channel supports variable descriptor set sizes within the configuration. The size of a descriptor set can contain as little as a single descriptor. The supported descriptor set sizes can differ between the various descriptor-based flow modes. In addition to the descriptor set size being configurable, descriptor-based DMA also allows altering the flow mode of the next descriptor set. This feature allows for the transition from descriptor array mode to descriptor list mode and permits configuring the flow to stop or autobuffer mode.

Stop Mode

In stop mode, the DMA operation executes only once. If started, the DMA channel transfers the desired number of data words and stops itself again when finished. If the DMA channel is no longer used, software configures the enable bit to disable a paused channel. The channel also can generate interrupts and triggers for each row or work unit completion, depending on the desired operation.

Autobuffer Mode

In autobuffer mode, the DMA operates repeatedly in a circular manner. If transfer of all data words completes, channel reloads the address pointer (`DMA_ADDR_CUR`) automatically with the `DMA_ADDRSTART` value. The channel also can generate an interrupt.

The `DMA_CFG.FLOW` field enables autobuffer mode. The the configuration must load the `DMA_CFG.NDSIZE` field value, such that the next descriptor size is zero.

Descriptor Based Flow Modes

Descriptor-based DMA operations fetch descriptor sets from memory allowing for autonomous loading of work units on other work units. Software does not need to set up the DMA sequences directly by writing into the DMA controller registers. Rather, software keeps DMA descriptor sets in memory.

Descriptor-based DMA operations have the following additional attributes.

- The DMA controller autonomously loads the descriptor set from memory to the affected DMA controller registers on demand.
- The channel can fetch descriptor sets from any memory space that supports DMA read operations.
- The descriptor set describes the next operation that the DMA controller performs.
- The descriptor set can include information such as the DMA configuration word as well as data source or destination address, transfer count, and address modify values.

A descriptor set describes a single work unit. The next work unit can reuse some values from the previous one descriptor set. But, this reusage is possible only if they are not overwritten in the subsequent descriptor set fetches and only if the work unit requires the use of this descriptor.

The DMA channel supports the following flow modes with descriptor-based operations.

- Descriptor-Array Mode
- Descriptor-List Mode

- Descriptor-On-Demand Modes

The DMA channel supports variable descriptor set sizes within the configuration. The size of a descriptor set can contain as little as a single descriptor and the supported descriptor set sizes can differ between the various descriptor-based flow modes. In addition to configurable descriptor set size, descriptor-based DMA also allows for altering of the flow mode of the next descriptor set. Programs can transition from one descriptor-based mode to another descriptor-based mode and can also transition to any of the register-based flow modes.

Descriptor-Array Mode

When configured in this mode, the descriptor sets do not contain further descriptor pointers. Software writes the initial descriptor-pointer value, which points to an array of descriptors. This operation assumes that the individual descriptors reside next to each other and assumes that their addresses are known.

The ***Offsets for Descriptor-Array Mode Parameters and Descriptors*** table illustrates how to structure a descriptor set in memory. The descriptor sets must reside in a contiguous block or memory in the format shown in the table. Understand from this illustration that you must locate the first descriptor of the next descriptor set in the memory location immediately following the last descriptor of the current descriptor set. The values have the same order as the corresponding offset addresses of the memory-mapped register.

Table 12-9: Offsets for Descriptor -Array Mode Parameters and Descriptors

Descriptor Offset	Parameter Register
0x00	DMA_ADDRSTART
0x04	DMA_CFG
0x08	DMA_XCNT
0x0C	DMA_XMOD
0x10	DMA_YCNT
0x14	DMA_YMOD

All other DMA channel registers not loaded as a result of the descriptor set fetch retain their previous values. The channel reloads all of the current registers between the descriptor set fetch and the start of the DMA operation for the work unit.

NOTE: At a minimum, write the `DMA_DSCPTR_CUR` register prior to writing to the `DMA_CFG` register, which is the special action required to start the DMA channel.

Descriptor-List Mode

In this flow mode, multiple descriptors form a chained list in which each descriptor set contains a pointer to the next descriptor set, allowing greater flexibility in memory layout options. When the channel fetches the descriptor set, the operation loads this pointer value into the next descriptor pointer register of the DMA channel.

Descriptor Sets

The ***Offsets for Descriptor-List Mode Parameters and Descriptors*** table shows how to structure a descriptor set in memory. You can disperse the placement of the descriptor sets throughout memory, having sets reside in different memory blocks. But, each descriptor of the descriptor set must reside in a contiguous section of memory in the format shown in the table. The values have the same order as the corresponding offset addresses of the memory-mapped registers.

Table 12-10: Offsets for Descriptor-List Mode Parameters and Descriptors

Descriptor Offset	Parameter Register
0x00	DMA_DSCPTR_NXT
0x04	DMA_ADDRSTART
0x08	DMA_CFG
0x0C	DMA_XCNT
0x10	DMA_XMOD
0x14	DMA_YCNT
0x18	DMA_YMOD

All other DMA channel registers not loaded as a result of the descriptor set fetch retain their previous values. The channel reloads all of the current values of the registers between the descriptor set fetch and the start of the DMA operation for the work unit.

Minimum Startup Requirements

At a minimum, write the `DMA_DSCPTR_NXT` register prior to write to the `DMA_CFG` register, which is the special action required to start the DMA channel.

Descriptor-On-Demand Modes

The Descriptor-Array Mode and Descriptor-List Mode each have an on-demand mode of operation.

In on-demand mode, at the end of the work unit, if the DMA channel has not detected an incoming trigger event, the channel repeats the current work unit. If the DMA channel receives an incoming trigger before completion of the work unit, the channel fetches a new descriptor set.

The ***Offsets for Descriptor-Array Mode Parameters and Descriptors*** and ***Offsets for Descriptor-List Mode Parameters and Descriptors*** tables illustrate how to structure each descriptor set in memory.

Table 12-11: Offsets for Descriptor-Array Mode Parameters and Descriptors

Descriptor Offset	Parameter Register
0x00	DMA_ADDRSTART
0x04	DMA_CFG

Table 12-11: Offsets for Descriptor-Array Mode Parameters and Descriptors (Continued)

Descriptor Offset	Parameter Register
0x08	DMA_XCNT
0x0C	DMA_XMOD
0x10	DMA_YCNT
0x14	DMA_YMOD

NOTE: For descriptor-array mode, at a minimum, write the DMA_DSCPTR_CUR register prior to writing to the DMA_CFG register, which is the special action required to start the DMA channel.

Table 12-12: Offsets for Descriptor-List Mode Parameters and Descriptors

Descriptor Offset	Parameter Register
0x00	DMA_DSCPTR_NXT
0x04	DMA_ADDRSTART
0x08	DMA_CFG
0x0C	DMA_XCNT
0x10	DMA_XMOD
0x14	DMA_YCNT
0x18	DMA_YMOD

NOTE: For descriptor-list mode, at a minimum, write the DMA_DSCPTR_NXT register prior to write to the DMA_CFG register, which is the special action required to start the DMA channel.

Data Transfer Modes

In addition to supporting basic one-dimensional DMA transfers, the DMA channel also supports two-dimensional functionality.

Two-Dimensional DMA

Register-based flow modes and descriptor-based flow modes support two-dimensional data transfers.

In two-dimensional (2D) mode, the X-direction count (DMA_XCNT), the X-direction modifier (DMA_XMOD), the Y-direction count (DMA_YCNT), and the Y-direction modifier (DMA_YMOD) support arbitrary row and column sizes. Also, the modify values can be negative, allowing implementation of interleaved data streams. The DMA_XCNT value specifies the row size, and the DMA_YCNT value specifies the column size; where the DMA_XCNT value must be 2 or greater.

The DMA start address (DMA_ADDRSTART), the X-direction modifier (DMA_XMOD), and theY-direction modifier (DMA_YMOD) specifications all are in bytes. The alignment must be to a multiple of the DMA transfer word size; configured using the DMA_CFG.MSIZE bit. Misalignment results in a DMA channel error.

The DMA_XMOD register value is the byte-address increment that the channel applies after each transfer, decrementing the DMA_XCNT register. The channel does not apply the DMA_XCNT when the inner loop count ends with the DMA_XCNT_CUR register decrementing to 0 from 1. Except, the channel does apply the DMA_XCNT on the final transfer when the DMA_YCNT register is 1 and the DMA_XCNT register decrements from 1 to 0.

The DMA_YMOD register value is the byte-address increment that the channel applies after each decrement of the value in DMA_YCNT_CUR. However, the channel does not apply the DMA_YMOD value to the last item in the array on which the outer loop count (DMA_YCNT_CUR) also expires by decrementing from 1 to 0.

After the last transfer completes, DMA_YCNT_CUR is 1 and the DMA_XCNT_CUR register is 0. The DMA channels current address points to the last items address plus the DMA_XMOD register value. If the DMA channel programming selects automatic refresh (such as in autobuffer mode), the channel reloads the DMA_XCNT_CUR, DMA_YCNT_CUR, and DMA_ADDR_CUR for the first data transfer of the next work unit.

Interrupt notification is configurable for end of row or end of work unit completion.

For example, two-dimensional DMA can be used to extract interleaved data (such as RGB values for a video frame) by modifying both of the DMA_XMOD and DMA_YMOD values. The ***Capturing a Video Data Stream 2D DMA Example*** figure depicts the process of receiving a stream of the R, G, B values from an N*M frame. The inner loop of the 2D DMA configuration has three values (DMA_XCNT = 3) and a stride (DMA_XMOD) of N*M, chosen such that successive elements in each row are 1-2-3, 4-5-6 and so forth. The outer loop of the 2D DMA configuration has N*M values (DMA_YCNT = N*M) and a negative stride (DMA_YMOD) of 1-2*N*M chosen to instruct the DMA controller to jump from element 3 to 4, 6 to 7 and so forth at the end of each inner loop.

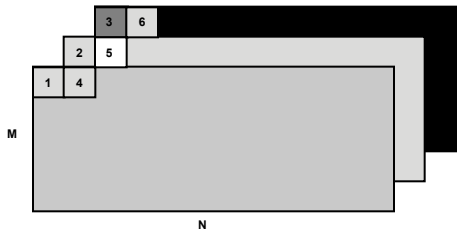

Figure 12-4: Capturing a Video Data Stream 2D DMA Example

DMA Channel Event Control

The DMA channel supports a number of events that provide notification of work unit state, peripheral data request, peripheral interrupt request and completion events, and DMA channel error conditions. In addition to flexible interrupt configuration, the DMA channel also supports incoming and outgoing triggers which are useful in synchronizing the DMA channel with other system resources.

The DMA channel has two interrupt signals for support of a number of events such as work-unit state events, peripheral interrupt request (PIRQ) events, peripheral data request (PDR) events, and DMA channel errors. The channel reports DMA channel errors on a dedicated interrupt signal. All other interrupt sources share an interrupt signal. In addition to flexible interrupt configuration, the DMA channel also supports incoming and outgoing triggers which are useful in synchronizing the DMA channel with other system resources.

The channel can signal the processor on DMA channel events using status information and optional interrupt requests. Programs can use these events to update progress of data transfers and to request intervention from the processor core. Configure most DMA channel interrupts using bits in the DMA_CFG register. Dedicated bits in the DMA_STAT register report the occurrence of various events. Use write-one-to-clear (W1C) operations to clear interrupt requests from the status register.

NOTE: Hardware does not clear the interrupt status bits automatically, even when programs disable then reenable the DMA channel. In this situation, the channel deasserts the interrupt signal, after the program disables the DMA channel. But, the status bit remains set until software either re-enables the DMA channel or clears the status bit.

The DMA channel supports the following categories of events on the interrupt signals:

- Work-unit state events generate interrupts on row or on work unit DMA completion.
- A peripheral uses peripheral interrupt request (PIRQ) events to signal when it has completed the transfer of all data.
- A peripheral uses peripheral data request (PDR) events to request data from a disabled or idle DMA channel.
- Error events signal a failure in the work unit.

ATTENTION: While in an error state, the DMA channel does not generate an interrupt to the processor for a work-unit state event or a PIRQ event, nor does the channel forward a PDR event.

Event Signals

The ***Event Signals*** table provides descriptions of DMA channel events.

Table 12-13: Event Signals

Signal	Width (bits)	Description
DMA_ERROR	1	Used to signal an error condition in the DMA channel. The source of the error can be determined by reading the DMA_STAT.ERRC bit.
DONE_PIRQ_INT	1	Signal used to indicate DMA completions events, PIRQ events and also for forwarding PDR events based on configuration. Read the corresponding fields in DMA_STAT to determine the source of the event.
DMA_TRIG_OUT	1	Trigger output that gets routed to the TRU and can be configured to provide notification on row or work unit completion.
DMA_TRIG_IN	1	Trigger input from the TRU that can be used to control the start of a work unit.

Work Unit State Events

Completing a row or a work unit generates a work-unit state event. For either of these events to generate an interrupt, the configuration of the interrupt of the DMA channel must select one of the available work-unit completion modes.

- Current X count reaching 0 for row completion or 1D DMA work unit completion.

- Current Y count reaching 0 for work unit completion of 2D DMA.

NOTE: For 1D DMA, a DMA channel configuration error results if the configuration generates the interrupt when the current Y counter reaches.

The DMA channel issues the last memory read or write transaction for the row or work unit, then pauses until the return of the read or write acknowledge. After successful acknowledge of the transfer, the DMA channel issues the interrupt and continues to process the next row or work unit.

Waiting for acknowledgement of the memory access results in a delay. However, programs can read or modify data in the memory without adversely affecting or being affected by the DMA transfer.

NOTE: While the DMA channel pauses waiting for acknowledgement of the memory transfer, the DMA channel is still capable of fetching the next descriptor set. This fetch gets the channel ready to process the next work unit as soon as the memory access completes.

The channel configuration of the synchronization feature also affects interrupt timing. For memory-read operations with synchronization enabled, the channel delays the interrupt until completion of the last transfer from the DMA channel FIFO to the peripheral. The synchronization feature does not affect interrupt timing for memory write operations.

Peripheral Interrupt-Request Events

For peripheral-transmit operations, a peripheral connected to the DMA channel can use peripheral interrupt request (PIRQ) events to indicate that data has left the channel FIFO and to indicate transfer completion.

In order to support PIRQ interrupts, correctly configure the interrupt of the DMA channel. This configuration disables the generation of interrupts based on work unit state and instead results in generating an interrupt when the DMA channel receives the command from the peripheral.

The channel only generates the interrupt under the following conditions:

- The configuration enables the DMA channel.
- The DMA channel is in the stop state.
- The configuration of the DMA channel interrupt selects PIRQ operation.

Peripheral Data-Request Events

Peripheral data request (PDR) events occur when an interfaced peripheral requests data from the DMA channel and the DMA channel (either disabled or enabled) is in the stop state.

When a peripheral sends a data-request command to a disabled DMA channel, the DMA channel generates an interrupt to the System Event Controller (SEC). There is no status information reported about this event in the status register of the DMA channel. Instead, the channel identifies the PDR event from the fact that the DMA channel generated an interrupt while disabled. It is possible to further confirmation event status by verifying the status of the peripheral interfaced to the DMA channel.

This operation forwards data requests as interrupts when the DMA channel is in the disabled state. Also, the DMA channel is able to forward PDR events as an interrupt when the DMA channel is in the stop state after completion of a work unit. The forwarding of this interrupt when the DMA channel is in the stop state is optional and configured by the program during DMA channel configuration.

DMA Channel Triggers

DMA channel triggers are useful for synchronizing the DMA channel with other events in the system. One usage is to combine channel triggers with each other to create ping-pong buffers. Another usage is to combine the triggers with interrupts to notify the processor on reaching a particular milestone that requires service. The channel also can use triggers to enforce a handshake DMA operation in which the trigger acts as a signal for a DMA request.

NOTE: Using the trigger to control the pace of data transfers, such as for handshake DMA, requires that all the data for the entire work unit is ready for transfer.

The DMA channel has a single incoming trigger that can control the pace of the data transfers performed by the DMA channel. The configuration can direct the DMA channel to wait for the incoming trigger before starting the work unit transfer or fetching a descriptor set from memory.

The DMA channel also has a single outgoing trigger signal. This configuration can direct this trigger to signal the end of row or an entire work unit. The DMA channel issues the last memory read or memory write transaction for the row or work unit, then pauses until return of the transfer acknowledge. After acknowledgement of the transfer, the DMA channel issues the trigger before processing the next row or work unit.

Issuing Triggers

The DMA channel configuration can direct the channel to generate an outgoing trigger signal at the end of row or the end of a work unit. The DMA channel issues the last memory read or memory write transaction for the row or work unit, then pauses until the return of the transfer acknowledge. After acknowledgement of the transfer, the DMA channel issues the trigger before processing the next row or work unit.

NOTE: While the DMA channel pauses waiting for acknowledgement of the memory transfer, the DMA channel is still capable of fetching the next descriptor set. This fetch gets the channel ready to process the next work unit as soon as the memory access completes.

Waiting For Triggers

Programs can use triggering to control the pace of data transfers performed by the DMA channel. The DMA channel enters a wait state before beginning the next work unit if the configuration enables `DMA_CFG.TWAIT` and either of the following apply:

- The channel receives a trigger since the last time the DMA channel left the wait state.
- The channel receives a trigger since its transition from disable to enable.

In the wait state, the DMA channel also does not perform a descriptor fetch. After receiving a trigger, the DMA channel leaves the wait state and begins the next work unit or fetches the next descriptor if configured for a descriptor-based mode of operation.

If a memory-mapped register write operation programs the channel with stop flow mode enabled (DMA_CFG.TWAIT bit) and the channel has not already received a trigger, the DMA channel enters a wait state before performing the data transfer. On receiving the trigger, the DMA channel begins the data transfer portion of the work unit. Once the data transfer is complete, the DMA channel enters the stop state.

If a memory-mapped register write operation programs the DMA channel with the flow mode configured to one of the descriptor-based modes, the DMA channel enters the wait state before performing the descriptor fetch. After completing the descriptor fetch, the DMA channel immediately proceeds to the data transfer, regardless of the value of the DMA_CFG.TWAIT bit. If another (next) descriptor fetch follows the descriptor fetch, the DMA channel enters a wait state before fetching the next descriptor.

If the descriptor fetch returns a descriptor with stop flow mode, the DMA_CFG.TWAIT value for that descriptor does not affect the DMA as the channel enters the stop state after completing the data transfer. The DMA channel only enters the wait state based on DMA_CFG.TWAIT before the next work unit or descriptor fetch.

If the descriptor fetch returns a descriptor configured for autobuffer flow mode, the DMA_CFG.TWAIT for that descriptor does not affect the DMA for the first work unit of the autobuffer transfer. After completing the first work unit and not receiving another trigger, the DMA channel enters the wait state before reinitializing its counters and address registers (if not configured for current addressing). The channel performs the next work unit after receiving the trigger.

The incoming trigger can occur when the DMA channel has not entered the wait state. The trigger can occur while the DMA channel is executing a work unit, is performing descriptor fetch, or is in the stop state. The trigger is held internally. After the work unit is complete, the DMA channel skips the wait state and proceeds directly to executing the following work unit. If the DMA_CFG.TWAIT bit is not enabled, the DMA channel also skips the wait state. However, the trigger is held internally and is used the next time the configuration enables DMA_CFG.TWAIT. This trigger retention allows programs to enable the DMA_CFG.TWAIT functionality several work units apart without concern for losing a trigger. The DMA channels trigger overrun enable functionality can be enabled in all work units to ensure that multiple triggers do not occur between the work units with the DMA_CFG.TWAIT bit enabled.

DMA Channel Programming Model

Several synchronization and control methods are available for use in development of software tasks which manage peripheral DMA and memory DMA. Software must accept requests for new DMA transfers from other software tasks, integrate these transfers into existing transfer queues, and reliably notify other tasks when the transfers are complete.

In the processor, it is possible to manage each peripheral DMA and memory DMA stream with a separate task or to manage them together with any other stream. Each DMA channel has independent, orthogonal control registers, resources, and interrupts. So, the selection of the control scheme for one channel does not affect the choice of control scheme on other channels. For example, one peripheral can use a linked-descriptor-list, interrupt-driven scheme

while another peripheral can simultaneously use a demand-driven, buffer-at-a-time scheme synchronized by polling DMA events.

The topics that follow describe the steps required to configure the DMA channel for the various modes in addition to the programming concepts required for software synchronization.

Mode Configuration

Use the step-by-step directions that follow to set up the DMA channel for operating modes.

Register-Based Linear-Buffer Stop Flow Mode

This procedure configures the DMA channel of a peripheral to read data from internal memory and to send it to the peripheral for transmission.

Assume that the peripheral is in a state where it is ready to transmit data received from the DMA channel.

The task involves writing to a number of DMA channel MMR registers to configure a DMA channel to:

- Read data from internal memory, and
- Send it to a peripheral connected to the peripheral DMA bus.

On DMA completion, the DMA channel enters the idle state until either disabled or reconfigured for a new transfer.

1. Write the DMA_ADDRSTART register.

   ADDITIONAL INFORMATION: Software can use the address to calculate the most optimum possible DMA_CFG.MSIZE.

2. Calculate most optimum DMA_CFG.MSIZE based on the DMA_ADDRSTART register and number of bytes in work unit.

   ADDITIONAL INFORMATION: The number of bytes in the work unit must be a multiple of the selected DMA_CFG.MSIZE, and the calculation also must consider the start address alignment.

3. Write the DMA_XCNT register based on the calculated DMA_CFG.MSIZE.

   ADDITIONAL INFORMATION: The DMA_XCNT value is the number of DMA_CFG.MSIZE transfers to make up the entire work unit.

4. Write the DMA_XMOD register.

   ADDITIONAL INFORMATION: For a linear buffer transfer, determine the value in DMA_XMOD from the selected DMA_CFG.MSIZE. Always specify this register as a number of bytes.

5. Write the DMA_CFG register with DMA_CFG.EN configured to enable the DMA channel.

ADDITIONAL INFORMATION: Set the DMA_CFG.FLOW bit for STOP mode. Configure the DMA_CFG.WNR bit for memory read operation. Configure the DMA_CFG.PSIZE bits to a value no larger than the supported bus width of the peripheral DMA bus.

- The DMA_CFG.SYNC bit can be configured to control DMA completion notification timing.
- Interrupts and triggers also can be configured at this step depending on requirements.

Now, the DMA channel is enabled, and the buffer is transferred. The DMA channel enters the IDLE state upon completion of the work unit.

Register-Based Autobuffer Flow Mode

This procedure configures the DMA channel of a peripheral to read data from internal memory and send it to the peripheral for transmission. The transmission of the buffer repeats endlessly.

Assume the peripheral is in a state where it is ready to transmit data received from the DMA channel.

The task involves writing to a number of DMA channel MMR registers to configure a DMA channel to:

- Read data from internal memory, and
- Send it to a peripheral connected to the peripheral DMA bus.

On DMA completion, the DMA channel restarts the DMA operation, creating an endless circular buffer transfer.

1. Write the DMA_ADDRSTART register.

   ADDITIONAL INFORMATION: Use the address to calculate the most optimum possible DMA_CFG.MSIZE.

2. Calculate most optimum DMA_CFG.MSIZE based on the DMA_ADDRSTART register and number of bytes in work unit.

   ADDITIONAL INFORMATION: The number of bytes in the work unit must be a multiple of the selected DMA_CFG.MSIZE, and the calculation must consider the start address alignment.

3. Write the DMA_XCNT register based on calculated DMA_CFG.MSIZE.

   ADDITIONAL INFORMATION: The DMA_XCNT register value is the number of DMA_CFG.MSIZE transfers to make up the entire work unit.

4. Write the DMA_XMOD register.

   ADDITIONAL INFORMATION: For a linear buffer transfer, determine the value in DMA_XMOD from the selected DMA_CFG.MSIZE. Always specify this register as a number of bytes.

5. Write the DMA_CFG register with the DMA_CFG.EN bit configured to enable the DMA channel.

   ADDITIONAL INFORMATION: Set the DMA_CFG.FLOW bit for auto-buffer mode. Configure the DMA_CFG.WNR bit for memory read operation. Configure the DMA_CFG.PSIZE bit to a value no larger than the supported bus width of the peripheral DMA bus.

- The DMA_CFG.SYNC bit can be configured to control DMA completion notification timing.
- Interrupts and triggers also can be configured at this step depending on requirements.

Now, the DMA channel is enabled, and the buffer transfers until the DMA channel is disabled.

Descriptor-Array Flow Mode

This procedure configures the DMA channel of a peripheral to:

- Read data from memory as described by the descriptor sets in the array, and
- Send the data to the peripheral for transmission.

Descriptor sets are read from an array in memory to configure the individual work units.

Assume the peripheral is in a state where it is ready to transmit data received from the DMA channel. Assume that the array of descriptors is to be initialized with the last descriptor set configured for STOP flow mode.

The task involves writing to a number of DMA channel MMR registers to:

- Configure a DMA channel to read the array in memory, containing the first descriptor set that configured the DMA channel to retrieve, and
- Send the data to a peripheral connected to the peripheral DMA bus.

On DMA completion, the DMA channel enters the idle state until either disabled or reconfigured for a new transfer.

1. Write the DMA_DSCPTR_CUR register with the address of the array in which the descriptor sets are stored.

   ADDITIONAL INFORMATION: The array address must meet any processor alignments restrictions imposed by descriptor fetches.

2. Write the DMA_CFG register with the DMA_CFG.EN bit configured to enable the DMA channel.

   ADDITIONAL INFORMATION: Set the DMA_CFG.FLOW bit for descriptor-array mode. Configure the DMA_CFG.NDSIZE bits to describe the number of descriptor elements contained within the first descriptor set. Configure the DMA_CFG.WNR bit for memory read operation. Configure the DMA_CFG.PSIZE bits to a value no larger than the supported bus width of the peripheral DMA bus.

   - The descriptor set that is fetched controls the DMA_CFG.SYNC configuration and the interrupt or trigger configurations.

   The first descriptor set is fetched from memory location provided by the DMA_DSCPTR_CUR register and loaded to the MMR registers of the DMA channel.

Now, the DMA channel is processing all the work units provided in the descriptor array. The DMA channel enters the IDLE state on completion of the final work unit that was configured for STOP flow mode.

Descriptor-List Flow Mode

This procedure configures DMA channel of a peripheral to:

- Read data from memory as described by the descriptor sets in the list, and
- Send it to the peripheral for transmission.

The DMA controller reads descriptor sets from a list of descriptors. With the list, each descriptor set has a descriptor that points to the next descriptor set location in memory.

Assume the peripheral is to be in a state where it is ready to transmit data received from the DMA channel. Assume that the list of descriptors is to be initialized with the last descriptor set in the list configured for STOP flow mode.

The task involves writing to a number of DMA channel MMR registers to:

- Configure a DMA channel to read the list in memory, containing the first descriptor set that configured the DMA channel to retrieve, and
- Send the data to a peripheral connected to the peripheral DMA bus.

On DMA completion, the DMA channel enters the idle state until either disabled or reconfigured for a new transfer.

1. Write the DMA_DSCPTR_NXT register with the address of the first descriptor in the list to be processed.

   ADDITIONAL INFORMATION: The array address must meet any processor alignments restrictions imposed by descriptor fetches.

2. Write the DMA_CFG register with the DMA_CFG.EN configured to enable the DMA channel.

   ADDITIONAL INFORMATION: Set the DMA_CFG.FLOW for descriptor-list mode. Configure the DMA_CFG.NDSIZE bit to describe the number of descriptor elements contained within the first descriptor set. Configure the DMA_CFG.WNR bit for memory read operation. Configure the DMA_CFG.PSIZE bit to a value no larger than the supported bus width of the peripheral DMA bus.

   - The descriptor set that is fetched controls the DMA_CFG.SYNC configuration and controls the interrupt or trigger configurations.

   The first descriptor set is fetched from the memory location provided by DMA_DSCPTR_NXT and is loaded to the MMR registers of the DMA channel.

Now, the DMA channel is processing all the work units provided in the descriptor list. The DMA channel enters the idle state when the final work unit that was configured for stop-flow mode is complete.

Register-Based Memory-to-Memory Transfer-in-Stop Flow Mode

This procedure configures a memory DMA channel pair in stop-flow mode. One DMA channel is configured for memory read operations, while the other DMA channel is configured for memory write.

The task involves writing to a number of DMA channels on two DMA channels that create a memory DMA pair. On DMA completion, the DMA channel enters the idle state, until either the DMA channel is disabled or is reconfigured for a new transfer.

1. Write the DMA_ADDRSTART register of the source DMA channel.

   ADDITIONAL INFORMATION: The address can be used to calculate the most optimum DMA_CFG.MSIZE possible.

2. Calculate most optimum DMA_CFG.MSIZE based on the DMA_ADDRSTART register and number of bytes in work unit.

   ADDITIONAL INFORMATION: The number of bytes in the work unit must be a multiple of the selected DMA_CFG.MSIZE and the start address alignment must also be considered.

3. Write the DMA_XCNT register of the source DMA channel based on calculated DMA_CFG.MSIZE.

   ADDITIONAL INFORMATION: DMA_XCNT is the number of DMA_CFG.MSIZE transfers to make up the entire work unit.

4. Write the DMA_XMOD register of the source DMA channel.

   ADDITIONAL INFORMATION: For a linear buffer transfer, determine the value in DMA_XMOD from the selected DMA_CFG.MSIZE. This register is always specified in the number of bytes.

5. Write the DMA_ADDRSTART register of the destination DMA channel.

   ADDITIONAL INFORMATION: The address can be used to calculate the most optimum DMA_CFG.MSIZE possible.

6. Calculate most optimum DMA_CFG.MSIZE based on the DMA_ADDRSTART register and number of bytes in work unit.

   ADDITIONAL INFORMATION: The number of bytes in the work unit must be a multiple of the selected DMA_CFG.MSIZE and the start address alignment must also be considered.

7. Write the DMA_XCNT register of the destination DMA channel based on calculated DMA_CFG.MSIZE.

   ADDITIONAL INFORMATION: DMA_XCNT is the number of DMA_CFG.MSIZE transfers to make up the entire work unit.

8. Write the DMA_XMOD register of the destination DMA channel.

   ADDITIONAL INFORMATION: For a linear buffer transfer, determine the value in DMA_XMOD from the selected DMA_CFG.MSIZE. This register is always specified in the number of bytes.

9. Write the DMA_CFG register of the source DMA channel with DMA_CFG.EN configured to enable the DMA channel.

ADDITIONAL INFORMATION: The DMA_CFG.FLOW bit must be set for stop mode. The DMA_CFG.WNR bit must be configured for memory read operation. The DMA_CFG.PSIZE bits must be configured to a value no larger than the supported bus width of the peripheral DMA bus.

- The DMA_CFG.SYNC bit can be configured to control DMA completion notification timing.
- Interrupts and triggers also can be configured at this step, depending on requirements. The interrupts and triggers are enabled within the destination DMA channel configuration.

The memory read DMA transfer begins.

10. Write the DMA_CFG register of the destination DMA channel with DMA_CFG.EN configured to enable the DMA channel.

    ADDITIONAL INFORMATION: The DMA_CFG.FLOW bit must be set for stop mode. The DMA_CFG.WNR bit must be configured for memory write operation. The DMA_CFG.PSIZE bits must be configured to a value no larger than the supported bus width of the peripheral DMA bus. This value must also match the value written for the source DMA channel configuration.

    - Interrupts and triggers also can be configured at this step depending on requirements.

    The memory write DMA transfer begins.

Both memory DMA channels are now running and the data is transferred from the source address to the destination address. The DMA channel enters the IDLE state upon completion of the work unit.

Programming Concepts

Using the features, operating modes, and event control for the DMA channel to their greatest potential requires an understanding of some DMA channel-related concepts.

Synchronization of Software and DMA

A critical element of software DMA management is synchronization of DMA work unit completion with the software. Achieve this synchronization using DMA channel interrupt and trigger events, using a poll of the status bits of these events within the DMA channel registers, or combining these techniques. Polling for address or count can only provide synchronization within loose tolerances comparable to pipeline lengths.

Interrupt and Trigger Event-Based Synchronization

Interrupt and trigger based synchronization methods must avoid interrupt or trigger overrun. An overrun occurs when some events fail to invoke the event handler of a DMA channel for every event due to excessive latency in processing of events. The system design must ensure to either:

- Schedule only one event per channel (for example, at the end of a descriptor list), or
- Space the generated events sufficiently far apart in time that system processing budgets can guarantee service of every event.

The DMA channel issues status information through an interrupt or trigger event or changes event status bits in the DMA_STAT register. This status guarantees that the last memory operation of the work unit is complete. For memory read DMA transactions, this status means that the FIFO of the DMA channel safely receives the final memory read data. For DMA transactions writing to memory, this status indicates that the DMA channel received an acknowledge of completion of the last write transfer of the work unit.

Register Polling Based Synchronization

Do not poll the DMA channel registers (DMA_ADDR_CUR, DMA_DSCPTR_CUR, DMA_XCNT_CUR, or DMA_YCNT_CUR) as a method of precisely synchronizing DMA with data processing. This approach is inaccurate due to the operation of the DMA channel FIFOs and DMA or memory pipelining. The current address, pointer, and count registers change several cycles in advance of the completion of the corresponding memory operation. This timing is measurable from the time at which the results of the operation are first visible to the core by memory read or write instructions.

For example, in a DMA channel memory write operation to external memory, assume DMA channel *A* initiates a DMA channel write operation. For memories with access latency, this operation requires many system-clock cycles. Meanwhile, DMA channel *B* (which does not in itself incur latency) initiates a transfer, which stalls behind the slow operation of channel *A*. Software monitoring channel *B* could not safely conclude whether the memory location pointed to by the DMA_ADDR_CUR of channel *B*. Also, the software cannot conclude whether the register has been written based solely on the contents of this register.

Polling of the current address, pointer, and count registers can permit loose synchronization of DMA with software. But, the software must allow for the lengths of the DMA or memory pipeline. Also, software must consider the length of the DMA FIFO for a particular peripheral. If the FIFOs are filled with incomplete work, the DMA channel does not advance current address, pointer, or count registers. The incomplete work includes reads that have been started but have not yet finished.

Additionally, software must consider the length of the pipelines to the destination memory. If the DMA FIFO length and channel memory-pipeline length are added, software can estimate the maximum number of incomplete memory operations in progress.

NOTE: The estimate would be a maximum, as the DMA or memory pipeline can include traffic from other DMA channels.

Descriptor Queues

A system designer may want to write a DMA manager facility which accepts DMA requests from other software. The DMA manager software does not know in advance when new work requests are received or what these requests contain. The software could manage these transfers using a circular linked list of DMA descriptors. In such a list, each descriptor sets the DMA_DSCPTR_NXT descriptor, which points to the next descriptor set. And, the last descriptor set in the list points to the first descriptor set.

The code that writes into this descriptor list could use the circular addressing modes of the processor. This approach does not need to use comparison and conditional instructions to manage the circular structure. In this case, the

DMA_DSCPTR_NXT descriptor of each descriptor set can be written once at startup, and skipped over as new contents are written for each descriptor.

The recommended method for synchronization of a descriptor queue is to use an interrupt or trigger. The descriptor queue is structured, such that (at least) the final valid descriptor set is always programmed to generate an interrupt or trigger event upon completion. More detail is provided in the following sections.

- Queues Using Event Generation for Every Descriptor Set
- Queues Using Minimal Events

Queues Using Event Generation for Every Descriptor Set

In this system, the DMA manager software synchronizes with the DMA channel by enabling an interrupt or trigger on every descriptor set. Only use this method if the system design can guarantee that each work unit completion event is serviced separately (no interrupt or trigger overrun).

To maintain synchronization of the descriptor set queue, the non-interrupt software maintains a count of descriptor sets added to the queue. The event handler (either interrupt or trigger) maintains a count of completed descriptor sets removed from the queue. The counts are equal only when the DMA channel is paused after having processed all the descriptor sets.

When each new work unit event is received, the DMA manager software initializes a new descriptor set, taking care to set the flow to stop mode. Next, the software compares the descriptor set counts to determine whether the DMA channel is running. If the DMA channel is paused (counts equal), the software increments its count. Then, the software starts the DMA channel by writing the DMA_CFG of the new descriptor set.

If the counts are unequal, the software instead modifies the DMA_CFG of the next-to-last descriptor set, such that it now describes the newly queued descriptor set. This operation does not disrupt the DMA channel provided the rest of the descriptors of the set are initialized in advance. It is necessary to synchronize the software to the DMA to determine whether the DMA channel read the new or the old DMA_CFG value.

The event handler performs the synchronization operation. When an event is detected, the handler reads the DMA_STAT register of the DMA channel. If the DMA_STAT.RUN bit indicates that the DMA channel is running, the channel has moved on to processing another descriptor. The event handler can increment its count and exit. If the DMA_STAT.RUN bit indicates that the channel is not running, the channel is paused because either:

- there are no more descriptor sets to process, or
- the last descriptor set was queued too late

Where *too late* means that the modification of the DMA_CFG of the next-to-last descriptor set occurred *after* that descriptor was read into the DMA channel. In this case, the event handler does the following:

- writes the DMA_CFG value appropriate for the last descriptor set to DMA_CFG register of the DMA channel,
- increments the completed descriptor count, and
- exits.

If the event latencies of the system are large enough to cause any of the events to be dropped, this system can fail. An event handler capable of safely synchronizing multiple descriptor set interrupts is complex, performing several MMR accesses to ensure robust operation. In such a system environment, a minimal event synchronization method is preferred.

Queues Using Minimal Events

In this system, only one DMA interrupt or trigger event is generated in the queue at any time. The DMA event handler for this system can also be extremely short. Here, the descriptor queue is organized into an *active* and a *waiting* portion, where events are enabled only on the last descriptor set in each portion.

When each new DMA request is processed, the software fills in the content of a new descriptor set and adds it to the waiting portion of the queue. The DMA_CFG descriptor of the descriptor set must have the flow set to stop mode. If more than one request is received before the DMA queue completion event occurs, the non-interrupt code queues later descriptor sets. It forms a waiting portion of the queue separate from the active portion of the queue that the DMA channel is processing. In other words, all but the last active descriptor sets contain FLOW values for a descriptor-based mode and have no event enable set.

The last active descriptor set has the stop flow mode and an event generation enabled. Also, all but the last waiting descriptor sets are configured for descriptor-based flow modes with no event generation. Only the last waiting descriptor set is configured for stop flow mode and event generation enabled. This configuration ensures that the DMA channel can automatically process the whole active queue before then issuing one event. Also, this arrangement makes it easy to start the waiting queue within the event handler by a single DMA_CFG register write.

After queuing a new waiting descriptor, the non-interrupt software leaves a message for its interrupt handler in a memory mailbox location. The location contains the desired DMA_CFG value for starting the first waiting descriptor set in the waiting queue (or 0, indicating no waiting descriptors).

The software must not modify the contents of the active descriptor set queue directly once processing by the DMA channel has started, unless careful synchronization measures are taken. In the most straightforward implementation of a descriptor set queue, the DMA manager software never modifies descriptors on the active queue. Instead, the DMA manager waits until the DMA queue completion event indicates that the processing of the entire active queue is complete.

When a DMA queue completion event is received, the event handler reads the mailbox from the non-interrupt software and writes the value to the DMA_CFG register of the DMA channel. This write to a register restarts the queue, effectively transforming the waiting queue to an active queue. The event handler then passes a message back to the non-interrupt software indicating the location of the last descriptor set accepted into the active queue.

However, the event handler can read its mailbox and find a DMA_CFG value of zero, indicating there is no more work to perform. It then passes an appropriate message back to the non-interrupt software indicating that the queue has stopped.

The non-interrupt software which accepts new DMA work unit requests must synchronize the activation of a new work unit with the interrupt handler. If the queue has stopped (the mailbox from the event handler is zero), the non-interrupt software must start the queue. (The queue starts by writing the first descriptor sets DMA_CFG value to the DMA_CFG register of the channel). If the queue is not stopped, the non-interrupt software must not write the

DMA_CFG register. (This write causes a DMA error). Instead, it must queue the descriptor onto the waiting queue and update its mailbox directed to the event handler.

ADSP-CM40x DMA Register Descriptions

DMA Channel (DMA) contains the following registers.

Table 12-14: ADSP-CM40x DMA Register List

Name	Description
DMA_ADDRSTART	Start Address of Current Buffer Register
DMA_ADDR_CUR	Current Address Register
DMA_BWLCNT	Bandwidth Limit Count Register
DMA_BWLCNT_CUR	Bandwidth Limit Count Current Register
DMA_BWMCNT	Bandwidth Monitor Count Register
DMA_BWMCNT_CUR	Bandwidth Monitor Count Current Register
DMA_CFG	Configuration Register
DMA_DSCPTR_CUR	Current Descriptor Pointer Register
DMA_DSCPTR_NXT	Pointer to Next Initial Descriptor Register
DMA_DSCPTR_PRV	Previous Initial Descriptor Pointer Register
DMA_STAT	Status Register
DMA_XCNT	Inner Loop Count Start Value Register
DMA_XCNT_CUR	Current Count(1D) or intra-row XCNT (2D) Register
DMA_XMOD	Inner Loop Address Increment Register
DMA_YCNT	Outer Loop Count Start Value (2D only) Register
DMA_YCNT_CUR	Current Row Count (2D only) Register
DMA_YMOD	Outer Loop Address Increment (2D only) Register

Start Address of Current Buffer Register

The DMA_ADDRSTART register contains the start address of the work unit currently targeted for DMA. This register is read/write prior to enabling the channel, but is read-only after enabling channel.

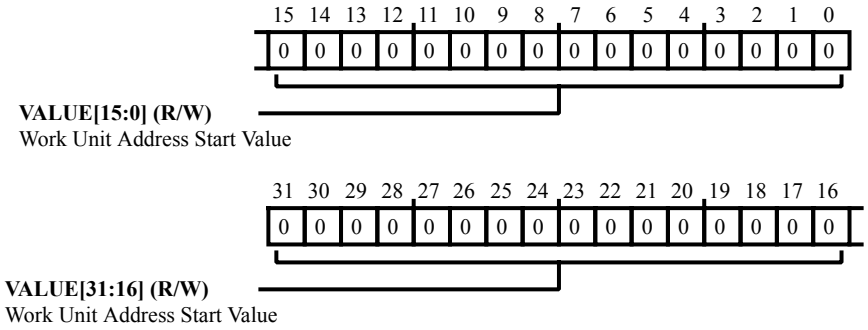

Figure 12-5: DMA_ADDRSTART Register Diagram

Table 12-15: DMA_ADDRSTART Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit Address Start Value. The DMA_ADDRSTART.VALUE bit field contains the start address of the Work Unit currently targeted for DMA.

Current Address Register

The DMA_ADDR_CUR register contains the present memory transfer address for a given work unit. At the start of a work unit, the DMA_ADDR_CUR register is loaded from the DMA_ADDRSTART register, and the DMA_ADDR_CUR register is incremented as each transfer occurs. The DMA_ADDR_CUR register is read/write prior to enabling the channel, but is read-only after enabling the channel.

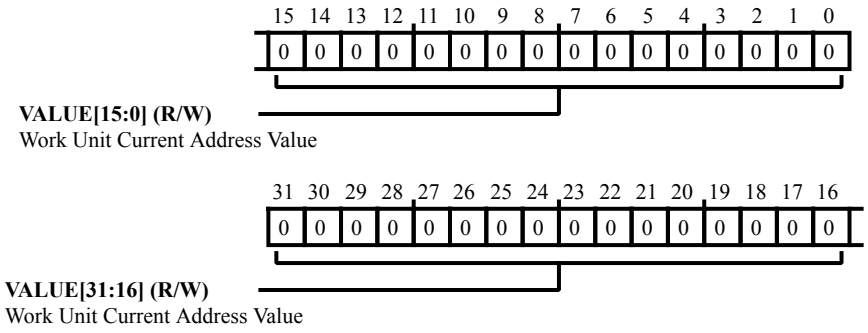

Figure 12-6: DMA_ADDR_CUR Register Diagram

Table 12-16: DMA_ADDR_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit Current Address Value. The DMA_ADDR_CUR.VALUE bit field contains the present memory transfer address for a given work unit.

Bandwidth Limit Count Register

The DMA_BWLCNT register contains a count that determines how often the DMA issues memory transactions. The DMA loads the value from DMA_BWLCNT register into DMA_BWLCNT_CUR and decrements the current value each SYSCLK cycle. When DMA_BWLCNT_CUR reaches 0x0000, the next request is issued, and the DMA reloads DMA_BWLCNT_CUR. This bandwidth limit functionality is not applied to descriptor fetch requests. Programming 0x0000 allows the DMA to request as often as possible. 0xFFFF is a special case and causes requests to stop.

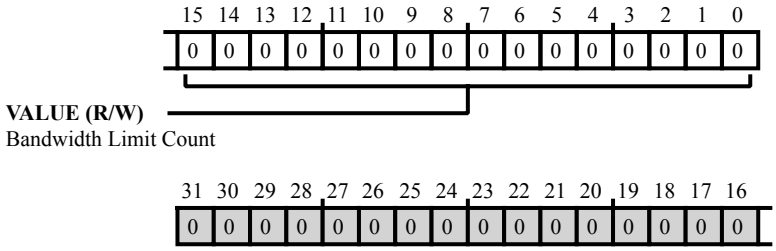

Figure 12-7: DMA_BWLCNT Register Diagram

Table 12-17: DMA_BWLCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Bandwidth Limit Count. The DMA_BWLCNT.VALUE bit field contains a count that determines how often the DMA issues memory transactions.

Bandwidth Limit Count Current Register

The DMA_BWLCNT_CUR register contains the number of SYSCLK count cycles remaining before the next request is issued.

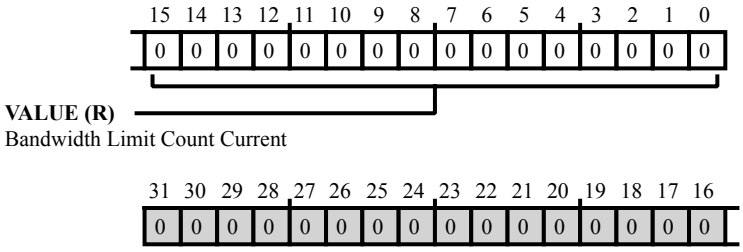

Figure 12-8: DMA_BWLCNT_CUR Register Diagram

Table 12-18: DMA_BWLCNT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Bandwidth Limit Count Current. The DMA_BWLCNT_CUR.VALUE bit field contains the number of SYSCLK count cycles remaining before the next request is issued.

Bandwidth Monitor Count Register

The DMA_BWMCNT register contains the maximum number of SYSCLK cycles allowed for a work unit to complete. Each time the DMA_CFG register is written (MMR access only), a work unit ends or an autobuffer wraps. The DMA loads the value in this register into the DMA_BWMCNT_CUR register.

The DMA decrements DMA_BWMCNT_CUR every SYSCLK a work unit is active. If the DMA_BWMCNT_CUR register reaches 0x0000_0000, the DMA_STAT.IRQERR bit is set, and the DMA_STAT.ERRC bit field is set to 0x6. The DMA_BWMCNT_CUR remains at 0x0000_0000 until it is reloaded when the work unit completes.

Unlike other errors, a bandwidth monitor error does not stop work unit processing. Programming 0x0000_0000 disables bandwidth monitor functionality.

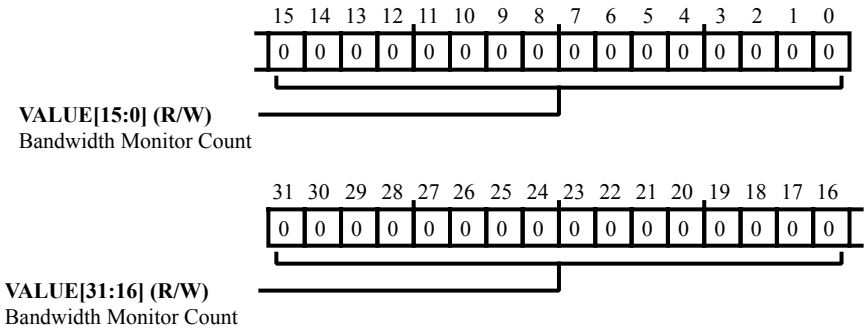

Figure 12-9: DMA_BWMCNT Register Diagram

Table 12-19: DMA_BWMCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Bandwidth Monitor Count. The DMA_BWMCNT.VALUE bit field contains the maximum number of SYSCLK cycles allowed for a work unit to complete.

Bandwidth Monitor Count Current Register

The DMA_BWMCNT_CUR register contains the number of cycles remaining for the current descriptor to complete.

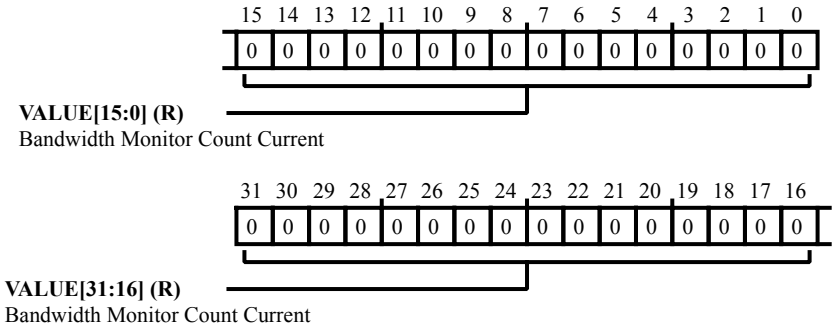

Figure 12-10: DMA_BWMCNT_CUR Register Diagram

Table 12-20: DMA_BWMCNT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Bandwidth Monitor Count Current. The DMA_BWMCNT_CUR.VALUE bit field contains the number of cycles remaining for the current descriptor to complete.

Configuration Register

The DMA_CFG sets up DMA parameters and operation modes. Writing to the DMA_CFG register while a DMA process is already running causes a DMA error (except when clearing the DMA_CFG.EN bit).

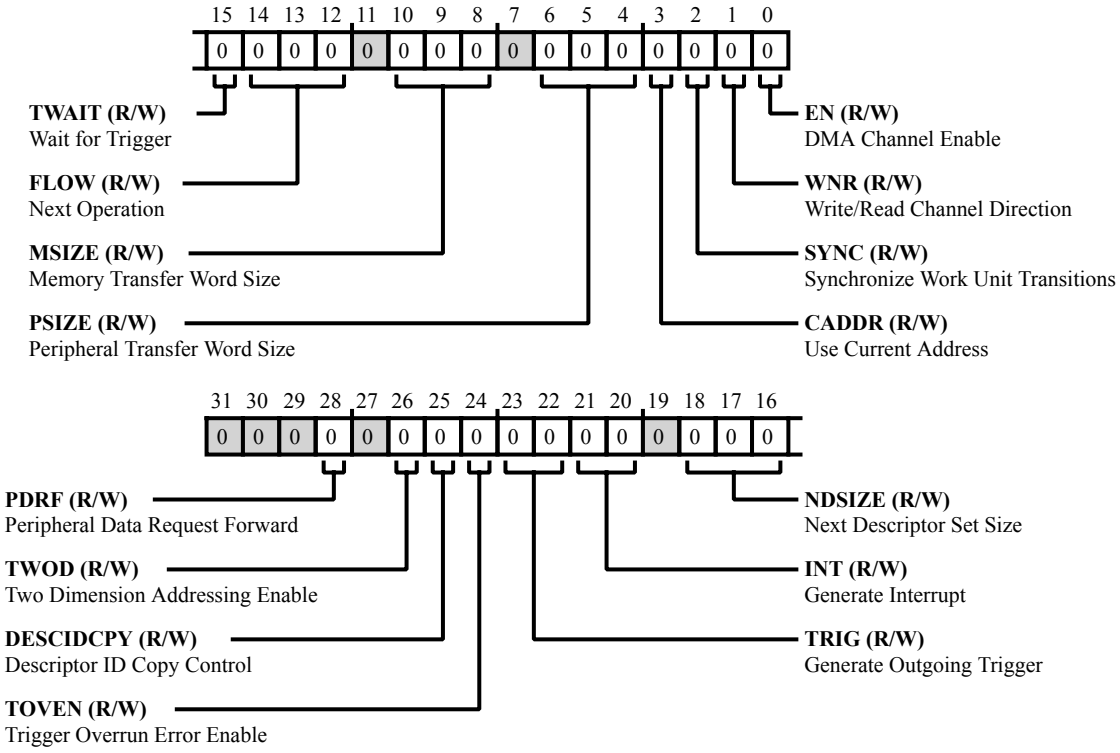

Figure 12-11: DMA_CFG Register Diagram

Table 12-21: DMA_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
28 (R/W)	PDRF	Peripheral Data Request Forward. The DMA_CFG.PDRF bit defines how the DMA handles data requests from the peripheral while in idle state after a stop mode or memory read work unit. If set, the DMA forwards the peripheral data request as an interrupt. This bit applies only to the DMA_CFG.FLOW bits configured for stop and DMA_CFG.WNR bits configured for memory read.	
		0	Peripheral Data Request Not Forwarded
		1	Peripheral Data Request Forwarded

Table 12-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/W)	TWOD	Two Dimension Addressing Enable. The DMA_CFG.TWOD selects whether the DMA addressing involves only DMA_XCNT and DMA_XMOD (one-dimensional DMA) or also involves DMA_YCNT and DMA_YMOD (two-dimensional DMA).	
		0	One-Dimensional Addressing
		1	Two-Dimensional Addressing
25 (R/W)	DESCIDCPY	Descriptor ID Copy Control. The DMA_CFG.DESCIDCPY specifies when to copy the initial descriptor pointer to the DMA_DSCPTR_PRV register. A bus write to the DMA_CFG register to clear the DMA_CFG.EN bit, causes the DMA to immediately use the new value of the DMA_CFG.DESCIDCPY bit. To preserve consistency (if required by application), match the new value of this bit to the previous value.	
		0	Never Copy
		1	Copy on Work Unit Complete
24 (R/W)	TOVEN	Trigger Overrun Error Enable. A trigger overrun occurs if more than one trigger was received before the DMA reached the trigger wait state. If DMA_CFG.TOVEN is set, a trigger overrun causes the DMA to flag an error. In cases where a trigger overrun is not a problem, clearing DMA_CFG.TOVEN prevents the overrun from causing an error and halting the DMA. The DMA_CFG.TOVEN operates independently of the DMA_CFG.TWAIT bit selection.	
		0	Ignore Trigger Overrun
		1	Error on Trigger Overrun

Table 12-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
23:22 (R/W)	TRIG	Generate Outgoing Trigger. The DMA_CFG.TRIG selects whether the DMA issues an outgoing trigger, based on the work unit counter values. In one-dimensional mode, the only options are to trigger at the end of the whole work unit (trigger when DMA_XCNT_CUR reaches 0) or not to trigger at all. If in one-dimensional addressing mode, programming the DMA_CFG.TRIG bit field to trigger when DMA_YCNT_CUR reaches 0 (or to reserved) causes the DMA to flag a configuration error. In two-dimensional addressing mode, the trigger options are: at the end of each row of the inner loop (when DMA_XCNT_CUR reaches 0), only after completing the whole work unit (when DMA_YCNT_CUR reaches 0), or no trigger. If in two-dimensional mode and set to trigger when DMA_XCNT_CUR reaches 0, the DMA also issues a trigger at the end of the work unit. If in two-dimensional addressing mode, programming DMA_CFG.TRIG to reserved causes the DMA to flag a configuration error. If DMA_CFG.TRIG is non-zero and the peripheral issues a finish command, the DMA issues a trigger after the finish procedure is complete. For more information about trigger generation timing, see the trigger section of the DMA functional description.	
		0	Never assert Trigger
		1	Trigger when XCNTCUR reaches 0
		2	Trigger when YCNTCUR reaches 0
		3	Reserved

Table 12-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/W)	INT	Generate Interrupt. The DMA_CFG.INT bit field selects whether an interrupt goes to the core based on work unit status or a peripheral interrupt request. For one-dimensional mode, setting the DMA_CFG.INT bits to generate an interrupt when the DMA_YCNT_CUR register reaches 0 causes the DMA to flag a configuration error. The peripheral interrupt setting lets the DMA generate the last grant indication and to accept and or forward the peripheral interrupt command. The peripheral interrupt selection applies only to the DMA_CFG.FLOW bits set for stop and the DMA_CFG.WNR bits set for memory read. If the DMA_CFG.INT is set for interrupt on count completion (DMA_XCNT_CUR or DMA_YCNT_CUR reach 0) and the peripheral issues a finish command, the DMA issues an interrupt after the finish procedure is complete. For more information, see the sections on interrupt generation and peripheral control in the DMA functional description.	
		0	Never assert Interrupt
		1	Interrupt when X Count Expires
		2	Interrupt when Y Count Expires
		3	Peripheral Interrupt
18:16 (R/W)	NDSIZE	Next Descriptor Set Size. The DMA_CFG.NDSIZE bit field specifies the number of descriptor elements in memory to load during the next descriptor fetch. The DMA loads the descriptors in a specific order. The descriptor set contains the next descriptor pointer when it is a descriptor list. The descriptor set does not contain the next descriptor pointer when it is a descriptor array.	
		0	Fetch one Descriptor Element
		1	Fetch two Descriptor Elements
		2	Fetch three Descriptor Elements
		3	Fetch four Descriptor Elements
		4	Fetch five Descriptor Elements
		5	Fetch six Descriptor Elements
		6	Fetch seven Descriptor Elements
		7	Reserved

Table 12-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	TWAIT	Wait for Trigger. The DMA_CFG.TWAIT controls whether the DMA waits for an incoming trigger from another channel or user. If DMA_CFG.TWAIT is set, the DMA enters the wait state before starting the next work unit, including descriptor fetch when in descriptor mode. Do not use the wait for trigger control for descriptor-on-demand mode which causes an error. For more information, see the trigger section of the DMA functional description.	
		0	Begin Work Unit Automatically (No Wait)
		1	Wait for Trigger (Halt before Work Unit)
14:12 (R/W)	FLOW	Next Operation. The DMA_CFG.FLOW selects descriptor handling options.	
		0	STOP - Stop See the Stop Flow Mode section
		1	AUTO - Autobuffer See the Autobuffer Flow Mode section
		2	Reserved
		3	Reserved
		4	DSCL - Descriptor List See the Descriptor List Mode section
		5	DSCA - Descriptor Array See the Descriptor Array Mode section
		6	Descriptor On-Demand List See the Descriptor List Mode section
		7	Descriptor On Demand Array See the Descriptor On Demand section
10:8 (R/W)	MSIZE	Memory Transfer Word Size. The DMA_CFG.MSIZE bits select memory transfer sizes of 8-, 16-, 32-, 64-, 128-, or 256-bit words. The transfer start address (DMA_ADDRSTART) and transfer increment values (DMA_XMOD, and, if needed, DMA_YMOD) must be a multiple of the memory transfer unit size.	
		0	1 Byte
		1	2 Bytes
		2	4 Bytes
		3	8 Bytes
		4	16 Bytes
		5	32 Bytes

Table 12-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6:4 (R/W)	PSIZE	Peripheral Transfer Word Size. The DMA_CFG.PSIZE bits select peripheral transfer sizes as 8, 16, 32, or 64 bits wide. Each request and grant results in a single peripheral access. There are no bursts on the peripheral bus, so the DMA_CFG.PSIZE selection must be less than, or equal to, the width of the bus. If the selection is greater than the bus width, a configuration error occurs. The peripheral bus of the processor is dedicated to DMA and peripheral accesses.	
		0	1 Byte
		1	2 Bytes
		2	4 Bytes
		3	8 Bytes
3 (R/W)	CADDR	Use Current Address. When the DMA_CFG.CADDR bit is cleared, the DMA loads the DMA_ADDRSTART register on the first access of the work unit. When the DMA_CFG.CADDR bit is set, the DMA uses the DMA_ADDR_CUR register value for the starting address for the work unit and writes the same value to the DMA_ADDRSTART register. This operation permits continuation of a previous work unit. When DMA uses this mode, the fetched start address value (as part of the descriptor set at the end of a descriptor list or array) is ignored.	
		0	Load Starting Address
		1	Use Current Address

Table 12-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	SYNC	Synchronize Work Unit Transitions. Setting the DMA_CFG.SYNC bit clears the DMA FIFO and pointers before starting the first work unit of a work unit chain. When the transfer direction is memory read/transmit (DMA_CFG.WNR =0), the DMA waits until all data transmits to a peripheral before moving on to the next work unit, clearing the FIFO and pointers. When the transfer direction is memory write/receive (DMA_CFG.WNR =1), the DMA ignores the DMA_CFG.SYNC bit value after processing the first work unit of a work unit chain. As a channel can receive data when turned on but idle, data from the peripheral can still be in the FIFO even though the channel was not programmed. When the DMA_CFG.SYNC bit field is set at the beginning of a work unit chain (during the first work unit), the DMA clears the FIFO, erasing the data put into the FIFO while the channel was idle. Syncing lets programs change the DMA_CFG.PSIZE between individual work units and (in some cases) work unit chains. The sync resets the pointers in the FIFO, preventing misaligned FIFO access. Programs can change the DMA_CFG.MSIZE field between consecutive work units, independent of the DMA_CFG.SYNC bit setting. Syncing also permits changes to transfer direction. Because the data in the FIFO is eliminated, the data that went into the FIFO from one direction (transmit or receive) is not sent back in the other direction after the direction change.	
		0	No Synchronization
		1	Synchronize Channel
1 (R/W)	WNR	Write/Read Channel Direction. The DMA_CFG.WNR selects receive (write to memory) or transmit (read from memory) channel direction.	
		0	Transmit (Read from memory)
		1	Receive (Write to memory)

Table 12-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	DMA Channel Enable. The DMA_CFG.EN enables the selected DMA channel. When a peripheral DMA channel is enabled, data requests from the peripheral denote DMA requests. When a channel is disabled, the DMA unit ignores the peripheral data request and passes it directly to the system event controller. To avoid unexpected results, enable the DMA channel before enabling the peripheral, and disable the peripheral before disabling the DMA channel. A transition of the DMA_CFG.EN bit from 0 to 1 creates a hard reset of all internal counters and states, including the DMA_STAT register. (All other register values remain unaffected.) A transition from 1 to 0 maintains all counters and registers for reading and analysis. Note that if a descriptor loads when this bit is cleared (see DMA_CFG.FLOW field), the DMA transitions to the off or idle state after the descriptor load is complete.	
		0	Disable
		1	Enable

Current Descriptor Pointer Register

The DMA_DSCPTR_CUR register contains the memory address for the next descriptor to be loaded. The DMA_DSCPTR_CUR register is read/write prior to enabling the channel, but is read-only after enabling the channel. For DMA_CFG.FLOW mode settings that involve descriptor fetches, this register is used to read descriptors into appropriate MMRs before a work unit begins. For descriptor list mode, the DMA_DSCPTR_CUR register is initialized from the DMA_DSCPTR_NXT register before fetching each descriptor set. Then, the address in the DMA_DSCPTR_CUR register increments as each descriptor is read.

When the entire descriptor set has been read, the DMA_DSCPTR_CUR register contains this value:

DMA_DSCPTR_CUR = Descriptor Start Address + Descriptor Size (# of elements)

For descriptor array mode, the DMA_DSCPTR_CUR register, and not the DMA_DSCPTR_NXT register, must be programmed by a MMR access before starting DMA operation.

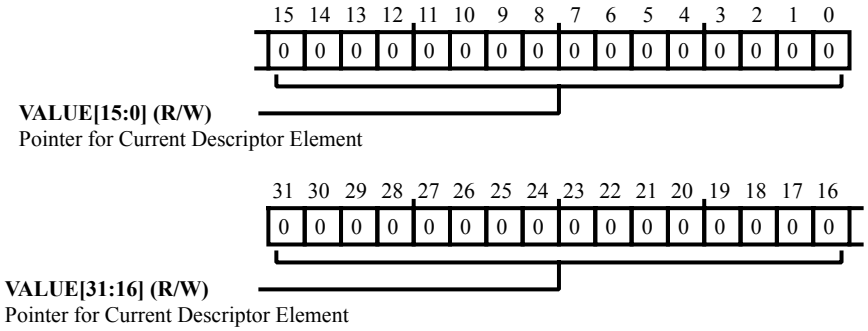

Figure 12-12: DMA_DSCPTR_CUR Register Diagram

Table 12-22: DMA_DSCPTR_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Pointer for Current Descriptor Element. The DMA_DSCPTR_CUR.VALUE bit field contains the memory address for the next descriptor to be loaded.

Pointer to Next Initial Descriptor Register

The DMA_DSCPTR_NXT register specifies the start location of the next descriptor set, which begins when the DMA activity specified by the current descriptor set completes. This register is read/write prior to enabling the channel, but is read-only after enabling channel.

The DMA_DSCPTR_NXT register is only used in descriptor list mode. At the start of a descriptor fetch in this mode, the DMA_DSCPTR_NXT register is copied into the DMA_DSCPTR_CUR register. During descriptor fetch, the DMA increments the DMA_DSCPTR_CUR register value after reading each element of the descriptor set.

In descriptor list mode, the DMA_DSCPTR_NXT register (not the DMA_DSCPTR_CUR register) must be programmed directly through MMR access, before the DMA operation is started. In descriptor array mode, the DMA disregards the DMA_DSCPTR_NXT register and uses the DMA_DSCPTR_CUR register to control descriptor fetch.

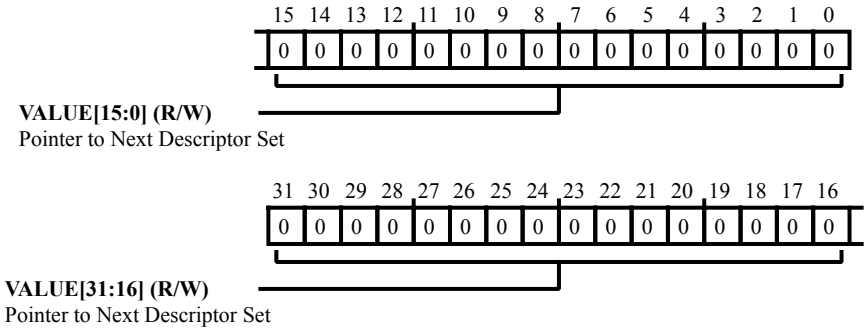

Figure 12-13: DMA_DSCPTR_NXT Register Diagram

Table 12-23: DMA_DSCPTR_NXT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Pointer to Next Descriptor Set. The DMA_DSCPTR_NXT.VALUE bit field specifies the start location of the next descriptor set.

Previous Initial Descriptor Pointer Register

The DMA_DSCPTR_PRV register contains the initial descriptor pointer for the previous work unit. If DMA_CFG.DESCIDCPY is set, the DMA copies the initial descriptor pointer to DMA_DSCPTR_PRV after the work unit completes. Otherwise, the value is not updated.

To indicate an overrun, bit 0 of the DMA_DSCPTR_PRV register is used as a previous descriptor pointer overrun (PDPO) status bit. Due to aligned addressing combined with all descriptors being 32 bits in width, bits 0 and 1 of all descriptor pointers must be zero. Otherwise, an alignment error occurs when used for descriptor fetches. As a result, bit 1 and 0 of the DMA_DSCPTR_PRV register can be used for status. For more information, see the section on descriptor pointer capture in the DMA functional description.

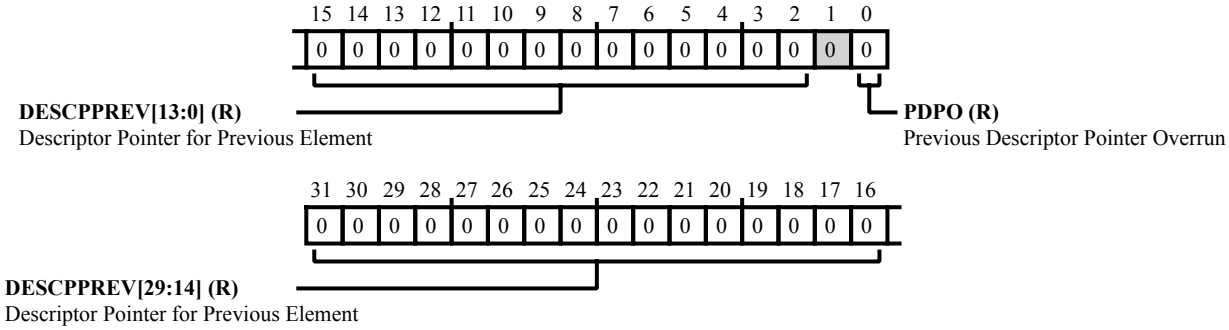

Figure 12-14: DMA_DSCPTR_PRV Register Diagram

Table 12-24: DMA_DSCPTR_PRV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:2 (R/NW)	DESCPPREV	Descriptor Pointer for Previous Element. The DMA_DSCPTR_PRV.DESCPPREV bit field contains the initial descriptor pointer for the previous work unit.	
0 (R/NW)	PDPO	Previous Descriptor Pointer Overrun. The DMA_DSCPTR_PRV.PDPO bit signifies an alignment error. Due to aligned addressing combined with all descriptors being 32 bits in width, bits 0 and 1 of all descriptor pointers must be zero. Otherwise, an alignment error would occur when used for descriptor fetches.	
		0	No error occurred
		1	Error occurred

Status Register

The DMA_STAT register indicates the status of DMA work units, the FIFO, errors, interrupts, and triggers.

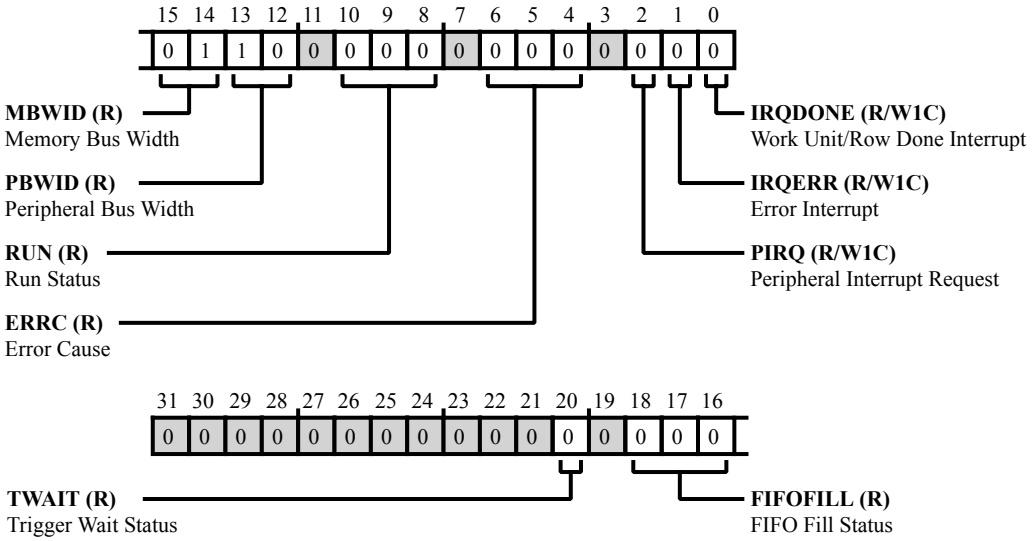

Figure 12-15: DMA_STAT Register Diagram

Table 12-25: DMA_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/NW)	TWAIT	Trigger Wait Status. The DMA_STAT.TWAIT indicates whether the DMA has or has not received a trigger. This bit is set until it reaches the next wait state. At that point, the bit is cleared, the DMA stops processing that work unit, and the following work unit processes. The DMA does not distinguish between one or more triggers received.	
		0	No trigger received
		1	Trigger received
18:16 (R/NW)	FIFOFILL	FIFO Fill Status. The DMA_STAT.FIFOFILL reports the quantity of data in the FIFO relative to available space.	
		0	Empty
		1	Empty < FIFO = 1/4 Full
		2	1/4 Full < FIFO = 1/2 Full
		3	1/2 Full < FIFO = 3/4 Full
		4	3/4 Full < FIFO = Full
		5	Reserved

Table 12-25: DMA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		6	Reserved
		7	Full
15:14 (R/NW)	MBWID	Memory Bus Width. The DMA_STAT.MBWID indicates the width of the memory bus connected to this DMA.	
		0	2 Bytes
		1	4 Bytes
		2	8 Bytes
		3	16 Bytes
13:12 (R/NW)	PBWID	Peripheral Bus Width. The DMA_STAT.PBWID bit field indicates the width of the peripheral bus connected to this DMA.	
		0	1 Byte
		1	2 Bytes
		2	4 Bytes
		3	8 Bytes
10:8 (R/NW)	RUN	Run Status. The DMA_STAT.RUN bit field reports the DMA's current operational state. If the DMA is in idle or stop state, the DMA_CFG.EN bit is either 0 or 1. This bit field does not clear when a transition of the DMA_CFG.EN bit from 0 to 1 occurs. The DMA_STAT.RUN clears automatically when the DMA completes.	
		0	Idle/Stop State
		1	Descriptor Fetch
		2	Data Transfer
		3	Waiting for Trigger
		4	Waiting for Write ACK/FIFO Drain to Peripheral
		5	Reserved
		6	Reserved
		7	Reserved
6:4 (R/NW)	ERRC	Error Cause. When an interrupt request error occurs (DMA_STAT.IRQERR), the DMA updates DMA_STAT.ERRC to identify the type of error. For more information, see the errors section of the DMA functional description.	

Table 12-25: DMA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	Configuration Error
		1	Illegal Write Occurred While Channel Running
		2	Address Alignment Error
		3	Memory Access or Fabric Error
		4	Reserved
		5	Trigger Overrun
		6	Bandwidth Monitor Error
		7	Reserved
2 (R/W1C)	PIRQ	Peripheral Interrupt Request. The DMA_STAT.PIRQ bit indicates a peripheral interrupt. Programs can use the DMA_STAT.PIRQ bit status to help determine which DMA asserted the interrupt. This bit also helps to distinguish between an interrupt caused by the state of the work unit and an interrupt caused by the peripheral.	
		0	No Interrupt
		1	Interrupt Signaled by Peripheral
1 (R/W1C)	IRQERR	Error Interrupt. The DMA_STAT.IRQERR indicates that the DMA has detected a documented rule violation during DMA programming or operation. The DMA cannot, however, flag all possible programming or operation issues to indicate errors. Programmers can use DMA_STAT.IRQERR to help determine which DMA issued the error interrupt. The DMA_STAT.IRQERR does not clear a transition of the DMA_CFG.EN bit from 0 to 1. Clear the DMA_STAT.IRQERR with a write-1-to-clear operation prior to the DMA_CFG.EN transition for the fields to be reset.	
		0	No Error
		1	Error Occurred
0 (R/W1C)	IRQDONE	Work Unit/Row Done Interrupt. The DMA_STAT.IRQDONE indicates the DMA has detected the completion of a work unit or row (inner loop count) and has issued an interrupt. Programs can use the DMA_STAT.IRQDONE status to help determine which DMA asserted the interrupt. Programs can also use these bits to help distinguish between an interrupt based on the state of the work unit and an interrupt made by the peripheral. For more information, see the interrupts section of the DMA functional description.	
		0	Inactive
		1	Active

Inner Loop Count Start Value Register

For 2D DMA, the DMA_XCNT register contains the inner loop count. This value selects the number of DMA_CFG.MSIZE size data transfers that make up the length of a row. For 1D DMA, the DMA_XCNT register specifies the number of DMA_CFG.MSIZE size data transfers for the entire work unit. The DMA_XCNT register is read/write prior to enabling the channel, but is read-only after enabling channel. Note that the DMA generates a configuration error if this register is 0x0 when a work unit begins.

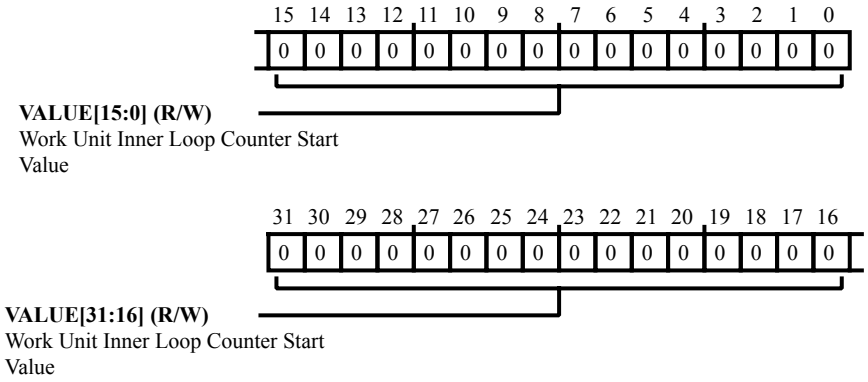

Figure 12-16: DMA_XCNT Register Diagram

Table 12-26: DMA_XCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit Inner Loop Counter Start Value. For 2D DMA, the DMA_XCNT.VALUE bit field contains the inner loop count. For 1D DMA, DMA_XCNT.VALUE specifies the number of DMA_CFG.MSIZE size data transfers for the entire work unit.

Current Count(1D) or intra-row XCNT (2D) Register

For 1D DMA, the DMA loads the DMA_XCNT_CUR register from the DMA_XCNT register at the beginning of each work unit. For 2D DMA, the DMA loads the DMA_XCNT_CUR register from the DMA_XCNT register after the end of each row. The DMA decrements the value in DMA_XCNT_CUR register each time a DMA_CFG.MSIZE size data transfer occurs. When the count in DMA_XCNT_CUR register expires, the work unit is complete. In 2D DMA, the DMA_XCNT_CUR value is 0 only when the entire transfer is complete.

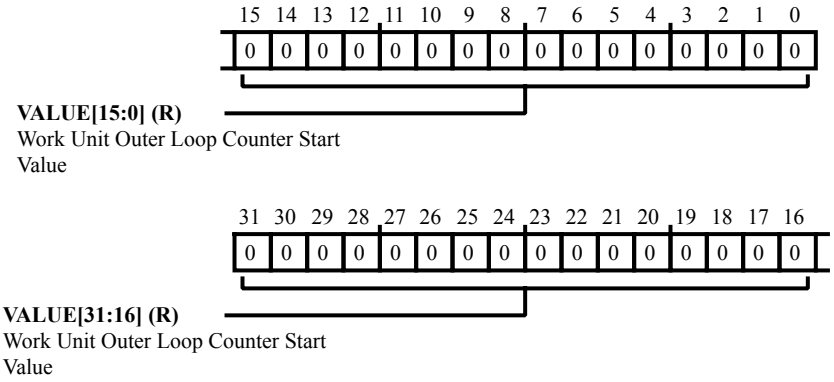

Figure 12-17: DMA_XCNT_CUR Register Diagram

Table 12-27: DMA_XCNT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Work Unit Outer Loop Counter Start Value. For 1D DMA, the DMA_XCNT_CUR.VALUE bit field holds the DMA_XCNT value at the beginning of each work unit. For 2D DMA, the DMA_XCNT_CUR.VALUE bit field holds the DMA_XCNT value after the end of each row.

Inner Loop Address Increment Register

The DMA_XMOD register contains a signed, two's-complement byte address increment. In 1D DMA, this increment is the stride that is applied after each DMA_CFG.MSIZE size data transfer. The DMA_XMOD register is read/write prior to enabling the channel, but is read-only after enabling the channel.

The DMA_XMOD register value is specified in bytes, regardless of the work unit size. In 2D DMA, this increment is applied after each DMA_CFG.MSIZE size data transfer in the inner loop, up to but not including the last DMA_CFG.MSIZE size data transfer in each inner loop. After the last DMA_CFG.MSIZE size data transfer in each inner loop, the DMA_YMOD register is applied instead, including the last DMA_CFG.MSIZE size data transfer of a work unit.

The DMA_XMOD field can be set to 0. In this case, DMA is performed repeatedly to or from the same address. This approach can be useful for transferring data between a data register and an external memory-mapped peripheral.

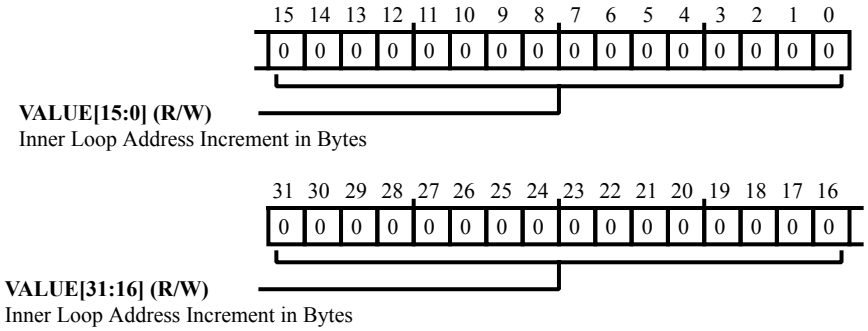

Figure 12-18: DMA_XMOD Register Diagram

Table 12-28: DMA_XMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Inner Loop Address Increment in Bytes. The DMA_XMOD.VALUE bit field contains a signed, two's-complement byte address increment.

Outer Loop Count Start Value (2D only) Register

For 2D DMA, the DMA_YCNTregister contains the outer loop count. This register is not used in 1D DMA mode. The DMA_YCNT register specifies the number of rows in the outer loop of a 2D DMA sequence. The DMA_YCNT register is read/write prior to enabling the channel, but is read-only after enabling channel. Note that the DMA generates a configuration error if this register is 0x0 when a work unit begins.

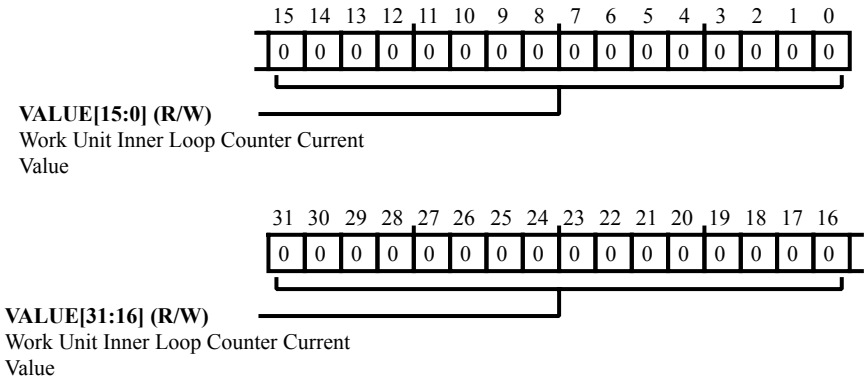

Figure 12-19: DMA_YCNT Register Diagram

Table 12-29: DMA_YCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit Inner Loop Counter Current Value. For 2D DMA, the DMA_YCNT.VALUE bit field contains the outer loop count.

Current Row Count (2D only) Register

For 2D DMA, the DMA loads the DMA_YCNT_CUR register from the DMA_YCNT register at the beginning of each 2D DMA session. The DMA_YCNT_CUR register is not used for 1D DMA. The DMA decrements the DMA_YCNT_CUR register each time the DMA_XCNT_CUR register expires during 2D DMA operation, signifying the completion of an entire row transfer.

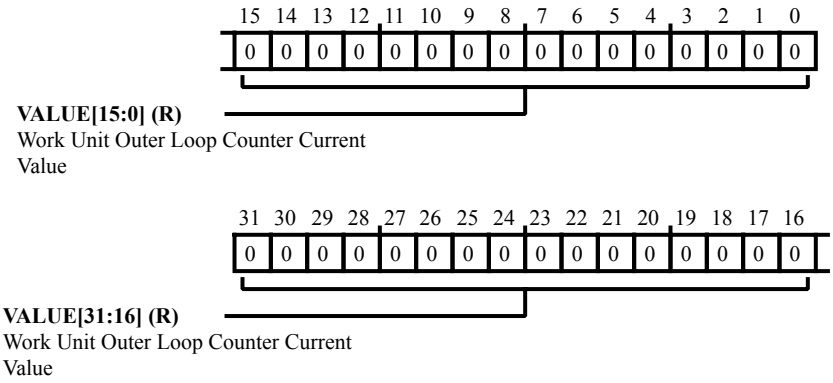

Figure 12-20: DMA_YCNT_CUR Register Diagram

Table 12-30: DMA_YCNT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Work Unit Outer Loop Counter Current Value. For 2D DMA, the DMA_YCNT_CUR.VALUE bit field holds the value from the DMA_YCNT register at the beginning of each 2D DMA session.

Outer Loop Address Increment (2D only) Register

The DMA_YMOD register contains a signed, two's-complement value. This byte address increment is applied after each decrement of the DMA_YCNT_CUR register. The value is the offset between the last word of one row and the first word of the next row. Note that DMA_YMOD is specified in bytes, regardless of the work unit size. The DMA_YMOD register is read/write prior to enabling the channel, but is read-only after enabling the channel.

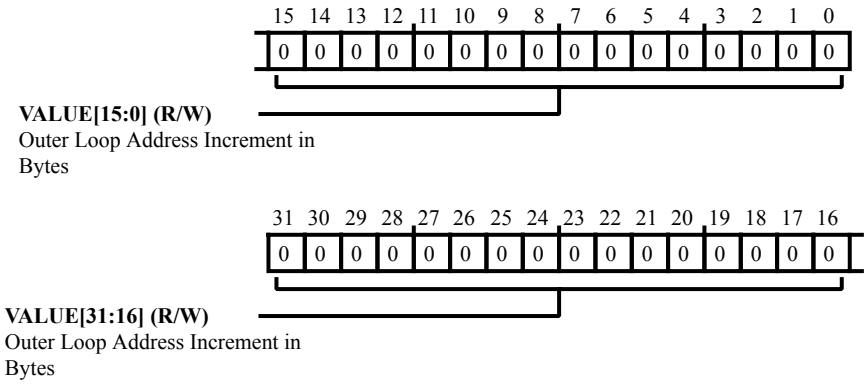

Figure 12-21: DMA_YMOD Register Diagram

Table 12-31: DMA_YMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Outer Loop Address Increment in Bytes. The DMA_YMOD.VALUE bit field contains a signed, two's-complement value.

13 General-Purpose Ports (PORT)

This section describes general-purpose ports, pin multiplexing, general-purpose input/output (GPIO) functionality, and pin interrupts.

The general-purpose ports provide the following three functions.

- Pin multiplexing scheme
- GPIO functionality
- Pin interrupts

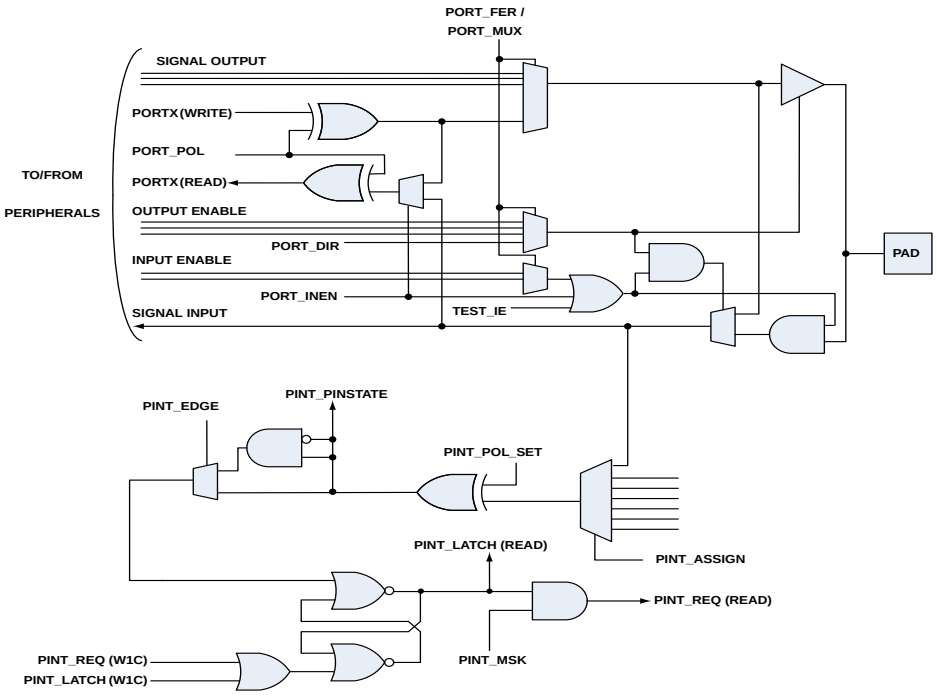

Figure 13-1: Simplified GPIO and Pin Interrupt Signal Flow

PORT Features

The PORTs include the following features:

- Up to 91 general-purpose I/O (GPIO) pins
- Input mode, output mode, and open-drain mode of GPIO operation
- Port multiplexing controlled by individual pin-per-pin base
- No glue hardware required for unused pins
- Five interrupt channels dedicated to pin interrupts
- All port pins provide interrupt functionality
- Byte-wide pin-to-interrupt assignment

PORT Functional Description

Every port pin can operate in GPIO mode. GPIO mode is the default, after reset. The port-specific PORT_FER register enables GPIO mode. Every port has a dedicated set of MMR registers that control the GPIO functionality. Every bit in these registers represents a certain GPIO pin of the specific port.

Input mode, output mode, and open-drain mode of GPIO operation

The default mode of every GPIO pin, after reset, is input mode. But, the input drivers are not enabled. To enable any GPIO input drivers, set the corresponding bits in the input enable register.

The GPIO output drivers are enabled by setting the corresponding bits in the direction registers.

The PORT can use every GPIO in open-drain mode by clearing the respective bit in the PORT_DATA register or the PORT_DATA_CLR register. Then, set the 1 bit in the PORT_INEN register. Read from the PORT_DATA register. Then, return the status from the pin. Do not return the state of the internal flip-flop.

Port multiplexing controlled by individual pin-per-pin base

Each port has two dedicated MMRs that control the port multiplexing, the 16-bit function enable (PORT_FER) registers and the 32-bit port multiplexing (PORT_MUX) registers.

All port pins provide interrupt functionality

The pin interrupts are completely decoupled from GPIO functionality. Pins are connected to the PINTx module and then to the system event controller. Pay attention to how the pins are assigned to the PINTx modules.

ADSP-CM40x PORT Register List

The PORT module (PORT) regulates the use of the multiplexable processor pins. Every port pin can operate in general-purpose I/O (GPIO) mode or as an alternate function. This GPIO operation is the default after processor reset and is controlled by a set of registers that control GPIO functionality. Every bit in these registers represents a certain GPIO pin of a specific port. For more information on PORT functionality, see the PORT register descriptions.

Table 13-1: ADSP-CM40x PORT Register List

Name	Description
PORT_DATA	Port x GPIO Data Register
PORT_DATA_CLR	Port x GPIO Data Clear Register
PORT_DATA_SET	Port x GPIO Data Set Register
PORT_DATA_TGL	Port x GPIO Output Toggle Register
PORT_DIR	Port x GPIO Direction Register
PORT_DIR_CLR	Port x GPIO Direction Clear Register
PORT_DIR_SET	Port x GPIO Direction Set Register
PORT_FER	Port x Function Enable Register
PORT_FER_CLR	Port x Function Enable Clear Register
PORT_FER_SET	Port x Function Enable Set Register
PORT_INEN	Port x GPIO Input Enable Register
PORT_INEN_CLR	Port x GPIO Input Enable Clear Register
PORT_INEN_SET	Port x GPIO Input Enable Set Register
PORT_LOCK	Port x GPIO Lock Register
PORT_MUX	Port x Multiplexer Control Register
PORT_POL	Port x GPIO Polarity Invert Register
PORT_POL_CLR	Port x GPIO Polarity Invert Clear Register
PORT_POL_SET	Port x GPIO Polarity Invert Set Register

ADSP-CM40x PADS Register List

The PADS controls signal hysteresis and other system interface signal features for a number of module interfaces.The PADS also provides drive strength control, but this feature is currently unsupported.

Table 13-2: ADSP-CM40x PADS Register List

Name	Description
PADS_PCFG0	Peripheral Configuration0 Register

ADSP-CM40x PINT Register List

The Pin Interrupt module (PINT) controls the pin-to-interrupt assignment in a byte-wide manner. The pin-interrupt assignment registers do not consist of 32 individual bits. They consist of four control bytes, each functioning as a multiplexer control. For more information, see the PINT register descriptions.

All PINT registers are 32 bits wide and can be accessed by 32-bit load/store instructions. They also support 16-bit operation where the upper 16 bits are ignored and the application uses the lower 16 bits only. Consequently, all

PINT registers support 32-bit accesses as well as 16-bit accesses for the lower half words. Applications may use faster 16-bit accesses as long as they do not require functionality of upper register halves.

Table 13-3: ADSP-CM40x PINT Register List

Name	Description
PINT_ASSIGN	Pint Assign Register
PINT_EDGE_CLR	Pint Edge Clear Register
PINT_EDGE_SET	Pint Edge Set Register
PINT_INV_CLR	Pint Invert Clear Register
PINT_INV_SET	Pint Invert Set Register
PINT_LATCH	Pint Latch Register
PINT_MSK_CLR	Pint Mask Clear Register
PINT_MSK_SET	Pint Mask Set Register
PINT_PINSTATE	Pint Pinstate Register
PINT_REQ	Pint Request Register

ADSP-CM40x PINT Interrupt List

Table 13-4: ADSP-CM40x PINT Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
18	PINT0_BLOCK	PINT0 Block Interrupt Generated	Level	
19	PINT1_BLOCK	PINT1 Block Interrupt Generated	Level	
20	PINT2_BLOCK	PINT2 Block Interrupt Generated	Level	
21	PINT3_BLOCK	PINT3 Block Interrupt Generated	Level	
22	PINT4_BLOCK	PINT4 Block Interrupt Generated	Level	

ADSP-CM40x PINT Trigger List

Table 13-5: ADSP-CM40x PINT Trigger List Masters

Trigger ID	Name	Description	Sensitivity
10	PINT0_BLOCK	PINT0 Block Interrupt Generated	Level
11	PINT1_BLOCK	PINT1 Block Interrupt Generated	Level
12	PINT2_BLOCK	PINT2 Block Interrupt Generated	Level
13	PINT3_BLOCK	PINT3 Block Interrupt Generated	Level
14	PINT4_BLOCK	PINT4 Block Interrupt Generated	Level

Table 13-6: ADSP-CM40x PINT Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

PORT Definitions

This section provides definitions relating to the GPIO ports.

x (PORTx)

The naming convention for bits uses a lowercase “x” to represent one of the existing ports alphabetically named beginning with A,B,C, and so on. For example, the name `PORTx_REG` represents any one or all of `PORTA_REG`, `PORTB_REG`, `PORTC_REG`, and so on. The bit name `Px0` represents `PA0`, `PB0`, and so on.

PORT Architectural Concepts

These sections describe in more detail how the PORT module connects externally to pins and internally to the MMR bus. Port groups are named alphabetically beginning with A.

- Internal Interfaces
- External Interfaces
- GPIO Functionality
- Port Multiplexing Control

Internal Interfaces

All MMR registers of the pin multiplexing, GPIO and pin interrupt control blocks can be accessed through the MMR bus. There is no DMA support. Every one of the pin interrupt modules has its own and dedicated interrupt request output signal that connects directly to the interrupt or event controller.

External Interfaces

The pin multiplexing hardware can be seen as a layer between the on-chip peripherals and the pads of the silicon. The PORT unit controls all port groups.

GPIO Functionality

By default, the PORT sets every GPIO to input mode. The input drivers are not enabled, which avoids the need for unnecessary current sinks and the external pulling of resistors on unused or *do not care* pins.

Input Mode

The default mode of every GPIO pin after reset is input mode, but the input drivers are not enabled. To enable any GPIO input drivers, set the corresponding bits in the input enable register (`PORT_INEN`). When enabled, a read

from the PORT_DATA register returns the logical state of the input pin. The input signal does not overwrite the state of the flip-flop used for the output case. Only software can alter the state. If the input driver is enabled, a write to the PORT_DATA register can alter the state of the flip-flop, but the change cannot be readback.

Output Mode

Any GPIO pin can be configured for output mode. The GPIO output drivers are enabled by setting the corresponding bits in the direction registers. The PORT implements direction registers as a pair of write-1-to-set (W1S) and write-1-to-clear (W1C) MMRs, called PORT_DIR_SET and PORT_DIR_CLR. Separate software threads can alter the direction of the signal flow of individual GPIO pins without mutually impacting other GPIOs on the same port.

Both PORT_DIR_SET and PORT_DIR_CLR registers return the same value when read and a logical 1 indicates an enabled output. The PORT_DATA registers control the state of output pins. A logical 0 drives the output low while a logical 1 drives the output high.

While writes to the PORT_DATA register can alter all GPIOs of a specific port at once, there are also a pair of W1S and W1C MMRs, called PORT_DATA_SET and PORT_DATA_CLR. These registers enable the manipulation of individual GPIO outputs. The state of the outputs can be obtained by reading the PORT_DATA registers. Because the state of the GPIO output can be controlled before the output driver is enabled, set or clear the flip-flop first to avoid any volatile levels on the output.

Open-Drain Mode

Every GPIO can also be used in open-drain mode. First, clear the respective bit in the PORT_DATA register or the PORT_DATA_CLR register. Then, set the one bit in the PORT_INEN register. Read from the PORT_DATA register, then return the status from the pin and do not return the state of the internal flip-flop.

By toggling the output driver through the PORT_DIR_SET and PORT_DIR_CLR register pair, the output signal can be pulled low or three-stated, as required. The polarity of the driven signal can be inverted when the internal flip-flop is set. When using a GPIO port in open-drain mode, take care to not exceed the V_{IH} operating condition associated with the respective pins.

Port Multiplexing Control

To configure pins properly, it is necessary to determine which bits in the PORT_FER and PORT_MUX register map to the pin of interest, and set them appropriately according to the desired function.

By default, after reset, all port pins are in GPIO input mode with their output and input drivers disabled. As a result, all unused port pins can be left unconnected. Disabled pins appear in high-impedance mode to external circuits. The pins are pulled low to internal logic.

Each port has two dedicated MMRs that control the port multiplexing, the 16-bit function enable (PORT_FER) registers and the 32-bit port multiplexing (PORT_MUX) registers.

NOTE: In this chapter, the naming convention for registers and bits omits the alphabetic group enumeration to refer to all or any of the port groups. For example, PORT_FER represents PORTA_FER, PORTB_FER, and so on. Likewise PORT_FER.PX1 represents PA1, PB1, and so on.

The function enable register specifies whether the pin is used as a GPIO pin, or another function, but does not specify what that other function is. Each bit in the 16-bit PORT_FER register represents one port pin. For example, bit 1 of the PORT_FER register configures the PA1 pin to GPIO operations mode when cleared. When set, one of the available peripheral functions becomes active.

Every pair of bits in the PORT_MUX register controls the multiplexing between the peripheral functions available to a pin. This field is a 2-bit field because some pins provide up to four options. The truth table of the bit field is identical to all family derivatives, regardless of the options that are available on the specific part.

Refer to the Signal Muxing table in the data sheet for the specific PORT_MUX settings.

PORT Event Control

The following sections describe event generation in the PORT module.

PORT Interrupt Signals

The pin interrupts are decoupled from GPIO functionality which has the following advantages.

- Flexible mapping scheme enables pins from up to four different ports to be grouped into one common interrupt scheme.
- Interrupts work on input and output pins regardless of whether in GPIO or functional mode.

The processor has a number of interrupt channels dedicated to pin interrupts. A set of hardware blocks named PINTx manage these channels. Every PINTx block can sense up to 32 GPIO pins as described in the following list and shown in the following figure.

The processor supports both 32-bit and 16-bit peripheral bus accesses to PINTx registers.

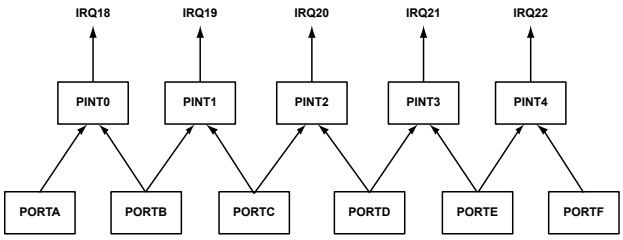

Figure 13-2: ADSP-CM40x GPIO to PINTx Assignment

Pins connect to the PINTx module and then to the system event controller. Pay attention to how the pins are assigned to the PINTx modules as shown in the ***PINTx Block Diagram***.

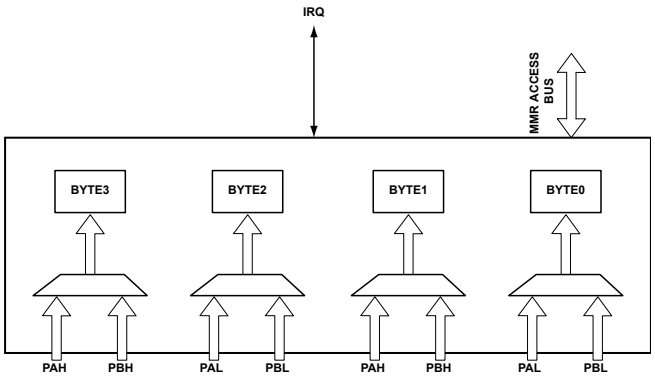

Figure 13-3: PINTx Block Diagram

The ports are subdivided into 8-bit half ports, with lower, and upper half 8-bit units. The PINT_ASSIGN registers control the 8-bit multiplexers shown in the ***PINTx Block Diagram***. The lower half units of eight pins can be forwarded to either byte 0 or byte 2 of either associated PINTx block. The upper half units can be forwarded to either byte 1 or byte 3 of the pin interrupts blocks, without further restrictions.

When a half port is assigned to a byte in any PINTx block, the state of the eight pins appears in the PINT_PINSTATE register. (The state does not depend on GPIO or function, input or output). When neither input or output drivers of the pin are enabled, the pin state is read as zero. The PINT_PINSTATE register reports the inverted state of the pin when the PINT_INV_SET register activates the signal inverter. The inverter can be enabled on an individual bit-by-bit basis. Every bit in the PINT_INV_SET/PINT_INV_CLR register pair represents a pin signal.

The PORT can generate an interrupt on an active high level of the signal or a rising edge of the signal. The default behavior is level-sensitivity. The PINTx block uses the PINT_EDGE_SET register to change the behavior to edge-sensitivity. By enabling the inverter using the PINT_INV_SET register, the PINTx block can alter interrupt behavior to trigger on active-low signals or falling edges.

The PINTx modules also assist when both signals must generate interrupts. If two different interrupt requests are required, the PINT_ASSIGN registers can route a signal to two different PINTx blocks, where one block inverts the signal and the other one does not. When both signal edges can report over the same interrupt, every signal can be routed through to different bit positions within a single PINTx block. Enable the inverted signal for either one. The servicing software routine can then detect from the PINT_LATCH register whether a falling, a rising, or both edges have occurred.

Regardless of whether using level-sensitive or edge-sensitive mode, the hardware always latches an interrupt. Latched signals can be read from the PINT_LATCH registers. Only a software or hardware reset clears the latches. To clear, W1C the PINT_REQ or the PINT_LATCH register. If the pin state does not change by the time the interrupt service routine returns, the interrupt is requested again when in level-sensitive mode.

Because every PINTx block groups up to 32-pin signals, the PINT_MSK_SET/ PINT_INV_CLR register pair can control which of the signals can request an interrupt at the system level. Software can interrogate the PINT_REQ register for signaling pins. The PINT_REQ bits represent a logical AND between the mask and the latch. When any of these bits is set, an interrupt is forwarded to the SIC controller.

All MMR registers in the pin interrupt module are 32 bits wide. Individual bits of the PINT_ASSIGN registers represent the associated pins. Nevertheless, the 32 bits can also be represented as four groups of eight pins. Each group can manage up to eight pins out of either the lower or an upper half of any associated port.

PORT Programming Model

The following sections description of the overall program model of the general-purpose ports.

GPIO Programming Model Flow (Part 1), ***GPIO Programming Model Flow (Part 2)***, and ***GPIO Programming Model Flow (Part 3)*** show the programming model of the general-purpose ports. This programming includes the GPIO input and output operation, open-drain mode, and the pin interrupt PINTx modules.

NOTE: These process flow diagrams connect where call-out letters appear. For example, call-out A on the ***GPIO Programming Model Flow (Part 1)*** diagram connects to call-out A on the ***GPIO Programming Model Flow (Part 2)*** diagram.

The following flowcharts describe the processes for setting up pins for different available functionality. Begin the process from the ***GPIO Programming Model Flow (Part 1)*** chart. The first decision affects the value of the PORT_FER register. Set this value to 1 for peripheral functions. For more information on setting up for peripheral functions, refer to the Port Multiplexing Control.

If the pin is to be a GPIO pin, a series of decisions must be made. Consider the following configuration registers: PORT_DATA, PORT_POL, PORT_DIR, and PORT_INEN. Depending on the type of GPIO pin desired, some configurations do not apply, and can have different meanings. The following paragraphs briefly describe the function of the different settings for each of the pin functions in GPIO modes: input, output, and open-drain. For all registers, use the SET or CLR versions.

For more detailed descriptions of the configurations, see PORT Register Descriptions.

For output mode, first use the PORT_DATA register to make all the pins low. Use the PORT_DIR register to define the direction of each pin (output). In this mode, the other registers are not significant. The ***GPIO Programming Model Flow (Part 1)*** chart shows this flow starting at label 2.

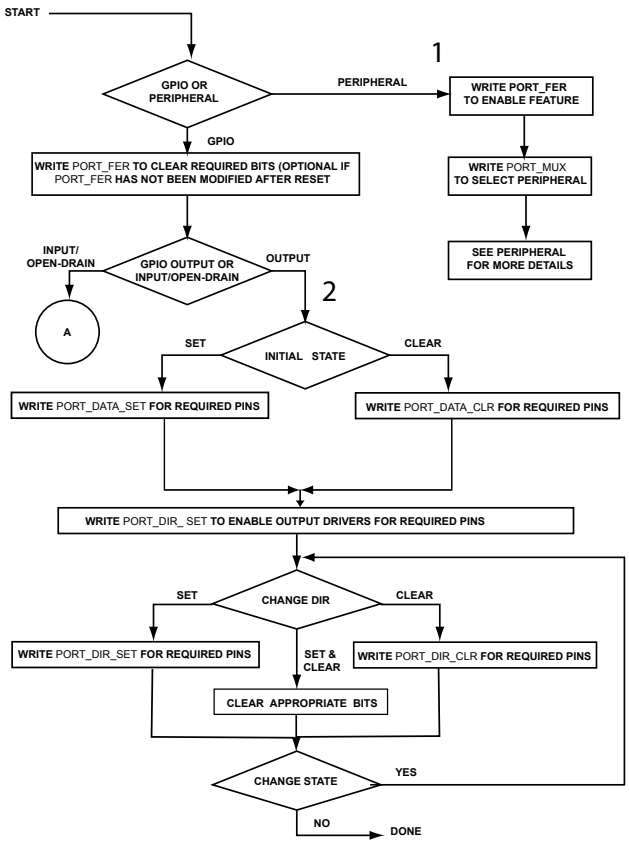

Figure 13-4: GPIO Programming Model Flow (Part 1)

For input mode, first decide the polarity for each pin using the PORT_POL register. The PORT uses the PORT_DIR register to define the appropriate pins for input. If interrupts are desired, configure the PINT module correctly. The ***GPIO Programming Model Flow (Part 3)*** chart shows these steps starting at B. Finally, the PORT uses the PORT_INEN register to enable the associated input drivers. The ***GPIO Programming Model Flow (Part 2)*** chart shows this entire flow starting at 3.

For open drain mode, set all pins low using the PORT_DATA register. Then, use the PORT_INEN register to enable the appropriate input drivers. Set the PORT_DIR register in this mode to indicate whether the pin is in an active state or not (active being 0). The ***GPIO Programming Model Flow (Part 2)*** chart shows this flow starting at 4.

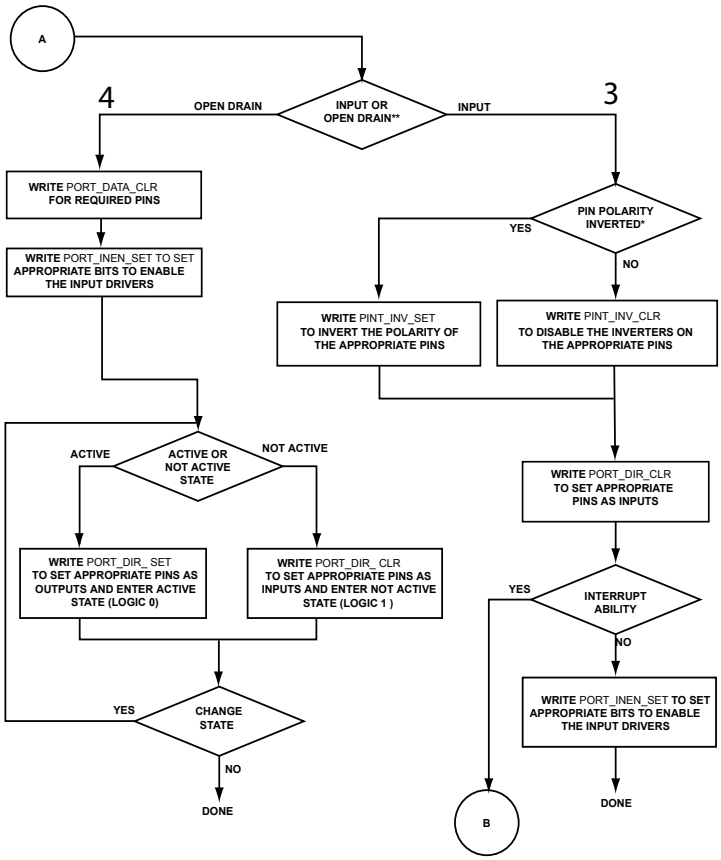

Figure 13-5: GPIO Programming Model Flow (Part 2)

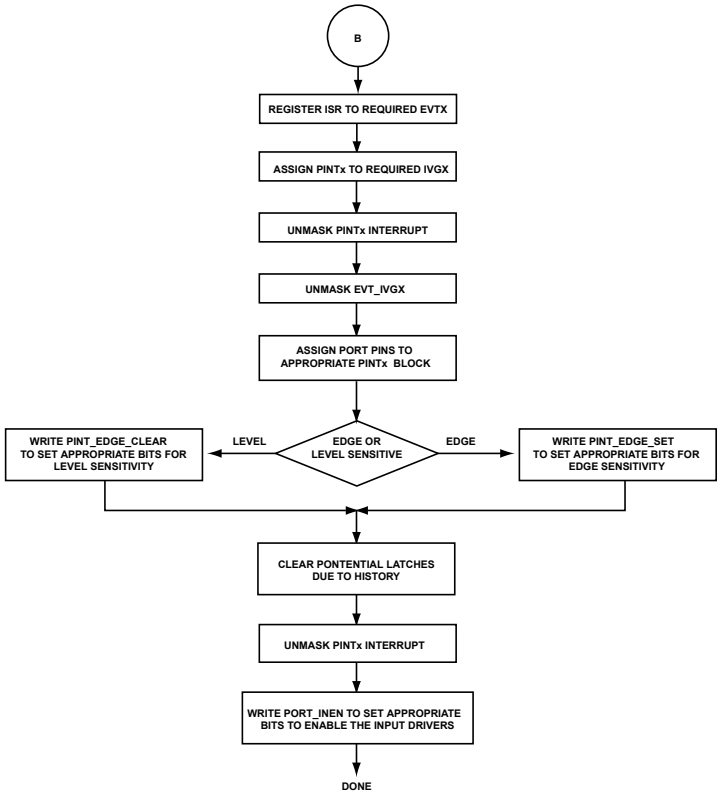

Figure 13-6: GPIO Programming Model Flow (Part 3)

ADSP-CM40x PORT Register Descriptions

General Purpose Input/Output (PORT) contains the following registers.

Table 13-7: ADSP-CM40x PORT Register List

Name	Description
PORT_DATA	Port x GPIO Data Register
PORT_DATA_CLR	Port x GPIO Data Clear Register
PORT_DATA_SET	Port x GPIO Data Set Register
PORT_DATA_TGL	Port x GPIO Output Toggle Register
PORT_DIR	Port x GPIO Direction Register
PORT_DIR_CLR	Port x GPIO Direction Clear Register
PORT_DIR_SET	Port x GPIO Direction Set Register
PORT_FER	Port x Function Enable Register
PORT_FER_CLR	Port x Function Enable Clear Register
PORT_FER_SET	Port x Function Enable Set Register

Table 13-7: ADSP-CM40x PORT Register List (Continued)

Name	Description
PORT_INEN	Port x GPIO Input Enable Register
PORT_INEN_CLR	Port x GPIO Input Enable Clear Register
PORT_INEN_SET	Port x GPIO Input Enable Set Register
PORT_LOCK	Port x GPIO Lock Register
PORT_MUX	Port x Multiplexer Control Register
PORT_POL	Port x GPIO Polarity Invert Register
PORT_POL_CLR	Port x GPIO Polarity Invert Clear Register
PORT_POL_SET	Port x GPIO Polarity Invert Set Register

Port x GPIO Data Register

The PORT_DATA register operates differently for port bits/pins, depending on whether the bit/pin is in output mode or input mode. In both modes, a set bit in the PORT_DATA register corresponds to a signal high on a GPIO pin, and a cleared bit in the PORT_DATA register corresponds to a signal low on a GPIO pin.

The PORT_DATA, PORT_DATA_SET, and PORT_DATA_CLR registers control the state of GPIO pins in output mode. To enable output mode (and output drivers), use the PORT_DIR_SET and PORT_DIR_CLR registers.

Writes to the PORT_DATA register affect the state of all pins of the port that are in output mode. To set or clear specific pins without impacting other pins of the port, use the PORT_DATA_SET and PORT_DATA_CLR registers.

When the GPIO pins are in input mode (input driver is enabled with the PORT_INEN register), reads from the PORT_DATA, PORT_DATA_SET, and PORT_DATA_CLR registers return the state of the respective GPIO pins.

Note that when the input driver is not enabled, reads from the PORT_DATA, PORT_DATA_SET, and PORT_DATA_CLR registers return the value previously written to the registers.

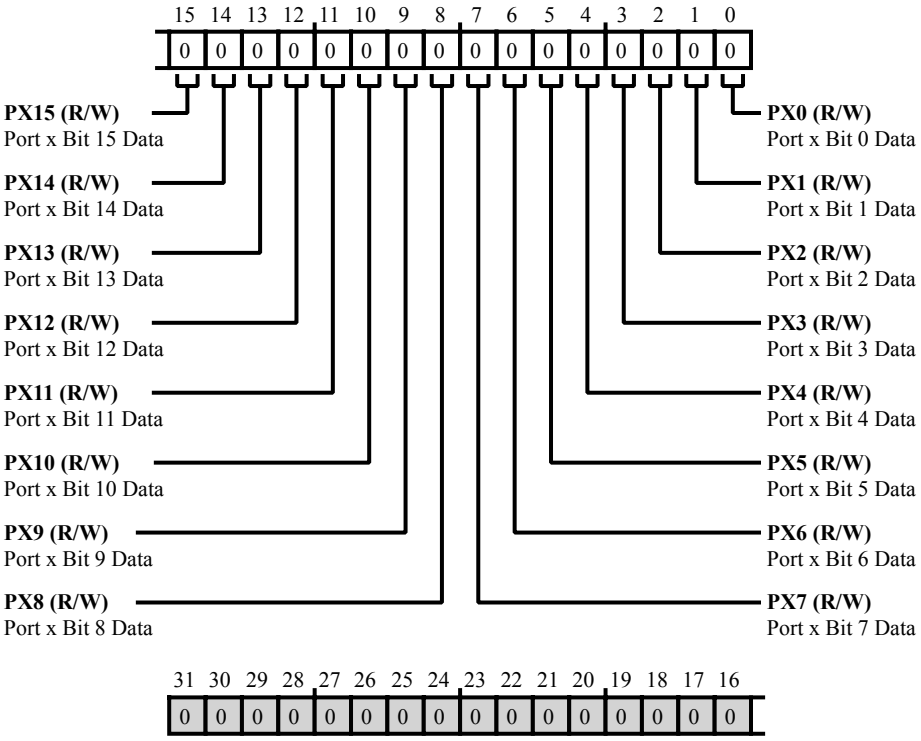

Figure 13-7: PORT_DATA Register Diagram

Table 13-8: PORT_DATA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Data.	
		0	Signal Low
		1	Signal High
14 (R/W)	PX14	Port x Bit 14 Data.	
		0	Signal Low
		1	Signal High
13 (R/W)	PX13	Port x Bit 13 Data.	
		0	Signal Low
		1	Signal High
12 (R/W)	PX12	Port x Bit 12 Data.	
		0	Signal Low
		1	Signal High
11 (R/W)	PX11	Port x Bit 11 Data.	
		0	Signal Low
		1	Signal High
10 (R/W)	PX10	Port x Bit 10 Data.	
		0	Signal Low
		1	Signal High
9 (R/W)	PX9	Port x Bit 9 Data.	
		0	Signal Low
		1	Signal High
8 (R/W)	PX8	Port x Bit 8 Data.	
		0	Signal Low
		1	Signal High
7 (R/W)	PX7	Port x Bit 7 Data.	
		0	Signal Low
		1	Signal High
6 (R/W)	PX6	Port x Bit 6 Data.	
		0	Signal Low
		1	Signal High

Table 13-8: PORT_DATA Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	PX5	Port x Bit 5 Data.	
		0	Signal Low
		1	Signal High
4 (R/W)	PX4	Port x Bit 4 Data.	
		0	Signal Low
		1	Signal High
3 (R/W)	PX3	Port x Bit 3 Data.	
		0	Signal Low
		1	Signal High
2 (R/W)	PX2	Port x Bit 2 Data.	
		0	Signal Low
		1	Signal High
1 (R/W)	PX1	Port x Bit 1 Data.	
		0	Signal Low
		1	Signal High
0 (R/W)	PX0	Port x Bit 0 Data.	
		0	Signal Low
		1	Signal High

Port x GPIO Data Clear Register

The PORT_DATA_CLR register operates differently for port bits/pins, depending on whether the bit/pin is output mode or input mode. For more information, see the PORT_DATA register description.

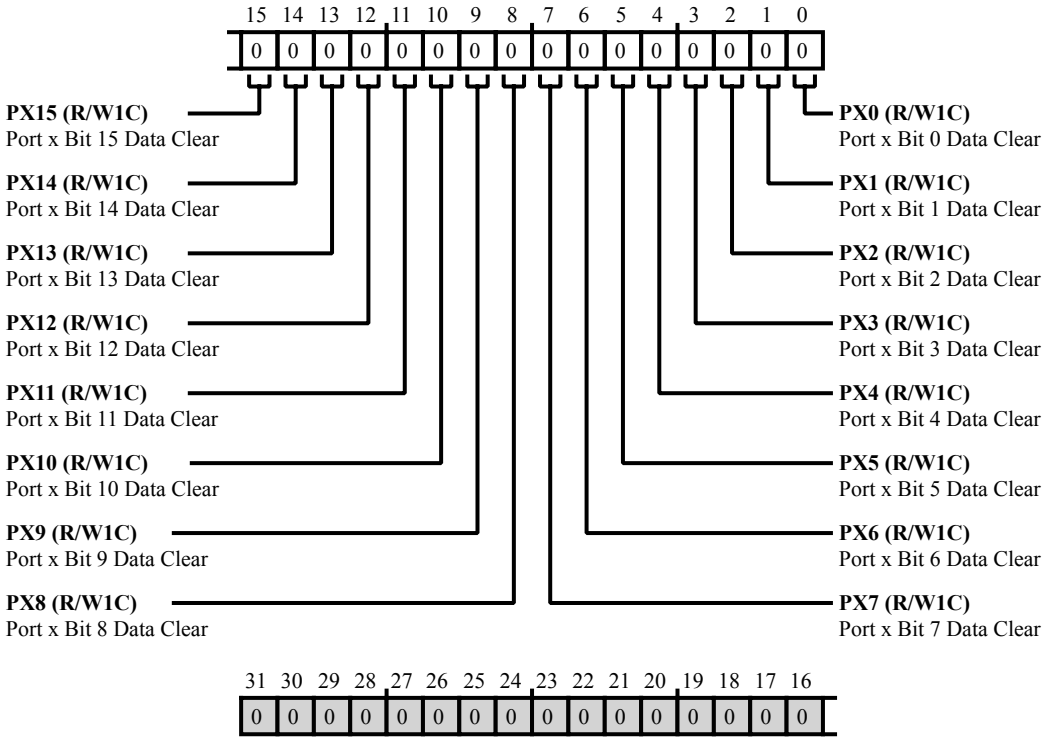

Figure 13-8: PORT_DATA_CLR Register Diagram

Table 13-9: PORT_DATA_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	PX15	Port x Bit 15 Data Clear.	
		0	No Effect
		1	Clear Bit Write 1 for signal low in output mode.
14 (R/W1C)	PX14	Port x Bit 14 Data Clear.	
		0	No Effect Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.
13 (R/W1C)	PX13	Port x Bit 13 Data Clear.	
		0	No Effect Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.

Table 13-9: PORT_DATA_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1C)	PX12	Port x Bit 12 Data Clear.	
		0	No Effect Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.
11 (R/W1C)	PX11	Port x Bit 11 Data Clear.	
		0	No Effect
		1	Clear Bit Write 1 for signal low in output mode.
10 (R/W1C)	PX10	Port x Bit 10 Data Clear.	
		0	No Effect Write 0 has no effect in output mode.Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.
9 (R/W1C)	PX9	Port x Bit 9 Data Clear.	
		0	No Effect Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.
8 (R/W1C)	PX8	Port x Bit 8 Data Clear.	
		0	No Effect Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.
7 (R/W1C)	PX7	Port x Bit 7 Data Clear.	
		0	No Effect Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.
6 (R/W1C)	PX6	Port x Bit 6 Data Clear.	
		0	No Effect Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.
5 (R/W1C)	PX5	Port x Bit 5 Data Clear.	
		0	No Effect Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.
4 (R/W1C)	PX4	Port x Bit 4 Data Clear.	
		0	No Effect Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.
3 (R/W1C)	PX3	Port x Bit 3 Data Clear.	
		0	No Effect Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.

Table 13-9: PORT_DATA_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	PX2	Port x Bit 2 Data Clear.	
		0	No Effect Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.
1 (R/W1C)	PX1	Port x Bit 1 Data Clear.	
		0	No Effect Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.
0 (R/W1C)	PX0	Port x Bit 0 Data Clear.	
		0	No Effect Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.

Port x GPIO Data Set Register

The PORT_DATA_SET register operates differently for port bits/pins, depending on whether the bit/pin is output mode or input mode. For more information, see the PORT_DATA register description.

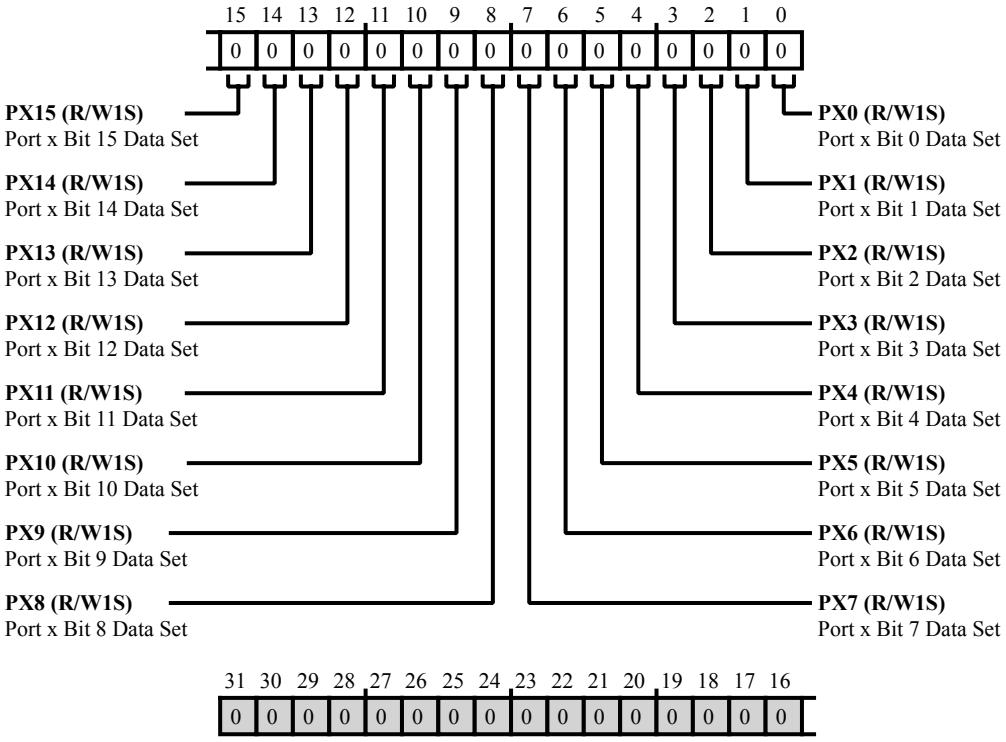

Figure 13-9: PORT_DATA_SET Register Diagram

Table 13-10: PORT_DATA_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1S)	PX15	Port x Bit 15 Data Set.	
		0	No Effect Write 0 has no effect in output mode.
		1	Set Bit Write 1 for signal high in output mode.
14 (R/W1S)	PX14	Port x Bit 14 Data Set.	
		0	No Effect Write 0 has no effect in output mode.
		1	Set Bit Write 1 for signal high in output mode.
13 (R/W1S)	PX13	Port x Bit 13 Data Set.	
		0	No Effect Write 0 has no effect in output mode.
		1	Set Bit Write 1 for signal high in output mode.

Table 13-10: PORT_DATA_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1S)	PX12	Port x Bit 12 Data Set.	
		0	No Effect Write 0 has no effect in output mode.
		1	Set Bit Write 1 for signal high in output mode.Write 1 for signal high in output mode.
11 (R/W1S)	PX11	Port x Bit 11 Data Set.	
		0	No Effect Write 0 has no effect in output mode.
		1	Set Bit
10 (R/W1S)	PX10	Port x Bit 10 Data Set.	
		0	No Effect Write 0 has no effect in output mode.
		1	Set Bit Write 1 for signal high in output mode.
9 (R/W1S)	PX9	Port x Bit 9 Data Set.	
		0	No Effect Write 0 has no effect in output mode.
		1	Set Bit Write 1 for signal high in output mode.
8 (R/W1S)	PX8	Port x Bit 8 Data Set.	
		0	No Effect Write 0 has no effect in output mode.
		1	Set Bit Write 1 for signal high in output mode.
7 (R/W1S)	PX7	Port x Bit 7 Data Set.	
		0	No Effect Write 0 has no effect in output mode.
		1	Set Bit Write 1 for signal high in output mode.
6 (R/W1S)	PX6	Port x Bit 6 Data Set.	
		0	No Effect Write 0 has no effect in output mode.
		1	Set Bit Write 1 for signal high in output mode.
5 (R/W1S)	PX5	Port x Bit 5 Data Set.	
		0	No Effect Write 0 has no effect in output mode.
		1	Set Bit Write 1 for signal high in output mode.
4 (R/W1S)	PX4	Port x Bit 4 Data Set.	
		0	No Effect Write 0 has no effect in output mode.
		1	Set Bit Write 1 for signal high in output mode.
3 (R/W1S)	PX3	Port x Bit 3 Data Set.	
		0	No Effect Write 0 has no effect in output mode.
		1	Set Bit Write 1 for signal high in output mode.

Table 13-10: PORT_DATA_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1S)	PX2	Port x Bit 2 Data Set.	
		0	No Effect Write 0 has no effect in output mode.
		1	Set Bit Write 1 for signal high in output mode.
1 (R/W1S)	PX1	Port x Bit 1 Data Set.	
		0	No Effect Write 0 has no effect in output mode.
		1	Set Bit Write 1 for signal high in output mode.
0 (R/W1S)	PX0	Port x Bit 0 Data Set.	
		0	No Effect Write 0 has no effect in output mode.
		1	Set Bit Write 1 for signal high in output mode.

Port x GPIO Output Toggle Register

The PORT_DATA_TGL register permits toggling the state of output GPIO pins. Setting bits in the PORT_DATA_TGL register affects the state of specific pins without impacting other pins of the port.

Reading the PORT_DATA_TGL returns the state of the PORT_DATA register output pin state, but does not return the input pin/signal state.

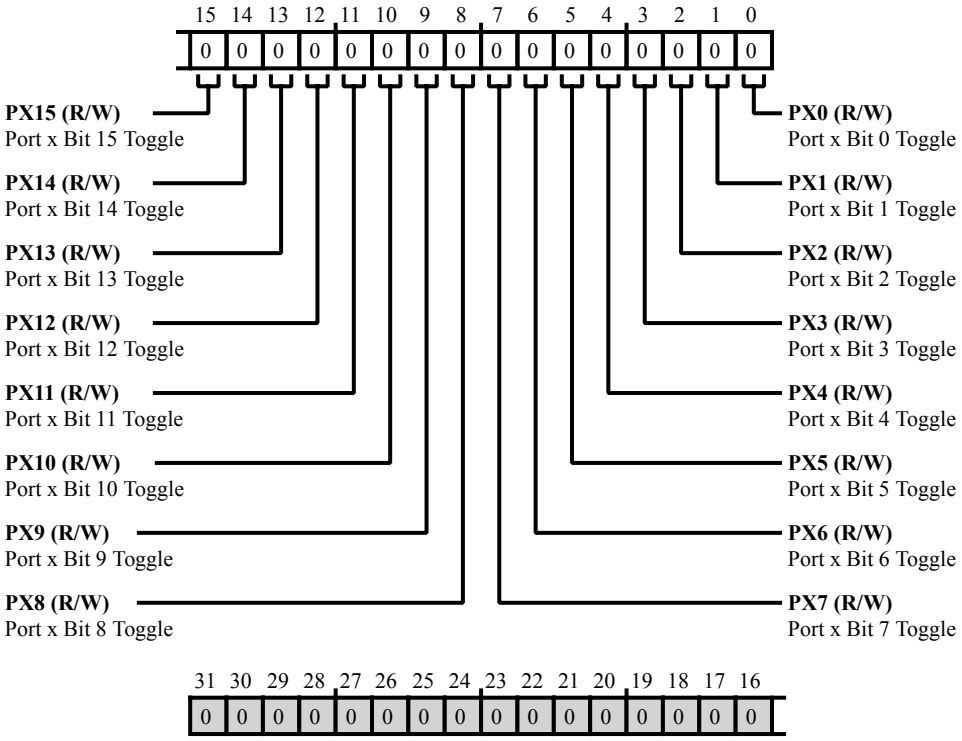

Figure 13-10: PORT_DATA_TGL Register Diagram

Table 13-11: PORT_DATA_TGL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
14 (R/W)	PX14	Port x Bit 14 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.

Table 13-11: PORT_DATA_TGL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	PX13	Port x Bit 13 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
12 (R/W)	PX12	Port x Bit 12 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
11 (R/W)	PX11	Port x Bit 11 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
10 (R/W)	PX10	Port x Bit 10 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
9 (R/W)	PX9	Port x Bit 9 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
8 (R/W)	PX8	Port x Bit 8 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
7 (R/W)	PX7	Port x Bit 7 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
6 (R/W)	PX6	Port x Bit 6 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
5 (R/W)	PX5	Port x Bit 5 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
4 (R/W)	PX4	Port x Bit 4 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.

Table 13-11: PORT_DATA_TGL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	PX3	Port x Bit 3 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
2 (R/W)	PX2	Port x Bit 2 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
1 (R/W)	PX1	Port x Bit 1 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
0 (R/W)	PX0	Port x Bit 0 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.

Port x GPIO Direction Register

The PORT_DIR, PORT_DIR_SET, and PORT_DIR_CLR registers select output or input mode for GPIO pins and enable output drivers. Use the PORT_INEN, PORT_INEN_SET, and PORT_INEN_CLR registers to enable or disable input drivers.

Writes to the PORT_DIR register affect the state of all pins of the port. To select direction for specific pins without impacting other pins of the port, use the PORT_DIR_SET and PORT_DIR_CLR registers.

Setting a bit in the PORT_DIR register enables output mode on the corresponding a GPIO pin, and a clearing a bit in the PORT_DIR register disables output mode on the corresponding GPIO pin.

Input Mode - The default mode of every GPIO pin after reset is input mode, but the input drivers are not enabled. To enable any GPIO input drivers, set the corresponding bits in PORT_INEN register. When enabled, a read from the PORT_DATA register returns the logical state of the input pin. The input signal does not overwrite the state of the bit used for the output case. That state can only be altered by software. If the input driver is enabled, a write to the PORT_DATA register can alter the state of the bit, but the change cannot be read back.

Output Mode - Any GPIO pin can be configured for output mode. The GPIO output drivers are enabled by setting the corresponding bits in the PORT_DIR, PORT_DIR_SET, or PORT_DIR_CLR registers. By using the PORT_DIR_SET and PORT_DIR_CLR registers, direction of the signal flow of individual GPIO pins can be altered by separate software threads without mutually impacting other GPIOs on the same port. Both registers return the same value when read. Because the state of the GPIO output can already be controlled before the output driver is enabled, it is recommended to first set or clear the bit (using the PORT_DATA, PORT_DATA_SET, or PORT_DATA_CLR registers) to avoid any volatile levels on the output.

Open-Drain Mode- Every GPIO can also be used in open-drain mode. To accomplish this, first, clear the respective bit in the PORT_DATA or PORT_DATA_CLR register then set the one bit in the PORT_INEN register. Reads from the PORT_DATA register then return the status from the pin and do not return the state of the internal flip-flop. By toggling the output driver through the PORT_DIR_SET and PORT_DIR_CLR register pair, the output signal can be pulled low or three-stated as required. Note that the polarity of the driven signal can be inverted when the internal flip-flop is set instead. When a GPIO port is used in open-drain mode, care must be taken not to exceed the V_{IH} operating condition associated with the respective pin.

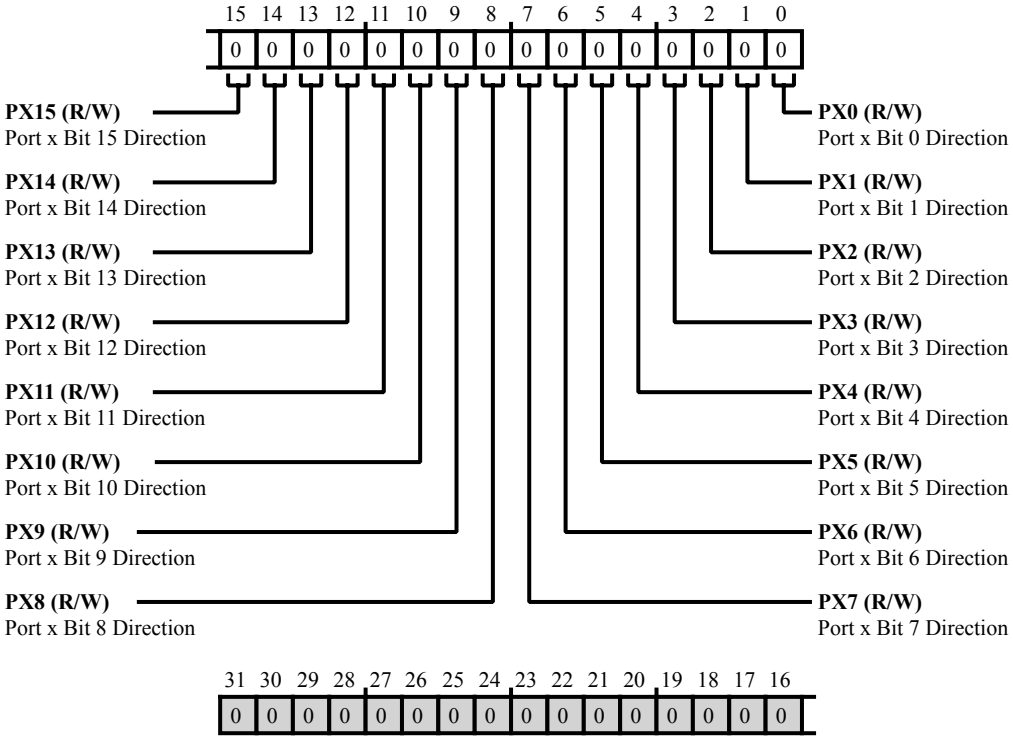

Figure 13-11: PORT_DIR Register Diagram

Table 13-12: PORT_DIR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
14 (R/W)	PX14	Port x Bit 14 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
13 (R/W)	PX13	Port x Bit 13 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
12 (R/W)	PX12	Port x Bit 12 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.

Table 13-12: PORT_DIR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	PX11	Port x Bit 11 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
10 (R/W)	PX10	Port x Bit 10 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
9 (R/W)	PX9	Port x Bit 9 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
8 (R/W)	PX8	Port x Bit 8 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
7 (R/W)	PX7	Port x Bit 7 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
6 (R/W)	PX6	Port x Bit 6 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
5 (R/W)	PX5	Port x Bit 5 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
4 (R/W)	PX4	Port x Bit 4 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
3 (R/W)	PX3	Port x Bit 3 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
2 (R/W)	PX2	Port x Bit 2 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.

Table 13-12: PORT_DIR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	PX1	Port x Bit 1 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
0 (R/W)	PX0	Port x Bit 0 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.

Port x GPIO Direction Clear Register

The PORT_DIR_CLR register disables output mode and disables output drivers for GPIO pins. For more information, see the PORT_DIR register description.

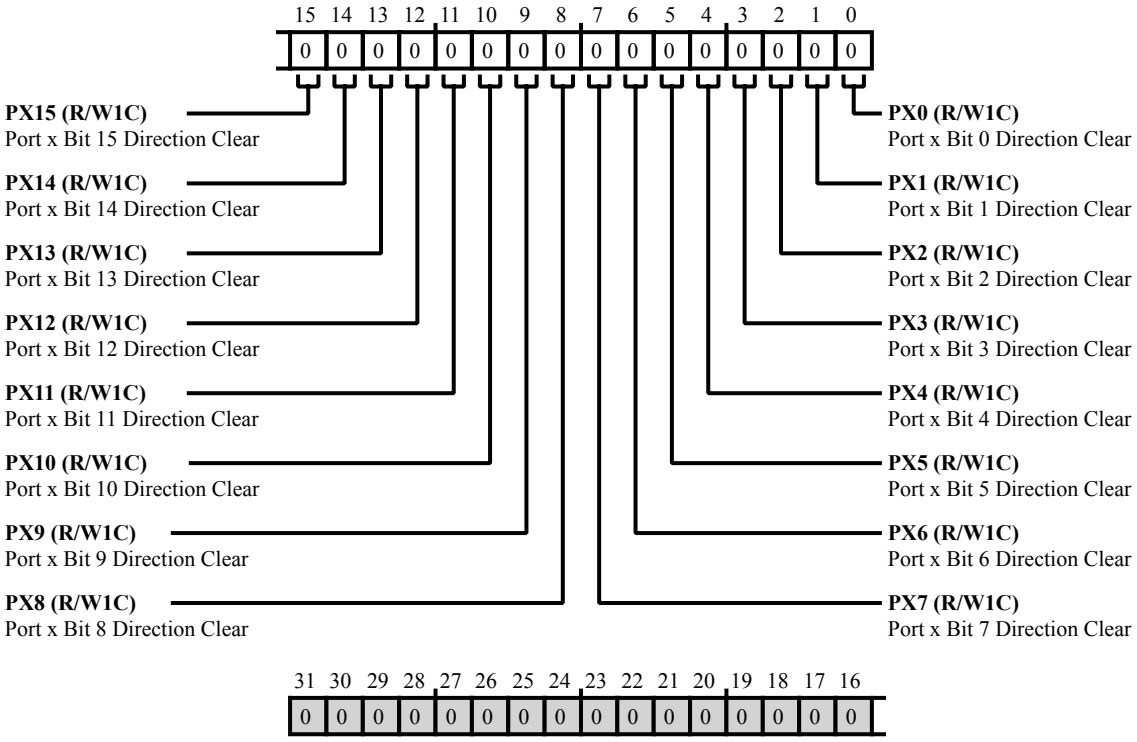

Figure 13-12: PORT_DIR_CLR Register Diagram

Table 13-13: PORT_DIR_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	PX15	Port x Bit 15 Direction Clear. The PORT_DIR_CLR.PX15 bit disables output mode/driver for port x.	
		0	No Effect
		1	Disable output mode/driver
14 (R/W1C)	PX14	Port x Bit 14 Direction Clear. The PORT_DIR_CLR.PX14 bit disables output mode/driver for port x.	
		0	No Effect
		1	Disable output mode/driver

Table 13-13: PORT_DIR_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1C)	PX13	Port x Bit 13 Direction Clear. The PORT_DIR_CLR.PX13 bit disables output mode/driver for port x.	
		0	No Effect
		1	Disable output mode/driver
12 (R/W1C)	PX12	Port x Bit 12 Direction Clear. The PORT_DIR_CLR.PX12 bit disables output mode/driver for port x.	
		0	No Effect
		1	Disable output mode/driver
11 (R/W1C)	PX11	Port x Bit 11 Direction Clear. The PORT_DIR_CLR.PX11 bit disables output mode/driver for port x.	
		0	No Effect
		1	Disable output mode/driver
10 (R/W1C)	PX10	Port x Bit 10 Direction Clear. The PORT_DIR_CLR.PX10 bit disables output mode/driver for port x.	
		0	No Effect
		1	Disable output mode/driver
9 (R/W1C)	PX9	Port x Bit 9 Direction Clear. The PORT_DIR_CLR.PX9 bit disables output mode/driver for port x.	
		0	No Effect
		1	Disable output mode/driver
8 (R/W1C)	PX8	Port x Bit 8 Direction Clear. The PORT_DIR_CLR.PX8 bit disables output mode/driver for port x.	
		0	No Effect
		1	Disable output mode/driver
7 (R/W1C)	PX7	Port x Bit 7 Direction Clear. The PORT_DIR_CLR.PX7 bit disables output mode/driver for port x.	
		0	No Effect
		1	Disable output mode/driver

Table 13-13: PORT_DIR_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1C)	PX6	Port x Bit 6 Direction Clear. The PORT_DIR_CLR.PX6 bit disables output mode/driver for port x.	
		0	No Effect
		1	Disable output mode/driver
5 (R/W1C)	PX5	Port x Bit 5 Direction Clear. The PORT_DIR_CLR.PX5 bit disables output mode/driver for port x.	
		0	No Effect
		1	Disable output mode/driver
4 (R/W1C)	PX4	Port x Bit 4 Direction Clear.	
		0	No Effect
		1	Disable output mode/driver
3 (R/W1C)	PX3	Port x Bit 3 Direction Clear. The PORT_DIR_CLR.PX3 bit disables output mode/driver for port x.	
		0	No Effect
		1	Disable output mode/driver
2 (R/W1C)	PX2	Port x Bit 2 Direction Clear. The PORT_DIR_CLR.PX2 bit disables output mode/driver for port x.	
		0	No Effect
		1	Disable output mode/driver
1 (R/W1C)	PX1	Port x Bit 1 Direction Clear. The PORT_DIR_CLR.PX1 bit disables output mode/driver for port x.	
		0	No Effect
		1	Disable output mode/driver
0 (R/W1C)	PX0	Port x Bit 0 Direction Clear. The PORT_DIR_CLR.PX0 bit disables output mode/driver for port x.	
		0	No Effect
		1	Disable output mode/driver

Port x GPIO Direction Set Register

The PORT_DIR_SET register enable output mode and enables output drivers for GPIO pins. For more information, see the PORT_DIR register description.

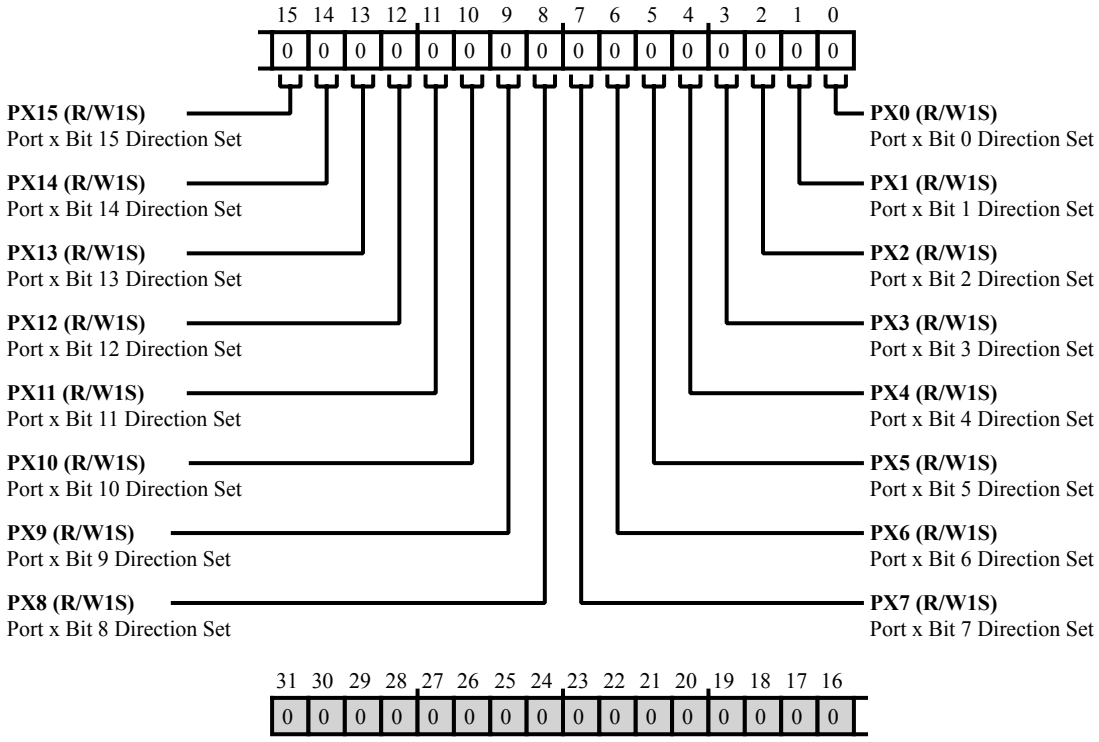

Figure 13-13: PORT_DIR_SET Register Diagram

Table 13-14: PORT_DIR_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1S)	PX15	Port x Bit 15 Direction Set. The PORT_DIR_SET.PX15 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
14 (R/W1S)	PX14	Port x Bit 14 Direction Set. The PORT_DIR_SET.PX14 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver

Table 13-14: PORT_DIR_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1S)	PX13	Port x Bit 13 Direction Set. The PORT_DIR_SET.PX13 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
12 (R/W1S)	PX12	Port x Bit 12 Direction Set. The PORT_DIR_SET.PX12 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
11 (R/W1S)	PX11	Port x Bit 11 Direction Set. The PORT_DIR_SET.PX11 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
10 (R/W1S)	PX10	Port x Bit 10 Direction Set. The PORT_DIR_SET.PX10 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
9 (R/W1S)	PX9	Port x Bit 9 Direction Set. The PORT_DIR_SET.PX9 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
8 (R/W1S)	PX8	Port x Bit 8 Direction Set. The PORT_DIR_SET.PX8 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
7 (R/W1S)	PX7	Port x Bit 7 Direction Set. The PORT_DIR_SET.PX7 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver

Table 13-14: PORT_DIR_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1S)	PX6	Port x Bit 6 Direction Set. The PORT_DIR_SET.PX6 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
5 (R/W1S)	PX5	Port x Bit 5 Direction Set. The PORT_DIR_SET.PX5 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
4 (R/W1S)	PX4	Port x Bit 4 Direction Set. The PORT_DIR_SET.PX4 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
3 (R/W1S)	PX3	Port x Bit 3 Direction Set. The PORT_DIR_SET.PX3 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
2 (R/W1S)	PX2	Port x Bit 2 Direction Set. The PORT_DIR_SET.PX2 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
1 (R/W1S)	PX1	Port x Bit 1 Direction Set. The PORT_DIR_SET.PX1 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
0 (R/W1S)	PX0	Port x Bit 0 Direction Set. The PORT_DIR_SET.PX0 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver

Port x Function Enable Register

The PORT_FER register bits indicate each port bit's operating mode: general purpose I/O mode or peripheral mode. After reset, all pins default to GPIO mode. Setting a bit in the PORT_FER registers enables a peripheral module to take ownership of the pin. The function enable bits impact output control only. Regardless of the setting of the function enable bits, both GPIO and peripherals can still sense the pin input. After a function is enabled, it is up to the PORT_MUX registers as to which peripheral takes control.

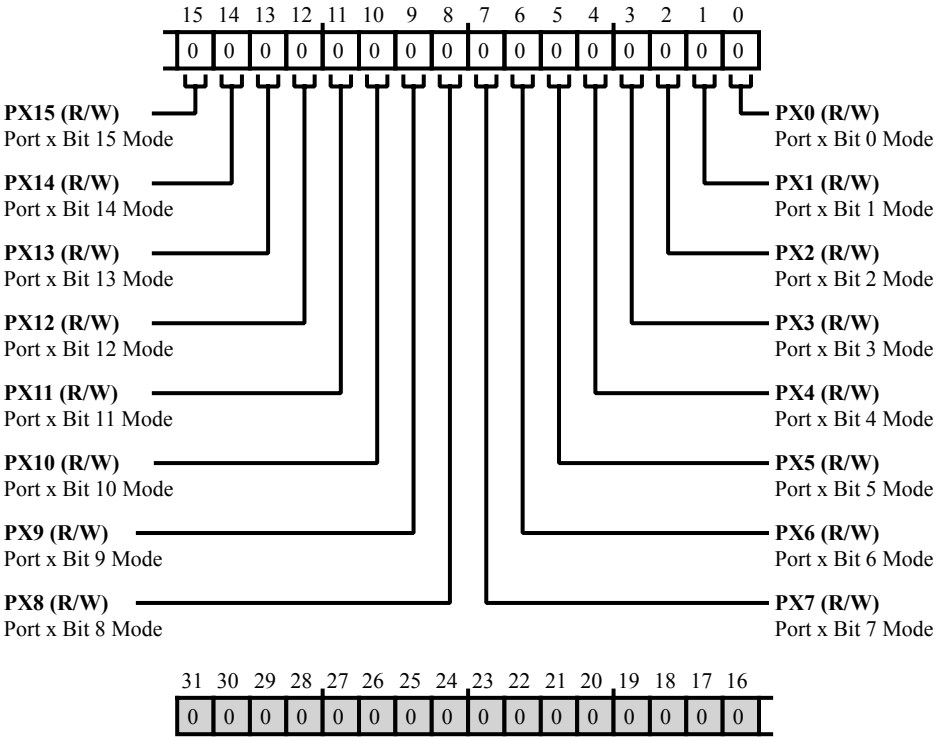

Figure 13-14: PORT_FER Register Diagram

Table 13-15: PORT_FER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
14 (R/W)	PX14	Port x Bit 14 Mode.	
		0	GPIO Mode
		1	Peripheral Mode

Table 13-15: PORT_FER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	PX13	Port x Bit 13 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
12 (R/W)	PX12	Port x Bit 12 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
11 (R/W)	PX11	Port x Bit 11 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
10 (R/W)	PX10	Port x Bit 10 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
9 (R/W)	PX9	Port x Bit 9 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
8 (R/W)	PX8	Port x Bit 8 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
7 (R/W)	PX7	Port x Bit 7 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
6 (R/W)	PX6	Port x Bit 6 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
5 (R/W)	PX5	Port x Bit 5 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
4 (R/W)	PX4	Port x Bit 4 Mode.	
		0	GPIO Mode
		1	Peripheral Mode

Table 13-15: PORT_FER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	PX3	Port x Bit 3 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
2 (R/W)	PX2	Port x Bit 2 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
1 (R/W)	PX1	Port x Bit 1 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
0 (R/W)	PX0	Port x Bit 0 Mode.	
		0	GPIO Mode
		1	Peripheral Mode

Port x Function Enable Clear Register

The PORT_FER_CLR register permits enabling GPIO mode for each bit and corresponding GPIO pin. Writing 1 to a bit in PORT_FER_CLR enables GPIO mode for the corresponding pin.

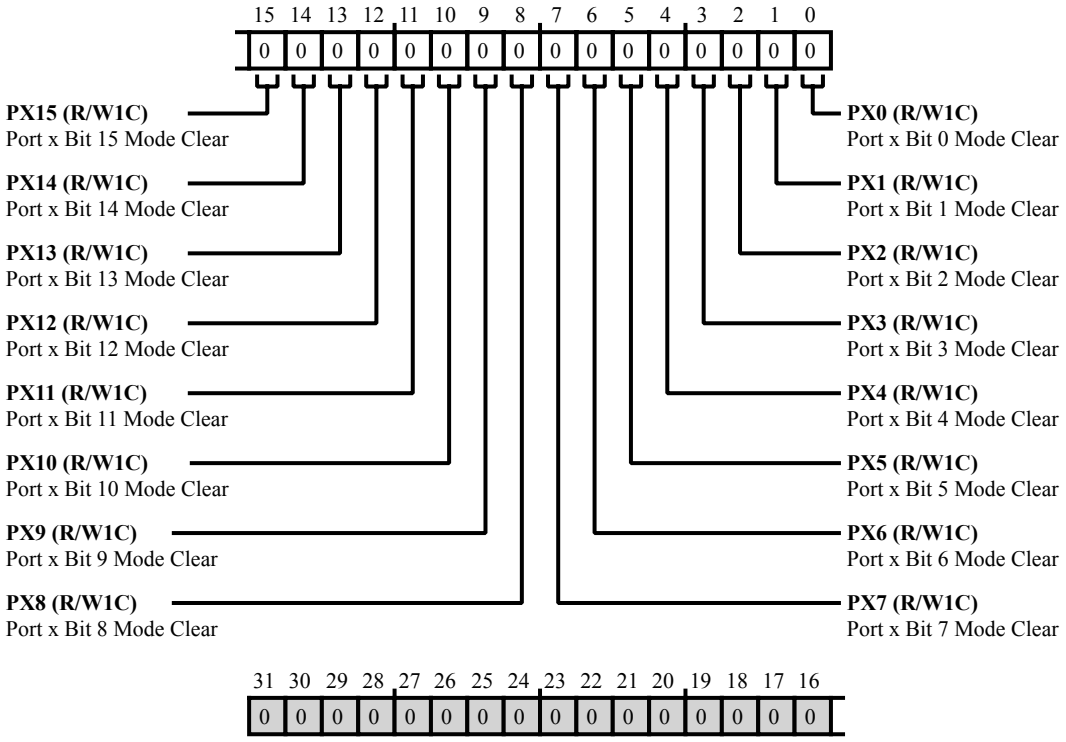

Figure 13-15: PORT_FER_CLR Register Diagram

Table 13-16: PORT_FER_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	PX15	Port x Bit 15 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
14 (R/W1C)	PX14	Port x Bit 14 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
13 (R/W1C)	PX13	Port x Bit 13 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode

Table 13-16: PORT_FER_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1C)	PX12	Port x Bit 12 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
11 (R/W1C)	PX11	Port x Bit 11 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
10 (R/W1C)	PX10	Port x Bit 10 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
9 (R/W1C)	PX9	Port x Bit 9 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
8 (R/W1C)	PX8	Port x Bit 8 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
7 (R/W1C)	PX7	Port x Bit 7 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
6 (R/W1C)	PX6	Port x Bit 6 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
5 (R/W1C)	PX5	Port x Bit 5 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
4 (R/W1C)	PX4	Port x Bit 4 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
3 (R/W1C)	PX3	Port x Bit 3 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode

Table 13-16: PORT_FER_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	PX2	Port x Bit 2 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
1 (R/W1C)	PX1	Port x Bit 1 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
0 (R/W1C)	PX0	Port x Bit 0 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode

Port x Function Enable Set Register

The PORT_FER_SET register permits enabling peripheral mode for each bit and corresponding GPIO pin. Writing 1 to a bit in PORT_FER_SET enables peripheral mode for the corresponding pin.

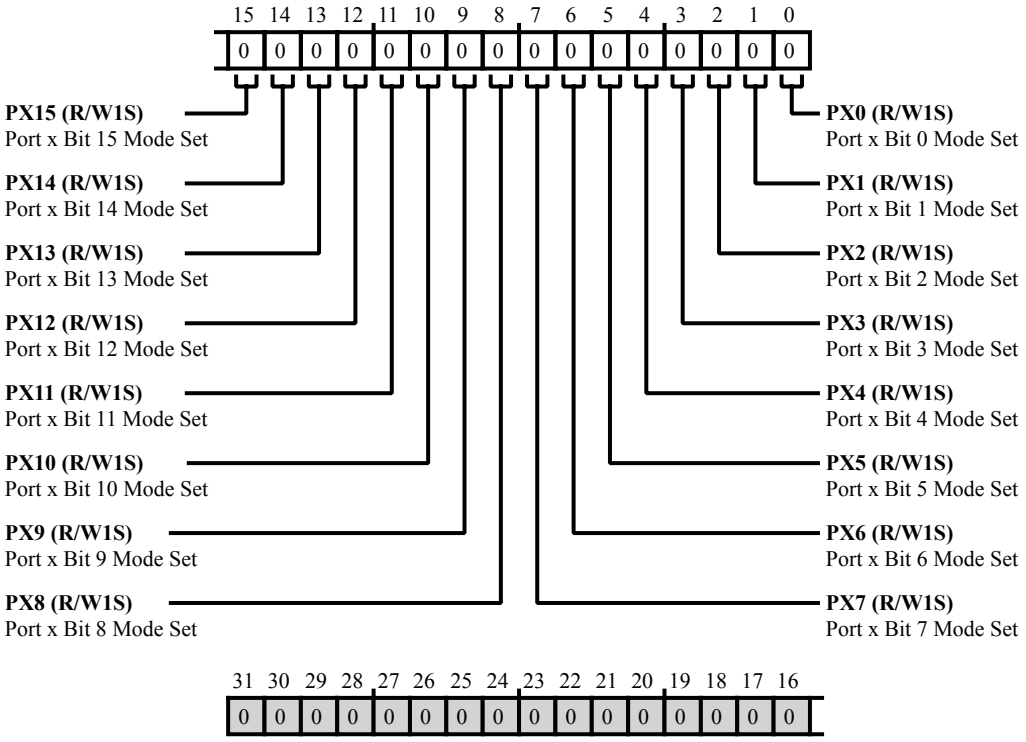

Figure 13-16: PORT_FER_SET Register Diagram

Table 13-17: PORT_FER_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1S)	PX15	Port x Bit 15 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
14 (R/W1S)	PX14	Port x Bit 14 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
13 (R/W1S)	PX13	Port x Bit 13 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode

Table 13-17: PORT_FER_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1S)	PX12	Port x Bit 12 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
11 (R/W1S)	PX11	Port x Bit 11 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
10 (R/W1S)	PX10	Port x Bit 10 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
9 (R/W1S)	PX9	Port x Bit 9 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
8 (R/W1S)	PX8	Port x Bit 8 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
7 (R/W1S)	PX7	Port x Bit 7 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
6 (R/W1S)	PX6	Port x Bit 6 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
5 (R/W1S)	PX5	Port x Bit 5 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
4 (R/W1S)	PX4	Port x Bit 4 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
3 (R/W1S)	PX3	Port x Bit 3 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode

Table 13-17: PORT_FER_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1S)	PX2	Port x Bit 2 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
1 (R/W1S)	PX1	Port x Bit 1 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
0 (R/W1S)	PX0	Port x Bit 0 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode

Port x GPIO Input Enable Register

The PORT_INEN, PORT_INEN_SET, and PORT_INEN_CLR registers enable or disable input drivers, which are required for using a GPIO pin in input mode.

Writes to the PORT_INEN register affect the input drivers for all pins of the port. To set or clear specific pin drivers without impacting other pin drivers of the port, use the PORT_INEN_SET and PORT_INEN_CLR registers.

If the input is enabled, reads from the PORT_DATA, PORT_DATA_SET, or PORT_DATA_CLR registers return the state of the pins. However, the state of the output is not overwritten by the input. It is altered by software writes only. Input and output drivers can be enabled at the same time. In this case, a read of the data register returns the true value of the data register and not the pin state.

For more information see the PORT_DATA register description and the PORT_DIR register description.

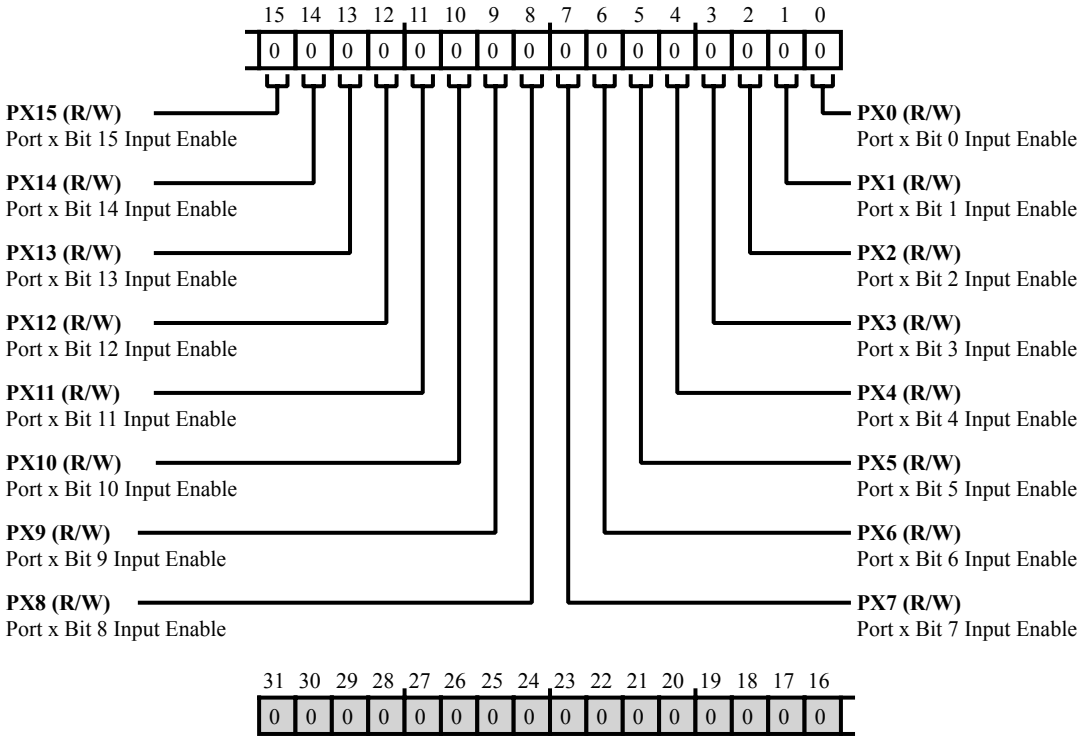

Figure 13-17: PORT_INEN Register Diagram

Table 13-18: PORT_INEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver

Table 13-18: PORT_INEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	PX14	Port x Bit 14 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
13 (R/W)	PX13	Port x Bit 13 Input Enable.	
		0	Input disabled
		1	Enable Input Driver
12 (R/W)	PX12	Port x Bit 12 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
11 (R/W)	PX11	Port x Bit 11 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
10 (R/W)	PX10	Port x Bit 10 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
9 (R/W)	PX9	Port x Bit 9 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
8 (R/W)	PX8	Port x Bit 8 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
7 (R/W)	PX7	Port x Bit 7 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
6 (R/W)	PX6	Port x Bit 6 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
5 (R/W)	PX5	Port x Bit 5 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver

Table 13-18: PORT_INEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	PX4	Port x Bit 4 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
3 (R/W)	PX3	Port x Bit 3 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
2 (R/W)	PX2	Port x Bit 2 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
1 (R/W)	PX1	Port x Bit 1 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
0 (R/W)	PX0	Port x Bit 0 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver

Port x GPIO Input Enable Clear Register

The PORT_INEN_CLR register disables input drivers for GPIO pins. For more information, see the PORT_INEN register description.

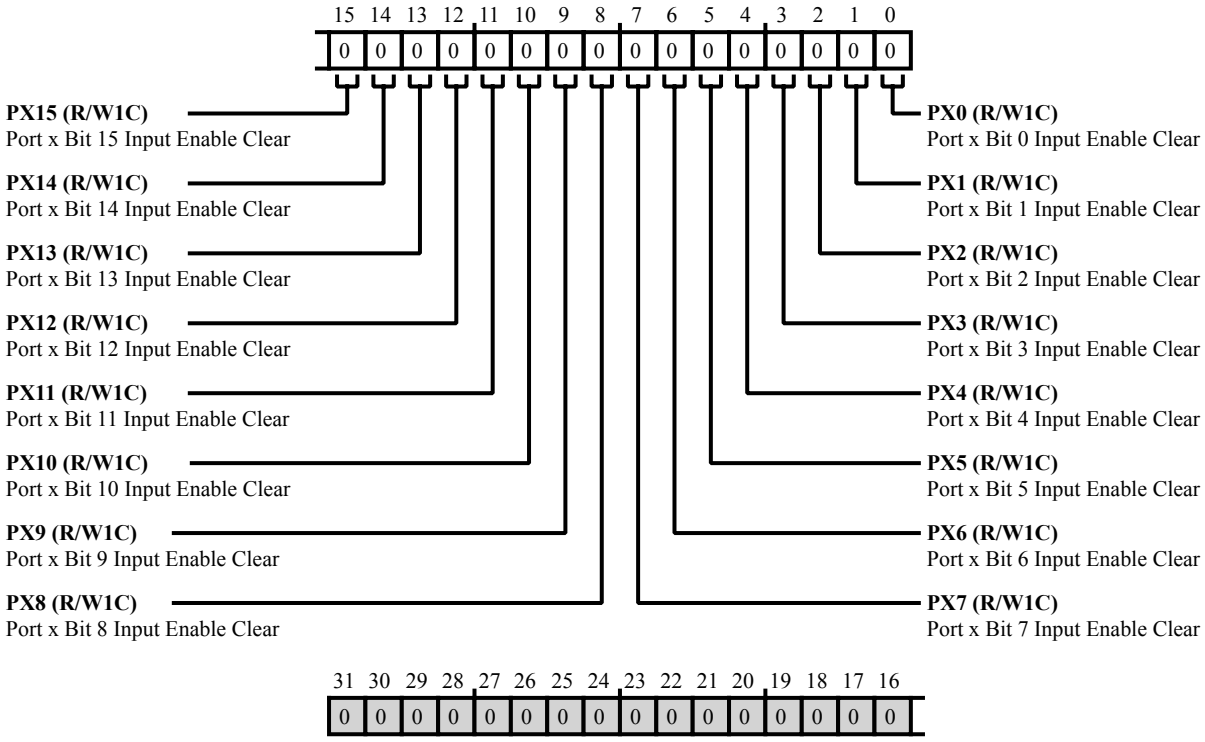

Figure 13-18: PORT_INEN_CLR Register Diagram

Table 13-19: PORT_INEN_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	PX15	Port x Bit 15 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
14 (R/W1C)	PX14	Port x Bit 14 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
13 (R/W1C)	PX13	Port x Bit 13 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.

Table 13-19: PORT_INEN_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1C)	PX12	Port x Bit 12 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
11 (R/W1C)	PX11	Port x Bit 11 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
10 (R/W1C)	PX10	Port x Bit 10 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
9 (R/W1C)	PX9	Port x Bit 9 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
8 (R/W1C)	PX8	Port x Bit 8 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
7 (R/W1C)	PX7	Port x Bit 7 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
6 (R/W1C)	PX6	Port x Bit 6 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
5 (R/W1C)	PX5	Port x Bit 5 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
4 (R/W1C)	PX4	Port x Bit 4 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
3 (R/W1C)	PX3	Port x Bit 3 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.

Table 13-19: PORT_INEN_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	PX2	Port x Bit 2 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
1 (R/W1C)	PX1	Port x Bit 1 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
0 (R/W1C)	PX0	Port x Bit 0 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.

Port x GPIO Input Enable Set Register

The PORT_INEN_SET register enables input drivers for GPIO pins. For more information, see the PORT_INEN register description.

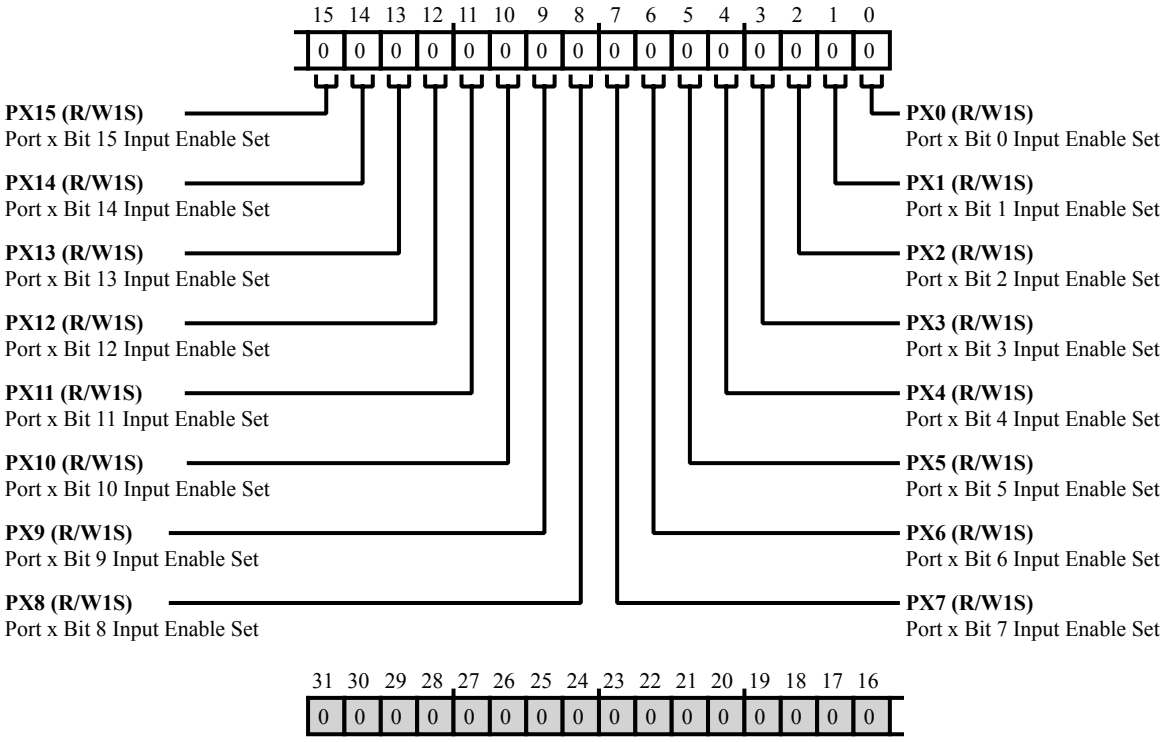

Figure 13-19: PORT_INEN_SET Register Diagram

Table 13-20: PORT_INEN_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1S)	PX15	Port x Bit 15 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
14 (R/W1S)	PX14	Port x Bit 14 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
13 (R/W1S)	PX13	Port x Bit 13 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.

Table 13-20: PORT_INEN_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1S)	PX12	Port x Bit 12 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
11 (R/W1S)	PX11	Port x Bit 11 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
10 (R/W1S)	PX10	Port x Bit 10 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
9 (R/W1S)	PX9	Port x Bit 9 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
8 (R/W1S)	PX8	Port x Bit 8 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
7 (R/W1S)	PX7	Port x Bit 7 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
6 (R/W1S)	PX6	Port x Bit 6 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
5 (R/W1S)	PX5	Port x Bit 5 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
4 (R/W1S)	PX4	Port x Bit 4 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
3 (R/W1S)	PX3	Port x Bit 3 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.

Table 13-20: PORT_INEN_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1S)	PX2	Port x Bit 2 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
1 (R/W1S)	PX1	Port x Bit 1 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
0 (R/W1S)	PX0	Port x Bit 0 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.

Port x GPIO Lock Register

The PORT_LOCK register enables (unlocks) or disables (locks) write access selectively for the PORT control registers.

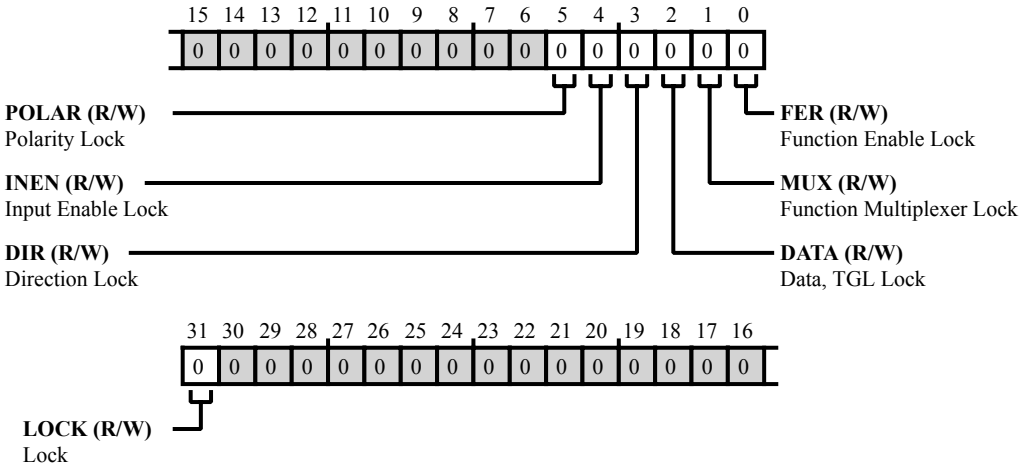

Figure 13-20: PORT_LOCK Register Diagram

Table 13-21: PORT_LOCK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the PORT_LOCK.LOCK bit is set, the PORT_LOCK register is read only (locked).	
		0	Unlock
		1	Lock
5 (R/W)	POLAR	Polarity Lock. The PORT_LOCK.POLAR disables write access to the PORT_POL, PORT_POL_SET, and PORT_POL_CLR registers.	
		0	Unlock POL
		1	Lock POL
4 (R/W)	INEN	Input Enable Lock. The PORT_LOCK.INEN disables write access to the PORT_INEN, PORT_INEN_SET, and PORT_INEN_CLR registers.	
		0	Unlock INEN
		1	Lock INEN

Table 13-21: PORT_LOCK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	DIR	Direction Lock. The PORT_LOCK.DIR disables write access to the PORT_DIR, PORT_DIR_SET, PORT_DIR_CLR registers.	
		0	Lock DIR
		1	Unlock DIR
2 (R/W)	DATA	Data, TGL Lock. The PORT_LOCK.DATA disables write access to the PORT_DATA, PORT_DATA_SET, PORT_DATA_CLR, and PORT_DATA_TGL registers.	
		0	Unlock DATA
		1	Lock DATA
1 (R/W)	MUX	Function Multiplexer Lock. The PORT_LOCK.MUX disables write accesses to the PORT_MUX register.	
		0	Unlock MUX
		1	Lock MUX
0 (R/W)	FER	Function Enable Lock. The PORT_LOCK.FER disables write access to the PORT_FER, PORT_FER_SET, and PORT_FER_CLR registers.	
		0	Unlock FER
		1	Lock FER

Port x Multiplexer Control Register

When a pin is in peripheral mode (not GPIO mode), the PORT_MUX register controls which peripheral takes ownership of a pin. Ports may have multiple, different peripheral functions. Two bits are required to describe every multiplexer on an individual pin-by-pin scheme. For example, Bit 0 and Bit 1 of the PORT_MUX register control the multiplexer of Pin 0, Bit 2 and Bit 3 of PORT_MUX control the multiplexer of Pin 1, and so on. The value of any PORT_MUX bit has no effect on the port pins when the associated bit in the PORT_FER register is 0 (selects GPIO mode). Even if a port has only one function, the PORT_MUX register is still present. For single function ports (no multiplexing is needed), leave the PORT_MUX bits at 0 (default). For all PORT_MUX bit fields: 00 = default/reset peripheral option, 01 = first alternate peripheral option, 10 = second alternate peripheral option, and 11 = third alternate peripheral option.

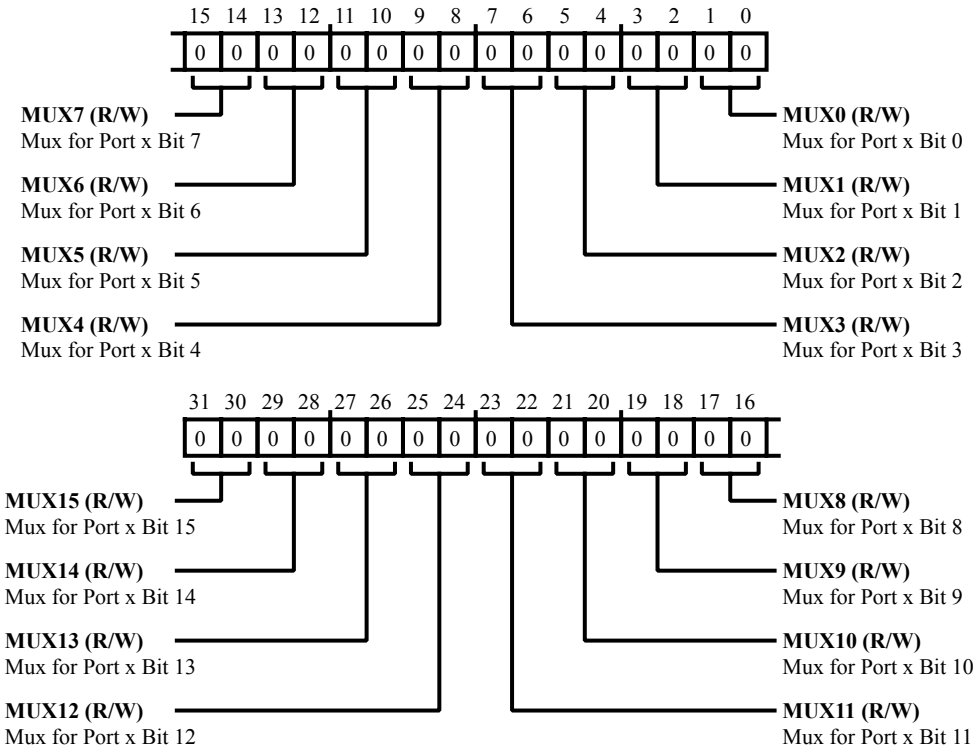

Figure 13-21: PORT_MUX Register Diagram

Table 13-22: PORT_MUX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:30 (R/W)	MUX15	Mux for Port x Bit 15. Multiplexer control for Port x bit 15.
29:28 (R/W)	MUX14	Mux for Port x Bit 14. Multiplexer control for Port x bit 14.

Table 13-22: PORT_MUX Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
27:26 (R/W)	MUX13	Mux for Port x Bit 13. Multiplexer control for Port x bit 13.
25:24 (R/W)	MUX12	Mux for Port x Bit 12. Multiplexer control for Port x bit 12.
23:22 (R/W)	MUX11	Mux for Port x Bit 11. Multiplexer control for Port x bit 11.
21:20 (R/W)	MUX10	Mux for Port x Bit 10. Multiplexer control for Port x bit 10.
19:18 (R/W)	MUX9	Mux for Port x Bit 9. Multiplexer control for Port x bit 9.
17:16 (R/W)	MUX8	Mux for Port x Bit 8. Multiplexer control for Port x bit 8.
15:14 (R/W)	MUX7	Mux for Port x Bit 7. Multiplexer control for Port x bit 7.
13:12 (R/W)	MUX6	Mux for Port x Bit 6. Multiplexer control for Port x bit 6.
11:10 (R/W)	MUX5	Mux for Port x Bit 5. Multiplexer control for Port x bit 5.
9:8 (R/W)	MUX4	Mux for Port x Bit 4. Multiplexer control for Port x bit 4.
7:6 (R/W)	MUX3	Mux for Port x Bit 3. Multiplexer control for Port x bit 3.
5:4 (R/W)	MUX2	Mux for Port x Bit 2. Multiplexer control for Port x bit 2.
3:2 (R/W)	MUX1	Mux for Port x Bit 1. Multiplexer control for Port x bit 1.
1:0 (R/W)	MUX0	Mux for Port x Bit 0. Multiplexer control for Port x bit 0.

Port x GPIO Polarity Invert Register

The PORT_POL, PORT_POL_SET, and PORT_POL_CLR registers enable or disable inverting polarity of GPIO signals. To invert polarity of peripheral signals, use the inversion selection programming in the signal's corresponding module.

Writes to the PORT_POL register affect the polarity inversion selection of all pins of the port. To enable or disable polarity inversion for specific pins without impacting other pins of the port, use the PORT_POL_SET and PORT_POL_CLR registers.

Setting a bit in the PORT_POL register enables polarity inversion on the corresponding inversion GPIO pin, making the pin active-low or falling-edge sensitive. Clearing a bit in the PORT_POL register disables polarity (default state) on the corresponding GPIO pin, making it active-high or rising-edge sensitive.

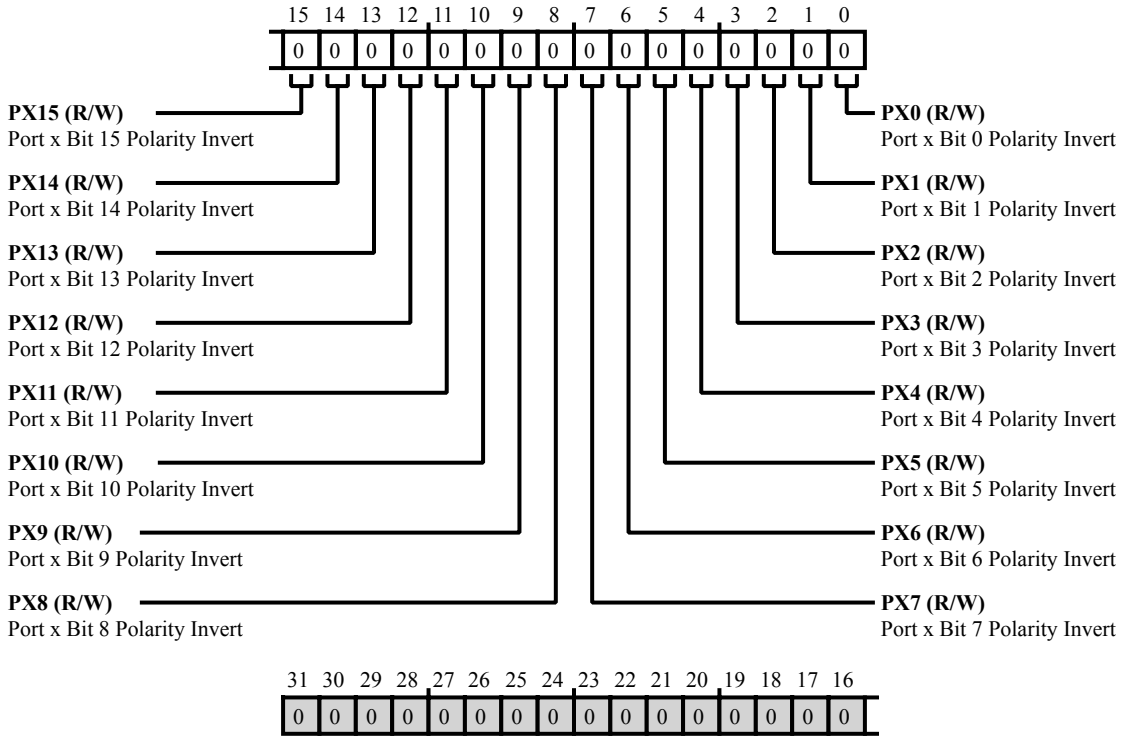

Figure 13-22: PORT_POL Register Diagram

Table 13-23: PORT_POL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Polarity Invert.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.

Table 13-23: PORT_POL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	PX14	Port x Bit 14 Polarity Invert.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.
13 (R/W)	PX13	Port x Bit 13 Polarity Invert.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.
12 (R/W)	PX12	Port x Bit 12 Polarity Invert.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.
11 (R/W)	PX11	Port x Bit 11 Polarity Invert.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.
10 (R/W)	PX10	Port x Bit 10 Polarity Invert.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.
9 (R/W)	PX9	Port x Bit 9 Polarity Invert.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.
8 (R/W)	PX8	Port x Bit 8 Polarity Invert.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.
7 (R/W)	PX7	Port x Bit 7 Polarity Invert.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.
6 (R/W)	PX6	Port x Bit 6 Polarity Invert.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.
5 (R/W)	PX5	Port x Bit 5 Polarity Invert.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.

Table 13-23: PORT_POL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	PX4	Port x Bit 4 Polarity Invert.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.
3 (R/W)	PX3	Port x Bit 3 Polarity Invert.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.
2 (R/W)	PX2	Port x Bit 2 Polarity Invert.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.
1 (R/W)	PX1	Port x Bit 1 Polarity Invert.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.
0 (R/W)	PX0	Port x Bit 0 Polarity Invert.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.

Port x GPIO Polarity Invert Clear Register

The PORT_POL_CLR register disables polarity inversion for GPIO pins. For more information, see the PORT_POL register description.

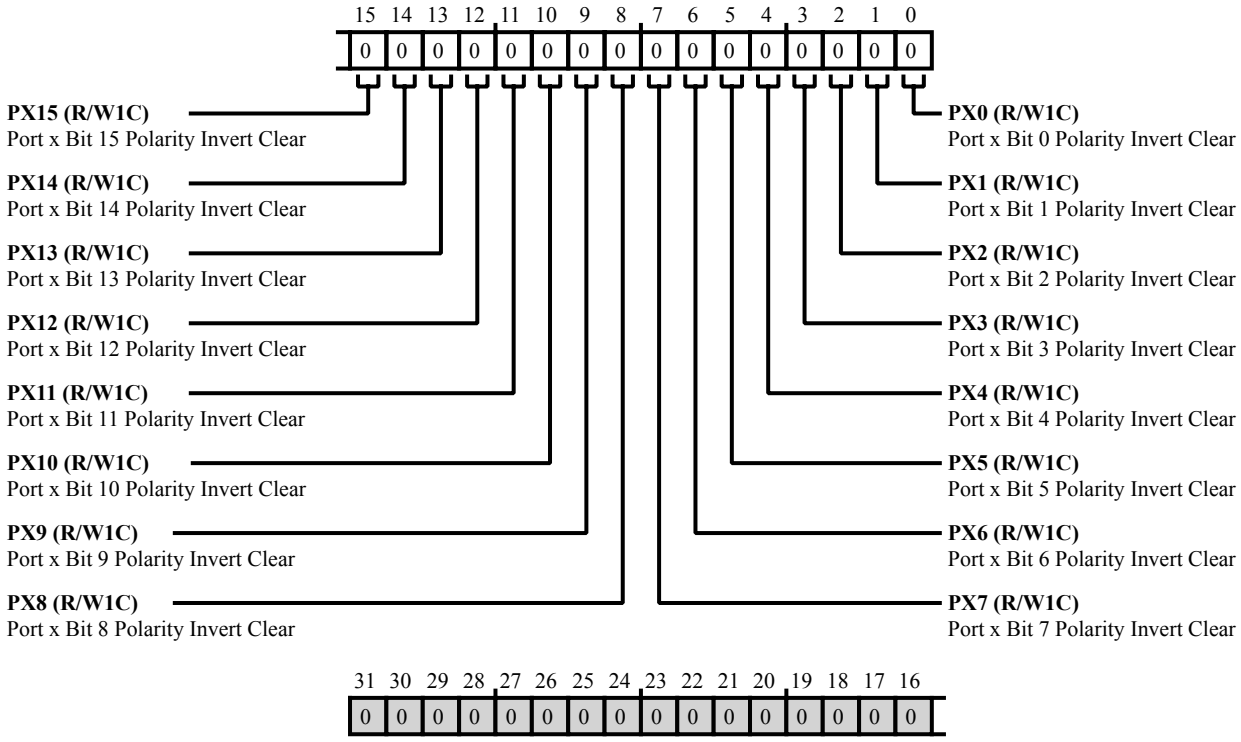

Figure 13-23: PORT_POL_CLR Register Diagram

Table 13-24: PORT_POL_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	PX15	Port x Bit 15 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit Set to disable GPIO pin polarity invert.
14 (R/W1C)	PX14	Port x Bit 14 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit Set to disable GPIO pin polarity invert.
13 (R/W1C)	PX13	Port x Bit 13 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit Set to disable GPIO pin polarity invert.

Table 13-24: PORT_POL_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1C)	PX12	Port x Bit 12 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit Set to disable GPIO pin polarity invert.
11 (R/W1C)	PX11	Port x Bit 11 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit Set to disable GPIO pin polarity invert.
10 (R/W1C)	PX10	Port x Bit 10 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit Set to disable GPIO pin polarity invert.
9 (R/W1C)	PX9	Port x Bit 9 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit Set to disable GPIO pin polarity invert.
8 (R/W1C)	PX8	Port x Bit 8 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit Set to disable GPIO pin polarity invert.
7 (R/W1C)	PX7	Port x Bit 7 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit Set to disable GPIO pin polarity invert.
6 (R/W1C)	PX6	Port x Bit 6 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit Set to disable GPIO pin polarity invert.
5 (R/W1C)	PX5	Port x Bit 5 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit Set to disable GPIO pin polarity invert.
4 (R/W1C)	PX4	Port x Bit 4 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit Set to disable GPIO pin polarity invert.
3 (R/W1C)	PX3	Port x Bit 3 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit Set to disable GPIO pin polarity invert.

Table 13-24: PORT_POL_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	PX2	Port x Bit 2 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit Set to disable GPIO pin polarity invert.
1 (R/W1C)	PX1	Port x Bit 1 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit Set to disable GPIO pin polarity invert.
0 (R/W1C)	PX0	Port x Bit 0 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit Set to disable GPIO pin polarity invert.

Port x GPIO Polarity Invert Set Register

The PORT_POL_SET register enables polarity inversion for GPIO pins. For more information, see the PORT_POL register description.

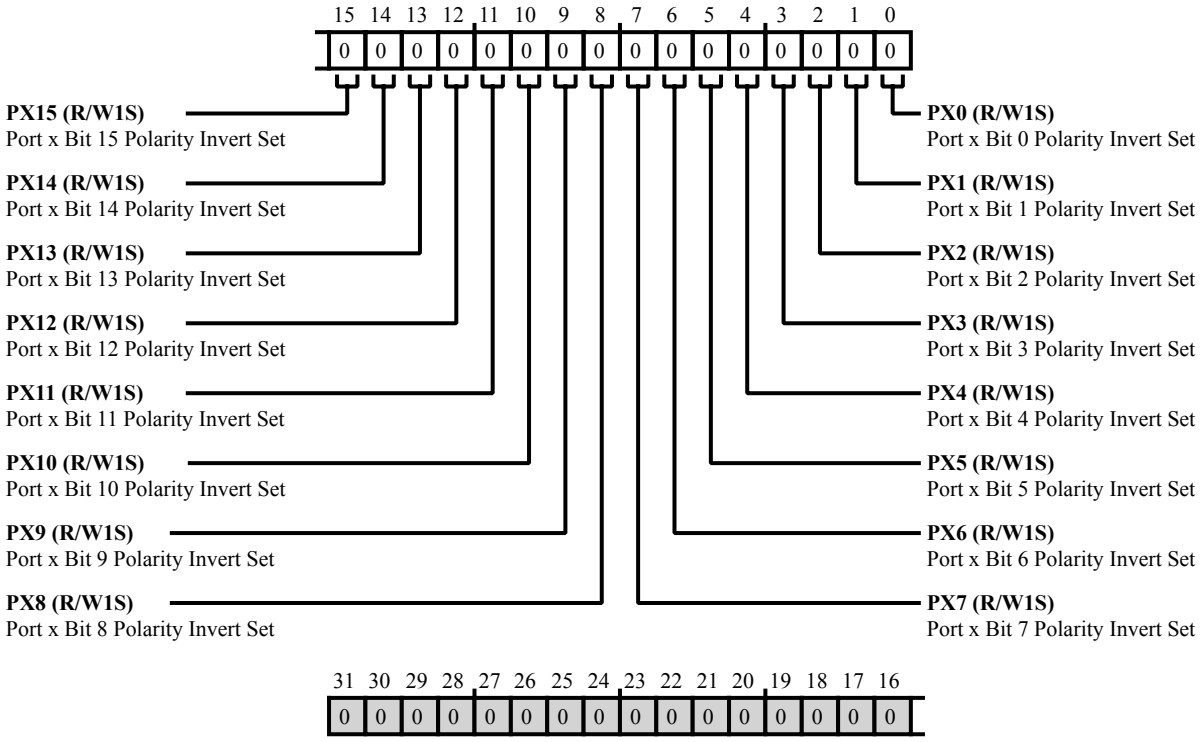

Figure 13-24: PORT_POL_SET Register Diagram

Table 13-25: PORT_POL_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1S)	PX15	Port x Bit 15 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
14 (R/W1S)	PX14	Port x Bit 14 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
13 (R/W1S)	PX13	Port x Bit 13 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.

Table 13-25: PORT_POL_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1S)	PX12	Port x Bit 12 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
11 (R/W1S)	PX11	Port x Bit 11 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
10 (R/W1S)	PX10	Port x Bit 10 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
9 (R/W1S)	PX9	Port x Bit 9 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
8 (R/W1S)	PX8	Port x Bit 8 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
7 (R/W1S)	PX7	Port x Bit 7 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
6 (R/W1S)	PX6	Port x Bit 6 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
5 (R/W1S)	PX5	Port x Bit 5 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
4 (R/W1S)	PX4	Port x Bit 4 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
3 (R/W1S)	PX3	Port x Bit 3 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.

Table 13-25: PORT_POL_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1S)	PX2	Port x Bit 2 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
1 (R/W1S)	PX1	Port x Bit 1 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
0 (R/W1S)	PX0	Port x Bit 0 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.

ADSP-CM40x PINT Register Descriptions

PINT (PINT) contains the following registers.

Table 13-26: ADSP-CM40x PINT Register List

Name	Description
PINT_ASSIGN	Pint Assign Register
PINT_EDGE_CLR	Pint Edge Clear Register
PINT_EDGE_SET	Pint Edge Set Register
PINT_INV_CLR	Pint Invert Clear Register
PINT_INV_SET	Pint Invert Set Register
PINT_LATCH	Pint Latch Register
PINT_MSK_CLR	Pint Mask Clear Register
PINT_MSK_SET	Pint Mask Set Register
PINT_PINSTATE	Pint Pinstate Register
PINT_REQ	Pint Request Register

Pint Assign Register

The PINT_ASSIGN register controls the pin-to-interrupt assignment in a byte-wide manner. This register consists of four control bytes that each function as a multiplexer control.

The PINT ports are subdivided into 8-bit half ports, resulting in lower and upper half 8-bit units. Using the multiplexers controlled by the PINT_ASSIGN register, the lower half units of eight pins can be forwarded to either byte 0 or byte 2 of either associated PINT block, and the upper half units can be forwarded to either byte 1 or byte 3 of the PINT block, without further restrictions.

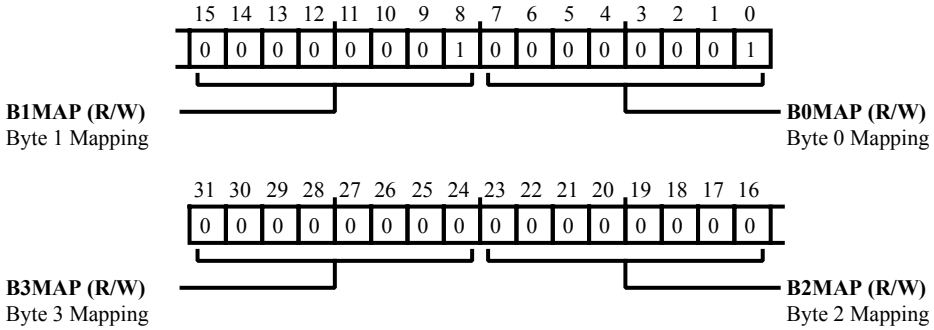

Figure 13-25: PINT_ASSIGN Register Diagram

Table 13-27: PINT_ASSIGN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:24 (R/W)	B3MAP	Byte 3 Mapping.	
		0	B3MAP_PAH Byte 3 = PA.H
		1	B3MAP_PBH Byte 3 = PB.H
23:16 (R/W)	B2MAP	Byte 2 Mapping.	
		0	B2MAP_PAL Byte 2 = PA.L
		1	B2MAP_PBL Byte 2 = PB.L
15:8 (R/W)	B1MAP	Byte 1 Mapping.	
		0	B1MAP_PAH Byte 1 = PA.H
		1	B1MAP_PBH Byte 1 = PB.H
7:0 (R/W)	B0MAP	Byte 0 Mapping.	
		0	B0MAP_PAL Byte 0 = PA.L
		1	B0MAP_PBL Byte 0 = PB.L

Pint Edge Clear Register

The PINT_EDGE_CLR register permits selecting level-sensitive interrupts. Writing 1 to a bit in PINT_EDGE_CLR enables level sensitivity for the corresponding pin interrupt.

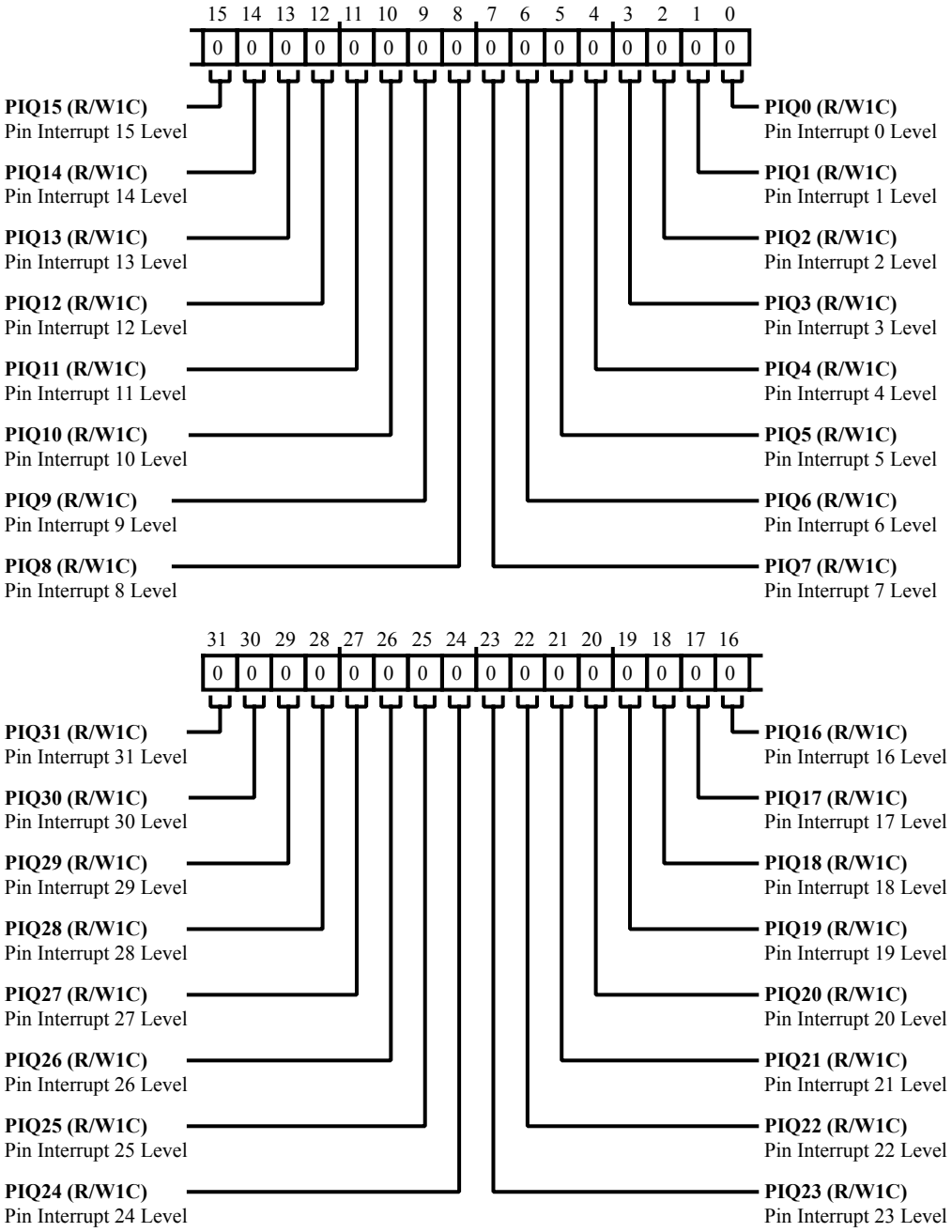

Figure 13-26: PINT_EDGE_CLR Register Diagram

Table 13-28: PINT_EDGE_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Level. Set the PINT_EDGE_CLR.PIQ31 bit to enable level sensitivity.
30 (R/W1C)	PIQ30	Pin Interrupt 30 Level. Set the PINT_EDGE_CLR.PIQ30 bit to enable level sensitivity.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Level. Set the PINT_EDGE_CLR.PIQ29 bit to enable level sensitivity.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Level. Set the PINT_EDGE_CLR.PIQ28 bit to enable level sensitivity.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Level. Set the PINT_EDGE_CLR.PIQ27 bit to enable level sensitivity.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Level. Set the PINT_EDGE_CLR.PIQ26 bit to enable level sensitivity.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Level. Set the PINT_EDGE_CLR.PIQ25 bit to enable level sensitivity.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Level. Set the PINT_EDGE_CLR.PIQ24 bit to enable level sensitivity.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Level. Set the PINT_EDGE_CLR.PIQ23 bit to enable level sensitivity.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Level. Set the PINT_EDGE_CLR.PIQ22 bit to enable level sensitivity.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Level. Set the PINT_EDGE_CLR.PIQ21 bit to enable level sensitivity.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Level. Set the PINT_EDGE_CLR.PIQ20 bit to enable level sensitivity.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Level. Set the PINT_EDGE_CLR.PIQ19 bit to enable level sensitivity.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Level. Set the PINT_EDGE_CLR.PIQ18 bit to enable level sensitivity.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Level. Set the PINT_EDGE_CLR.PIQ17 bit to enable level sensitivity.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Level. Set the PINT_EDGE_CLR.PIQ16 bit to enable level sensitivity.

Table 13-28: PINT_EDGE_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	PIQ15	Pin Interrupt 15 Level. Set the PINT_EDGE_CLR.PIQ15 bit to enable level sensitivity.
14 (R/W1C)	PIQ14	Pin Interrupt 14 Level. Set the PINT_EDGE_CLR.PIQ14 bit to enable level sensitivity.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Level. Set the PINT_EDGE_CLR.PIQ13 bit to enable level sensitivity.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Level. Set the PINT_EDGE_CLR.PIQ12 bit to enable level sensitivity.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Level. Set the PINT_EDGE_CLR.PIQ11 bit to enable level sensitivity.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Level. Set the PINT_EDGE_CLR.PIQ10 bit to enable level sensitivity.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Level. Set the PINT_EDGE_CLR.PIQ9 bit to enable level sensitivity.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Level. Set the PINT_EDGE_CLR.PIQ8 bit to enable level sensitivity.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Level. Set the PINT_EDGE_CLR.PIQ7 bit to enable level sensitivity.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Level. Set the PINT_EDGE_CLR.PIQ6 bit to enable level sensitivity.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Level. Set the PINT_EDGE_CLR.PIQ5 bit to enable level sensitivity.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Level. Set the PINT_EDGE_CLR.PIQ4 bit to enable level sensitivity.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Level. Set the PINT_EDGE_CLR.PIQ3 bit to enable level sensitivity.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Level. Set the PINT_EDGE_CLR.PIQ2 bit to enable level sensitivity.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Level. Set the PINT_EDGE_CLR.PIQ1 bit to enable level sensitivity.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Level. Set the PINT_EDGE_CLR.PIQ0 bit to enable level sensitivity.

Pint Edge Set Register

The PINT_EDGE_SET register permits selecting edge-sensitive interrupts. Writing 1 to a bit in PINT_EDGE_SET enables edge sensitivity for the corresponding pin interrupt.

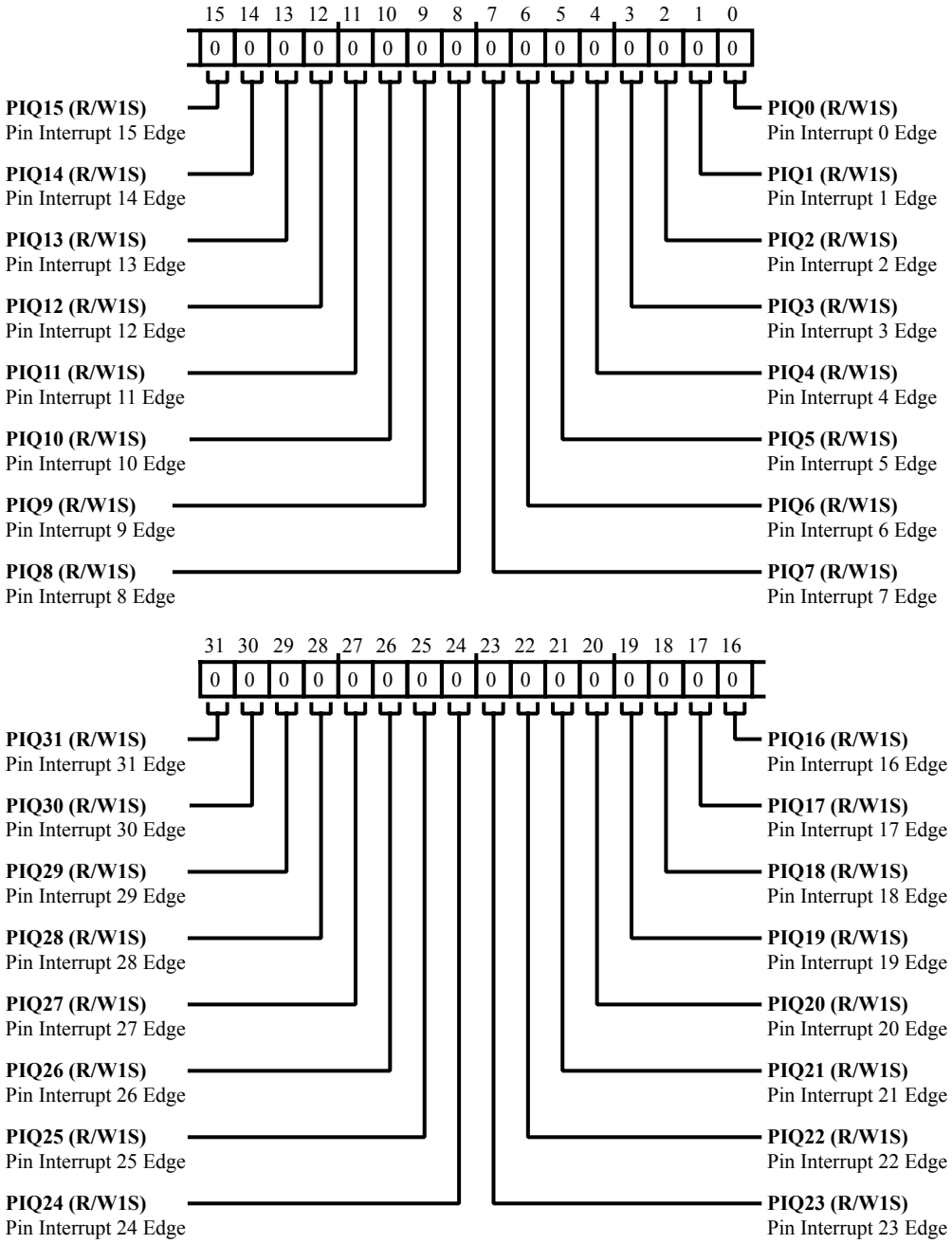

Figure 13-27: PINT_EDGE_SET Register Diagram

Table 13-29: PINT_EDGE_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1S)	PIQ31	Pin Interrupt 31 Edge. Set the PINT_EDGE_SET.PIQ31 bit to enable edge sensitivity.
30 (R/W1S)	PIQ30	Pin Interrupt 30 Edge. Set the PINT_EDGE_SET.PIQ30 bit to enable edge sensitivity.
29 (R/W1S)	PIQ29	Pin Interrupt 29 Edge. Set the PINT_EDGE_SET.PIQ29 bit to enable edge sensitivity.
28 (R/W1S)	PIQ28	Pin Interrupt 28 Edge. Set the PINT_EDGE_SET.PIQ28 bit to enable edge sensitivity.
27 (R/W1S)	PIQ27	Pin Interrupt 27 Edge. Set the PINT_EDGE_SET.PIQ27 bit to enable edge sensitivity.
26 (R/W1S)	PIQ26	Pin Interrupt 26 Edge. Set the PINT_EDGE_SET.PIQ26 bit to enable edge sensitivity.
25 (R/W1S)	PIQ25	Pin Interrupt 25 Edge. Set the PINT_EDGE_SET.PIQ25 bit to enable edge sensitivity.
24 (R/W1S)	PIQ24	Pin Interrupt 24 Edge. Set the PINT_EDGE_SET.PIQ24 bit to enable edge sensitivity.
23 (R/W1S)	PIQ23	Pin Interrupt 23 Edge. Set the PINT_EDGE_SET.PIQ23 bit to enable edge sensitivity.
22 (R/W1S)	PIQ22	Pin Interrupt 22 Edge. Set the PINT_EDGE_SET.PIQ22 bit to enable edge sensitivity.
21 (R/W1S)	PIQ21	Pin Interrupt 21 Edge. Set the PINT_EDGE_SET.PIQ21 bit to enable edge sensitivity.
20 (R/W1S)	PIQ20	Pin Interrupt 20 Edge. Set the PINT_EDGE_SET.PIQ20 bit to enable edge sensitivity.
19 (R/W1S)	PIQ19	Pin Interrupt 19 Edge. Set the PINT_EDGE_SET.PIQ19 bit to enable edge sensitivity.
18 (R/W1S)	PIQ18	Pin Interrupt 18 Edge. Set the PINT_EDGE_SET.PIQ18 bit to enable edge sensitivity.
17 (R/W1S)	PIQ17	Pin Interrupt 17 Edge. Set the PINT_EDGE_SET.PIQ17 bit to enable edge sensitivity.
16 (R/W1S)	PIQ16	Pin Interrupt 16 Edge. Set the PINT_EDGE_SET.PIQ16 bit to enable edge sensitivity.

Table 13-29: PINT_EDGE_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1S)	PIQ15	Pin Interrupt 15 Edge. Set the PINT_EDGE_SET.PIQ15 bit to enable edge sensitivity.
14 (R/W1S)	PIQ14	Pin Interrupt 14 Edge. Set the PINT_EDGE_SET.PIQ14 bit to enable edge sensitivity.
13 (R/W1S)	PIQ13	Pin Interrupt 13 Edge. Set the PINT_EDGE_SET.PIQ13 bit to enable edge sensitivity.
12 (R/W1S)	PIQ12	Pin Interrupt 12 Edge. Set the PINT_EDGE_SET.PIQ12 bit to enable edge sensitivity.
11 (R/W1S)	PIQ11	Pin Interrupt 11 Edge. Set the PINT_EDGE_SET.PIQ11 bit to enable edge sensitivity.
10 (R/W1S)	PIQ10	Pin Interrupt 10 Edge. Set the PINT_EDGE_SET.PIQ10 bit to enable edge sensitivity.
9 (R/W1S)	PIQ9	Pin Interrupt 9 Edge. Set the PINT_EDGE_SET.PIQ9 bit to enable edge sensitivity.
8 (R/W1S)	PIQ8	Pin Interrupt 8 Edge. Set the PINT_EDGE_SET.PIQ8 bit to enable edge sensitivity.
7 (R/W1S)	PIQ7	Pin Interrupt 7 Edge. Set the PINT_EDGE_SET.PIQ7 bit to enable edge sensitivity.
6 (R/W1S)	PIQ6	Pin Interrupt 6 Edge. Set the PINT_EDGE_SET.PIQ6 bit to enable edge sensitivity.
5 (R/W1S)	PIQ5	Pin Interrupt 5 Edge. Set the PINT_EDGE_SET.PIQ5 bit to enable edge sensitivity.
4 (R/W1S)	PIQ4	Pin Interrupt 4 Edge. Set the PINT_EDGE_SET.PIQ4 bit to enable edge sensitivity.
3 (R/W1S)	PIQ3	Pin Interrupt 3 Edge. Set the PINT_EDGE_SET.PIQ3 bit to enable edge sensitivity.
2 (R/W1S)	PIQ2	Pin Interrupt 2 Edge. Set the PINT_EDGE_SET.PIQ2 bit to enable edge sensitivity.
1 (R/W1S)	PIQ1	Pin Interrupt 1 Edge. Set the PINT_EDGE_SET.PIQ1 bit to enable edge sensitivity.
0 (R/W1S)	PIQ0	Pin Interrupt 0 Edge. Set the PINT_EDGE_SET.PIQ0 bit to enable edge sensitivity.

Pint Invert Clear Register

The PINT_INV_CLR register disables inverting input polarity. Writing 1 to a bit in PINT_INV_CLR disables an inverter for input on the corresponding pin.

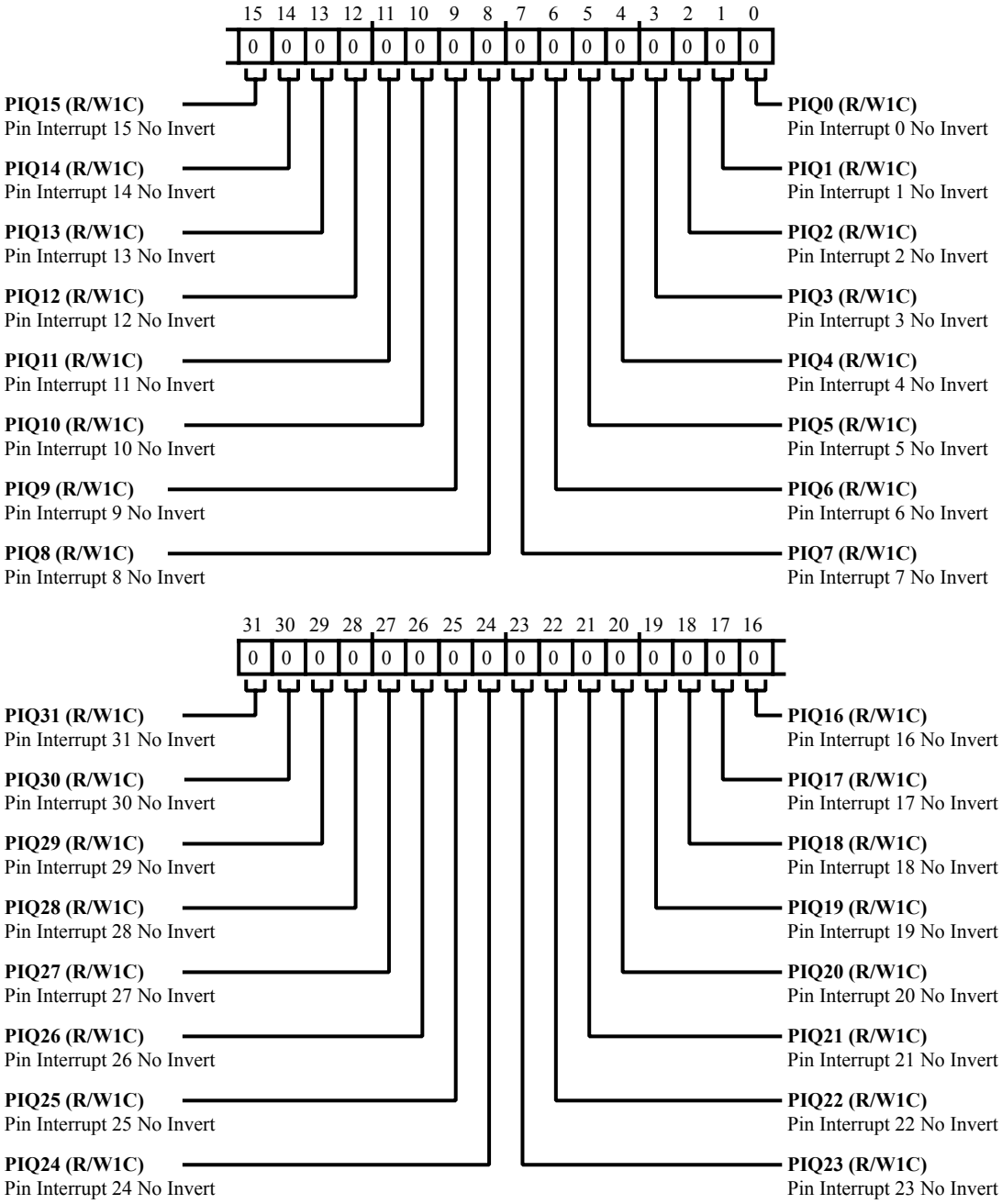

Figure 13-28: PINT_INV_CLR Register Diagram

Table 13-30: PINT_INV_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 No Invert. Set the PINT_INV_CLR.PIQ31 bit to disable inverted input.
30 (R/W1C)	PIQ30	Pin Interrupt 30 No Invert. Set the PINT_INV_CLR.PIQ30 bit to disable inverted input.
29 (R/W1C)	PIQ29	Pin Interrupt 29 No Invert. Set the PINT_INV_CLR.PIQ29 bit to disable inverted input.
28 (R/W1C)	PIQ28	Pin Interrupt 28 No Invert. Set the PINT_INV_CLR.PIQ28 bit to disable inverted input.
27 (R/W1C)	PIQ27	Pin Interrupt 27 No Invert. Set the PINT_INV_CLR.PIQ27 bit to disable inverted input.
26 (R/W1C)	PIQ26	Pin Interrupt 26 No Invert. Set the PINT_INV_CLR.PIQ26 bit to disable inverted input.
25 (R/W1C)	PIQ25	Pin Interrupt 25 No Invert. Set the PINT_INV_CLR.PIQ25 bit to disable inverted input.
24 (R/W1C)	PIQ24	Pin Interrupt 24 No Invert. Set the PINT_INV_CLR.PIQ24 bit to disable inverted input.
23 (R/W1C)	PIQ23	Pin Interrupt 23 No Invert. Set the PINT_INV_CLR.PIQ23 bit to disable inverted input.
22 (R/W1C)	PIQ22	Pin Interrupt 22 No Invert. Set the PINT_INV_CLR.PIQ22 bit to disable inverted input.
21 (R/W1C)	PIQ21	Pin Interrupt 21 No Invert. Set the PINT_INV_CLR.PIQ21 bit to disable inverted input.
20 (R/W1C)	PIQ20	Pin Interrupt 20 No Invert. Set the PINT_INV_CLR.PIQ20 bit to disable inverted input.
19 (R/W1C)	PIQ19	Pin Interrupt 19 No Invert. Set the PINT_INV_CLR.PIQ19 bit to disable inverted input.
18 (R/W1C)	PIQ18	Pin Interrupt 18 No Invert. Set the PINT_INV_CLR.PIQ18 bit to disable inverted input.
17 (R/W1C)	PIQ17	Pin Interrupt 17 No Invert. Set the PINT_INV_CLR.PIQ17 bit to disable inverted input.
16 (R/W1C)	PIQ16	Pin Interrupt 16 No Invert. Set the PINT_INV_CLR.PIQ16 bit to disable inverted input.

Table 13-30: PINT_INV_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	PIQ15	Pin Interrupt 15 No Invert. Set the PINT_INV_CLR.PIQ15 bit to disable inverted input.
14 (R/W1C)	PIQ14	Pin Interrupt 14 No Invert. Set the PINT_INV_CLR.PIQ14 bit to disable inverted input.
13 (R/W1C)	PIQ13	Pin Interrupt 13 No Invert. Set the PINT_INV_CLR.PIQ13 bit to disable inverted input.
12 (R/W1C)	PIQ12	Pin Interrupt 12 No Invert. Set the PINT_INV_CLR.PIQ12 bit to disable inverted input.
11 (R/W1C)	PIQ11	Pin Interrupt 11 No Invert. Set the PINT_INV_CLR.PIQ11 bit to disable inverted input.
10 (R/W1C)	PIQ10	Pin Interrupt 10 No Invert. Set the PINT_INV_CLR.PIQ10 bit to disable inverted input.
9 (R/W1C)	PIQ9	Pin Interrupt 9 No Invert. Set the PINT_INV_CLR.PIQ9 bit to disable inverted input.
8 (R/W1C)	PIQ8	Pin Interrupt 8 No Invert. Set the PINT_INV_CLR.PIQ8 bit to disable inverted input.
7 (R/W1C)	PIQ7	Pin Interrupt 7 No Invert. Set the PINT_INV_CLR.PIQ7 bit to disable inverted input.
6 (R/W1C)	PIQ6	Pin Interrupt 6 No Invert. Set the PINT_INV_CLR.PIQ6 bit to disable inverted input.
5 (R/W1C)	PIQ5	Pin Interrupt 5 No Invert. Set the PINT_INV_CLR.PIQ5 bit to disable inverted input.
4 (R/W1C)	PIQ4	Pin Interrupt 4 No Invert. Set the PINT_INV_CLR.PIQ4 bit to disable inverted input.
3 (R/W1C)	PIQ3	Pin Interrupt 3 No Invert. Set the PINT_INV_CLR.PIQ3 bit to disable inverted input.
2 (R/W1C)	PIQ2	Pin Interrupt 2 No Invert. Set the PINT_INV_CLR.PIQ2 bit to disable inverted input.
1 (R/W1C)	PIQ1	Pin Interrupt 1 No Invert. Set the PINT_INV_CLR.PIQ1 bit to disable inverted input.
0 (R/W1C)	PIQ0	Pin Interrupt 0 No Invert. Set the PINT_INV_CLR.PIQ0 bit to disable inverted input.

Pint Invert Set Register

The PINT_INV_SET register enables inverting input polarity. Writing 1 to a bit in PINT_INV_SET enables an inverter for input on the corresponding pin.

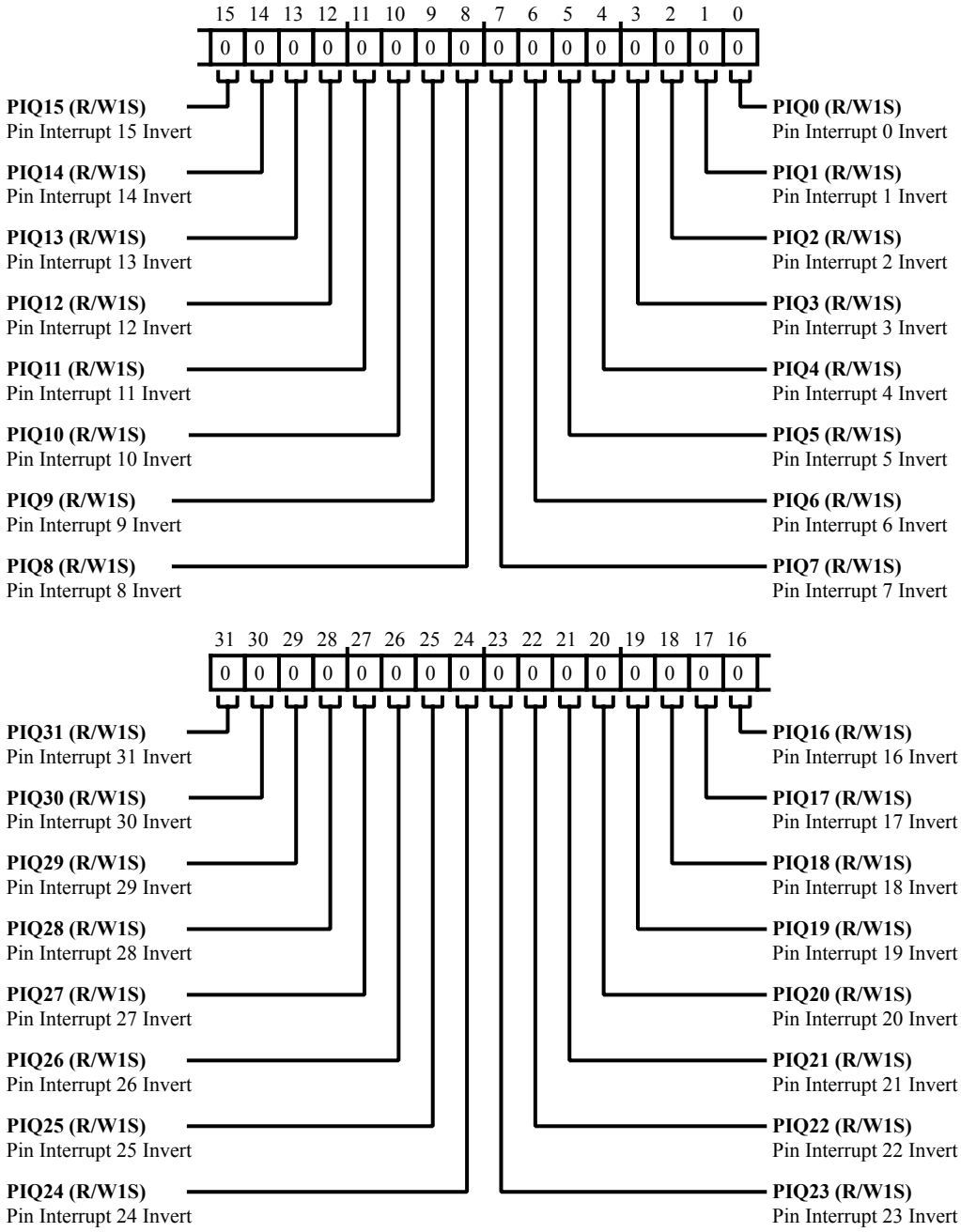

Figure 13-29: PINT_INV_SET Register Diagram

Table 13-31: PINT_INV_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1S)	PIQ31	Pin Interrupt 31 Invert. Set the PINT_INV_SET.PIQ31 bit to enable inverted input.
30 (R/W1S)	PIQ30	Pin Interrupt 30 Invert. Set the PINT_INV_SET.PIQ30 bit to enable inverted input.
29 (R/W1S)	PIQ29	Pin Interrupt 29 Invert. Set the PINT_INV_SET.PIQ29 bit to enable inverted input.
28 (R/W1S)	PIQ28	Pin Interrupt 28 Invert. Set the PINT_INV_SET.PIQ28 bit to enable inverted input.
27 (R/W1S)	PIQ27	Pin Interrupt 27 Invert. Set the PINT_INV_SET.PIQ27 bit to enable inverted input.
26 (R/W1S)	PIQ26	Pin Interrupt 26 Invert. Set the PINT_INV_SET.PIQ26 bit to enable inverted input.
25 (R/W1S)	PIQ25	Pin Interrupt 25 Invert. Set the PINT_INV_SET.PIQ25 bit to enable inverted input.
24 (R/W1S)	PIQ24	Pin Interrupt 24 Invert. Set the PINT_INV_SET.PIQ24 bit to enable inverted input.
23 (R/W1S)	PIQ23	Pin Interrupt 23 Invert. Set the PINT_INV_SET.PIQ23 bit to enable inverted input.
22 (R/W1S)	PIQ22	Pin Interrupt 22 Invert. Set the PINT_INV_SET.PIQ22 bit to enable inverted input.
21 (R/W1S)	PIQ21	Pin Interrupt 21 Invert. Set the PINT_INV_SET.PIQ21 bit to enable inverted input.
20 (R/W1S)	PIQ20	Pin Interrupt 20 Invert. Set the PINT_INV_SET.PIQ20 bit to enable inverted input.
19 (R/W1S)	PIQ19	Pin Interrupt 19 Invert. Set the PINT_INV_SET.PIQ19 bit to enable inverted input.
18 (R/W1S)	PIQ18	Pin Interrupt 18 Invert. Set the PINT_INV_SET.PIQ18 bit to enable inverted input.
17 (R/W1S)	PIQ17	Pin Interrupt 17 Invert. Set the PINT_INV_SET.PIQ17 bit to enable inverted input.
16 (R/W1S)	PIQ16	Pin Interrupt 16 Invert. Set the PINT_INV_SET.PIQ16 bit to enable inverted input.

Table 13-31: PINT_INV_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1S)	PIQ15	Pin Interrupt 15 Invert. Set the PINT_INV_SET.PIQ15 bit to enable inverted input.
14 (R/W1S)	PIQ14	Pin Interrupt 14 Invert. Set the PINT_INV_SET.PIQ14 bit to enable inverted input.
13 (R/W1S)	PIQ13	Pin Interrupt 13 Invert. Set the PINT_INV_SET.PIQ13 bit to enable inverted input.
12 (R/W1S)	PIQ12	Pin Interrupt 12 Invert. Set the PINT_INV_SET.PIQ12 bit to enable inverted input.
11 (R/W1S)	PIQ11	Pin Interrupt 11 Invert. Set the PINT_INV_SET.PIQ11 bit to enable inverted input.
10 (R/W1S)	PIQ10	Pin Interrupt 10 Invert. Set the PINT_INV_SET.PIQ10 bit to enable inverted input.
9 (R/W1S)	PIQ9	Pin Interrupt 9 Invert. Set the PINT_INV_SET.PIQ9 bit to enable inverted input.
8 (R/W1S)	PIQ8	Pin Interrupt 8 Invert. Set the PINT_INV_SET.PIQ8 bit to enable inverted input.
7 (R/W1S)	PIQ7	Pin Interrupt 7 Invert. Set the PINT_INV_SET.PIQ7 bit to enable inverted input.
6 (R/W1S)	PIQ6	Pin Interrupt 6 Invert. Set the PINT_INV_SET.PIQ6 bit to enable inverted input.
5 (R/W1S)	PIQ5	Pin Interrupt 5 Invert. Set the PINT_INV_SET.PIQ5 bit to enable inverted input.
4 (R/W1S)	PIQ4	Pin Interrupt 4 Invert. Set the PINT_INV_SET.PIQ4 bit to enable inverted input.
3 (R/W1S)	PIQ3	Pin Interrupt 3 Invert. Set the PINT_INV_SET.PIQ3 bit to enable inverted input.
2 (R/W1S)	PIQ2	Pin Interrupt 2 Invert. Set the PINT_INV_SET.PIQ2 bit to enable inverted input.
1 (R/W1S)	PIQ1	Pin Interrupt 1 Invert. Set the PINT_INV_SET.PIQ1 bit to enable inverted input.
0 (R/W1S)	PIQ0	Pin Interrupt 0 Invert. Set the PINT_INV_SET.PIQ0 bit to enable inverted input.

Pint Latch Register

The PINT_LATCH register indicates interrupt latch status for pin interrupts. When set, an interrupt request is latched. When cleared, there is no interrupt request latched.

Both the PINT_REQ and PINT_LATCH registers indicate whether an interrupt request is latched on the respective pin. The PINT_LATCH register is a latch that operates regardless of the interrupt masks. Bits of the PINT_REQ register depend on the mask register. The PINT_REQ register is a logical AND of the PINT_LATCH register and the interrupt mask.

Having two separate registers here enables the user to interrogate certain pins in polling mode while others work in interrupt mode. The PINT_LATCH registers can be used for edge detection or pin activity detection.

Both registers have W1C behavior. Writing a 1 to either clears respective bits in both registers. For interrupt operation, the user may prefer to W1C the PINT_REQ register (address still loaded in Px pointer). In polling mode it might be cleaner to W1C the PINT_LATCH register.

Regardless whether in edge-sensitive mode or level-sensitive mode, PINT_LATCH bits are never cleared by hardware except at system reset. Even in level-sensitive mode, the PINT_LATCH register functions as latch.

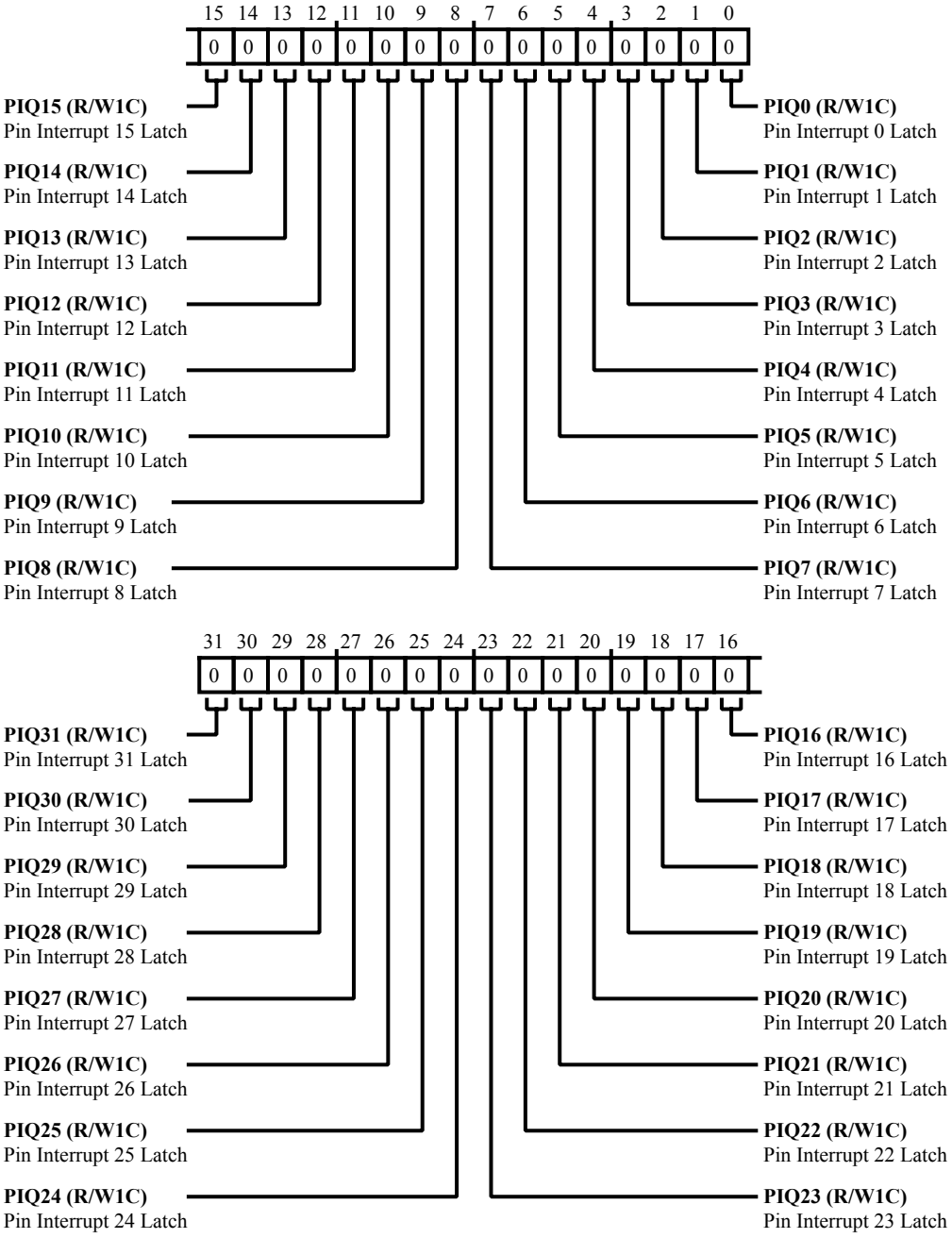

Figure 13-30: PINT_LATCH Register Diagram

Table 13-32: PINT_LATCH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Latch. If the `PINT_LATCH.PIQ31` bit is set, the request is latched.

Table 13-32: PINT_LATCH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/W1C)	PIQ30	Pin Interrupt 30 Latch. If the PINT_LATCH.PIQ30 bit is set, the request is latched.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Latch. If the PINT_LATCH.PIQ29 bit is set, the request is latched.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Latch. If the PINT_LATCH.PIQ28 bit is set, the request is latched.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Latch. If the PINT_LATCH.PIQ27 bit is set, the request is latched.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Latch. If the PINT_LATCH.PIQ26 bit is set, the request is latched.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Latch. If the PINT_LATCH.PIQ25 bit is set, the request is latched.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Latch. If the PINT_LATCH.PIQ24 bit is set, the request is latched.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Latch. If the PINT_LATCH.PIQ23 bit is set, the request is latched.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Latch. If the PINT_LATCH.PIQ22 bit is set, the request is latched.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Latch. If the PINT_LATCH.PIQ21 bit is set, the request is latched.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Latch. If the PINT_LATCH.PIQ20 bit is set, the request is latched.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Latch. If the PINT_LATCH.PIQ19 bit is set, the request is latched.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Latch. If the PINT_LATCH.PIQ18 bit is set, the request is latched.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Latch. If the PINT_LATCH.PIQ17 bit is set, the request is latched.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Latch. If the PINT_LATCH.PIQ16 bit is set, the request is latched.
15 (R/W1C)	PIQ15	Pin Interrupt 15 Latch. If the PINT_LATCH.PIQ15 bit is set, the request is latched.

Table 13-32: PINT_LATCH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W1C)	PIQ14	Pin Interrupt 14 Latch. If the PINT_LATCH.PIQ14 bit is set, the request is latched.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Latch. If the PINT_LATCH.PIQ13 bit is set, the request is latched.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Latch. If the PINT_LATCH.PIQ12 bit is set, the request is latched.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Latch. If the PINT_LATCH.PIQ11 bit is set, the request is latched.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Latch. If the PINT_LATCH.PIQ10 bit is set, the request is latched.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Latch. If the PINT_LATCH.PIQ9 bit is set, the request is latched.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Latch. If the PINT_LATCH.PIQ8 bit is set, the request is latched.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Latch. If the PINT_LATCH.PIQ7 bit is set, the request is latched.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Latch. If the PINT_LATCH.PIQ6 bit is set, the request is latched.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Latch. If the PINT_LATCH.PIQ5 bit is set, the request is latched.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Latch. If the PINT_LATCH.PIQ4 bit is set, the request is latched.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Latch. If the PINT_LATCH.PIQ3 bit is set, the request is latched.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Latch. If the PINT_LATCH.PIQ2 bit is set, the request is latched.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Latch. If the PINT_LATCH.PIQ1 bit is set, the request is latched.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Latch. If the PINT_LATCH.PIQ0 bit is set, the request is latched.

Pint Mask Clear Register

The PINT_MSK_CLR register permits masking (disabling) of interrupts. Writing 1 to a bit in PINT_MSK_CLR masks the corresponding pin interrupt.

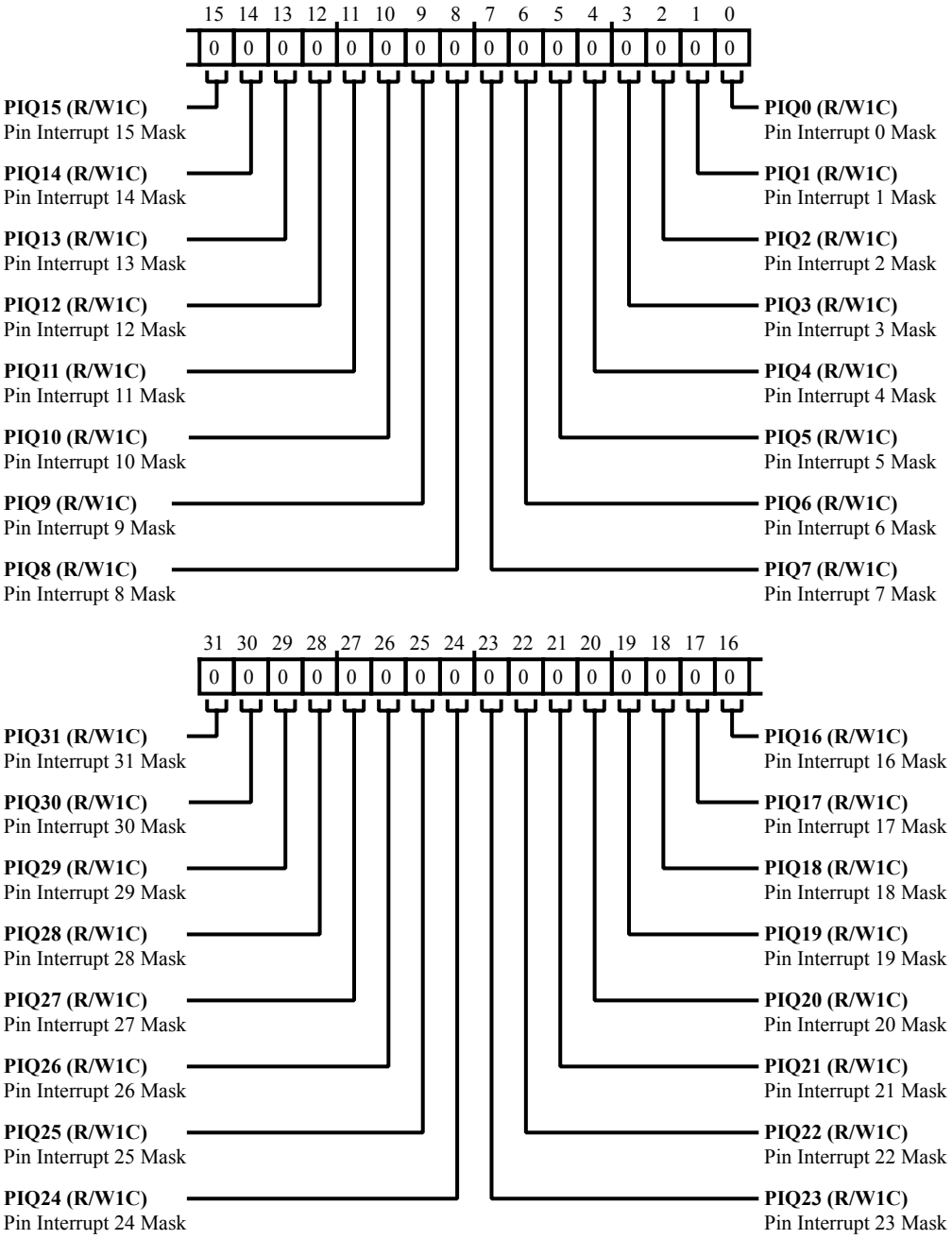

Figure 13-31: PINT_MSK_CLR Register Diagram

Table 13-33: PINT_MSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Mask. Set the PINT_MSK_CLR.PIQ31 bit to disable the interrupt.
30 (R/W1C)	PIQ30	Pin Interrupt 30 Mask. Set the PINT_MSK_CLR.PIQ30 bit to disable the interrupt.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Mask. Set the PINT_MSK_CLR.PIQ29 bit to disable the interrupt.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Mask. Set the PINT_MSK_CLR.PIQ28 bit to disable the interrupt.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Mask. Set the PINT_MSK_CLR.PIQ27 bit to disable the interrupt.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Mask. Set the PINT_MSK_CLR.PIQ26 bit to disable the interrupt.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Mask. Set the PINT_MSK_CLR.PIQ25 bit to disable the interrupt.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Mask. Set the PINT_MSK_CLR.PIQ24 bit to disable the interrupt.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Mask. Set the PINT_MSK_CLR.PIQ23 bit to disable the interrupt.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Mask. Set the PINT_MSK_CLR.PIQ22 bit to disable the interrupt.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Mask. Set the PINT_MSK_CLR.PIQ21 bit to disable the interrupt.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Mask. Set the PINT_MSK_CLR.PIQ20 bit to disable the interrupt.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Mask. Set the PINT_MSK_CLR.PIQ19 bit to disable the interrupt.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Mask. Set the PINT_MSK_CLR.PIQ18 bit to disable the interrupt.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Mask. Set the PINT_MSK_CLR.PIQ17 bit to disable the interrupt.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Mask. Set the PINT_MSK_CLR.PIQ16 bit to disable the interrupt.

Table 13-33: PINT_MSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	PIQ15	Pin Interrupt 15 Mask. Set the PINT_MSK_CLR.PIQ15 bit to disable the interrupt.
14 (R/W1C)	PIQ14	Pin Interrupt 14 Mask. Set the PINT_MSK_CLR.PIQ14 bit to disable the interrupt.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Mask. Set the PINT_MSK_CLR.PIQ13 bit to disable the interrupt.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Mask. Set the PINT_MSK_CLR.PIQ12 bit to disable the interrupt.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Mask. Set the PINT_MSK_CLR.PIQ11 bit to disable the interrupt.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Mask. Set the PINT_MSK_CLR.PIQ10 bit to disable the interrupt.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Mask. Set the PINT_MSK_CLR.PIQ9 bit to disable the interrupt.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Mask. Set the PINT_MSK_CLR.PIQ8 bit to disable the interrupt.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Mask. Set the PINT_MSK_CLR.PIQ7 bit to disable the interrupt.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Mask. Set the PINT_MSK_CLR.PIQ6 bit to disable the interrupt.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Mask. Set the PINT_MSK_CLR.PIQ5 bit to disable the interrupt.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Mask. Set the PINT_MSK_CLR.PIQ4 bit to disable the interrupt.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Mask. Set the PINT_MSK_CLR.PIQ3 bit to disable the interrupt.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Mask. Set the PINT_MSK_CLR.PIQ2 bit to disable the interrupt.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Mask. Set the PINT_MSK_CLR.PIQ1 bit to disable the interrupt.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Mask. Set the PINT_MSK_CLR.PIQ0 bit to disable the interrupt.

Pint Mask Set Register

The PINT_MSK_SET register permits unmasking (enabling) of interrupts. Writing 1 to a bit in PINT_MSK_SET unmasks the corresponding pin interrupt.

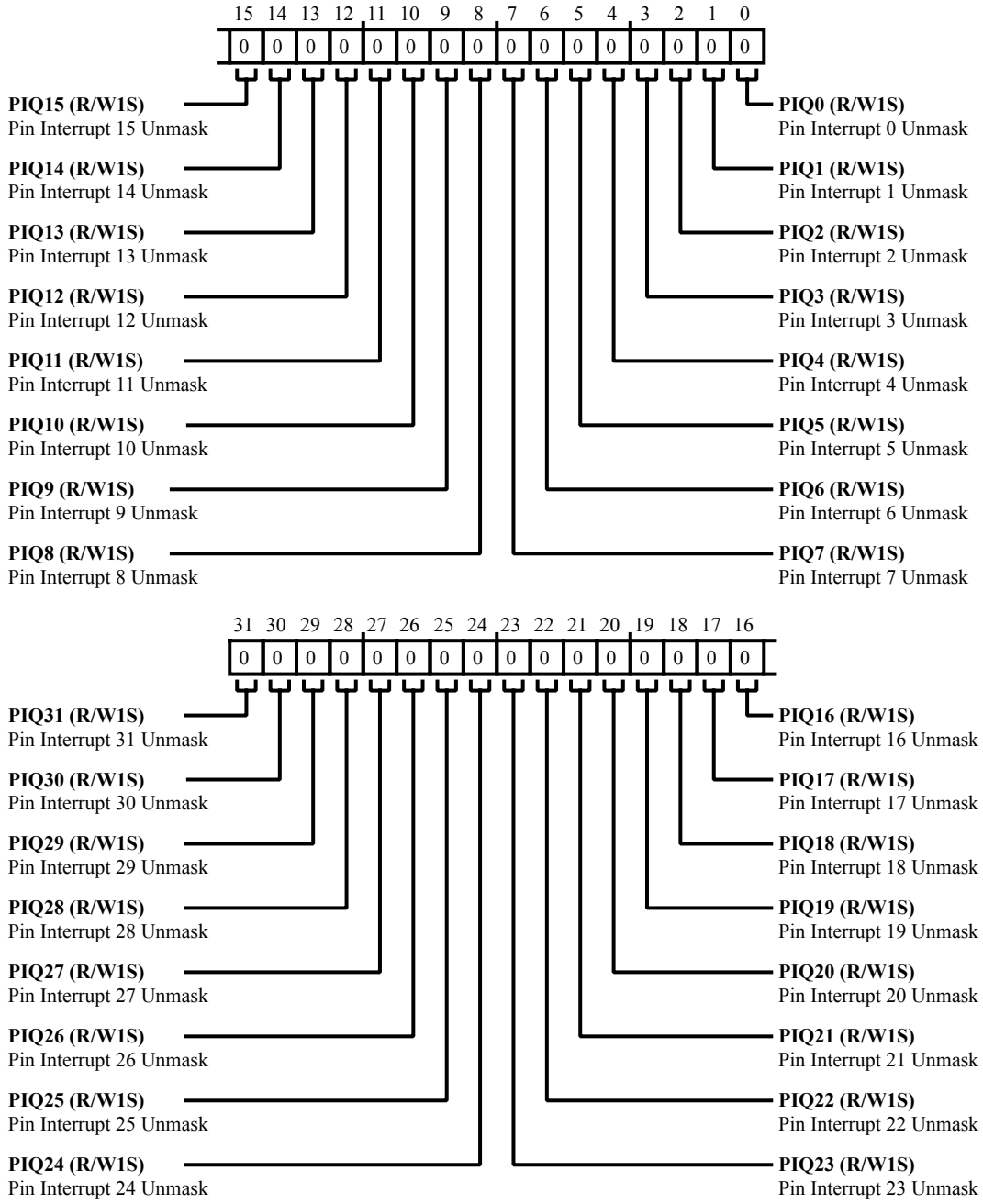

Figure 13-32: PINT_MSK_SET Register Diagram

Table 13-34: PINT_MSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1S)	PIQ31	Pin Interrupt 31 Unmask. Set to enable interrupt.
30 (R/W1S)	PIQ30	Pin Interrupt 30 Unmask. Set to enable interrupt.
29 (R/W1S)	PIQ29	Pin Interrupt 29 Unmask. Set to enable interrupt.
28 (R/W1S)	PIQ28	Pin Interrupt 28 Unmask. Set to enable interrupt.
27 (R/W1S)	PIQ27	Pin Interrupt 27 Unmask. Set to enable interrupt.
26 (R/W1S)	PIQ26	Pin Interrupt 26 Unmask. Set to enable interrupt.
25 (R/W1S)	PIQ25	Pin Interrupt 25 Unmask. Set to enable interrupt.
24 (R/W1S)	PIQ24	Pin Interrupt 24 Unmask. Set to enable interrupt.
23 (R/W1S)	PIQ23	Pin Interrupt 23 Unmask. Set to enable interrupt.
22 (R/W1S)	PIQ22	Pin Interrupt 22 Unmask. Set to enable interrupt.
21 (R/W1S)	PIQ21	Pin Interrupt 21 Unmask. Set to enable interrupt.
20 (R/W1S)	PIQ20	Pin Interrupt 20 Unmask. Set to enable interrupt.
19 (R/W1S)	PIQ19	Pin Interrupt 19 Unmask. Set to enable interrupt.
18 (R/W1S)	PIQ18	Pin Interrupt 18 Unmask. Set to enable interrupt.
17 (R/W1S)	PIQ17	Pin Interrupt 17 Unmask. Set to enable interrupt.
16 (R/W1S)	PIQ16	Pin Interrupt 16 Unmask. Set to enable interrupt.

Table 13-34: PINT_MSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1S)	PIQ15	Pin Interrupt 15 Unmask. Set to enable interrupt.
14 (R/W1S)	PIQ14	Pin Interrupt 14 Unmask. Set to enable interrupt.
13 (R/W1S)	PIQ13	Pin Interrupt 13 Unmask. Set to enable interrupt.
12 (R/W1S)	PIQ12	Pin Interrupt 12 Unmask. Set to enable interrupt.
11 (R/W1S)	PIQ11	Pin Interrupt 11 Unmask. Set to enable interrupt.
10 (R/W1S)	PIQ10	Pin Interrupt 10 Unmask. Set to enable interrupt.
9 (R/W1S)	PIQ9	Pin Interrupt 9 Unmask. Set to enable interrupt.
8 (R/W1S)	PIQ8	Pin Interrupt 8 Unmask. Set to enable interrupt.
7 (R/W1S)	PIQ7	Pin Interrupt 7 Unmask. Set to enable interrupt.
6 (R/W1S)	PIQ6	Pin Interrupt 6 Unmask. Set to enable interrupt.
5 (R/W1S)	PIQ5	Pin Interrupt 5 Unmask. Set to enable interrupt.
4 (R/W1S)	PIQ4	Pin Interrupt 4 Unmask. Set to enable interrupt.
3 (R/W1S)	PIQ3	Pin Interrupt 3 Unmask. Set to enable interrupt.
2 (R/W1S)	PIQ2	Pin Interrupt 2 Unmask. Set to enable interrupt.
1 (R/W1S)	PIQ1	Pin Interrupt 1 Unmask. Set to enable interrupt.
0 (R/W1S)	PIQ0	Pin Interrupt 0 Unmask. Set to enable interrupt.

Pint Pinstate Register

When a half port is assigned to a byte in any PINT block, the state of the eight pins (regardless of GPIO or function, input or output) can be seen in the PINT_PINSTATE register. While neither input nor output drivers of the pin are enabled, reads of the pin state in PINT_PINSTATE return zero. The PINT_PINSTATE register reports the inverted state of the pin if the signal inverter is activated by the PINT_INV_SET register. The inverter can be enabled on a individual bit by bit basis. Every bit in the PINT_INV_SET and PINT_INV_CLR register pair represents a pin signal.

The pin interrupt pin state registers enable the service routine to read the current state of the pin without reading from GPIO space. If there was an edge-sensitive interrupt, the service routine can check whether the state of the pin is still high or turned low.

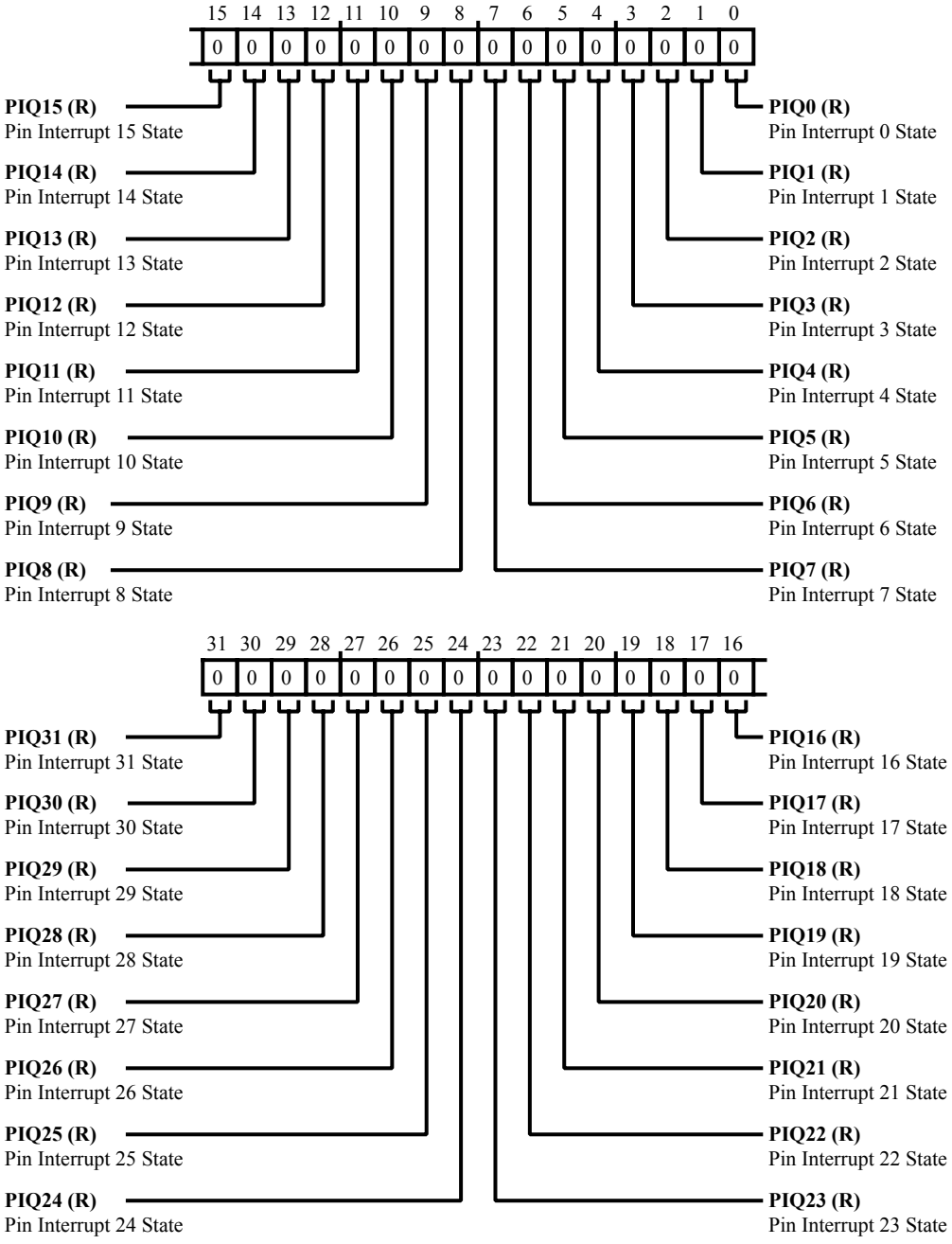

Figure 13-33: PINT_PINSTATE Register Diagram

Table 13-35: PINT_PINSTATE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/NW)	PIQ31	Pin Interrupt 31 State. A read of the `PINT_PINSTATE.PIQ31` bit returns the pin state.

Table 13-35: PINT_PINSTATE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/NW)	PIQ30	Pin Interrupt 30 State. A read of the PINT_PINSTATE.PIQ30 bit returns the pin state.
29 (R/NW)	PIQ29	Pin Interrupt 29 State. A read of the PINT_PINSTATE.PIQ29 bit returns the pin state.
28 (R/NW)	PIQ28	Pin Interrupt 28 State. A read of the PINT_PINSTATE.PIQ28 bit returns the pin state.
27 (R/NW)	PIQ27	Pin Interrupt 27 State. A read of the PINT_PINSTATE.PIQ27 bit returns the pin state.
26 (R/NW)	PIQ26	Pin Interrupt 26 State. A read of the PINT_PINSTATE.PIQ26 bit returns the pin state.
25 (R/NW)	PIQ25	Pin Interrupt 25 State. A read of the PINT_PINSTATE.PIQ25 bit returns the pin state.
24 (R/NW)	PIQ24	Pin Interrupt 24 State. A read of the PINT_PINSTATE.PIQ24 bit returns the pin state.
23 (R/NW)	PIQ23	Pin Interrupt 23 State. A read of the PINT_PINSTATE.PIQ23 bit returns the pin state.
22 (R/NW)	PIQ22	Pin Interrupt 22 State. A read of the PINT_PINSTATE.PIQ22 bit returns the pin state.
21 (R/NW)	PIQ21	Pin Interrupt 21 State. A read of the PINT_PINSTATE.PIQ21 bit returns the pin state.
20 (R/NW)	PIQ20	Pin Interrupt 20 State. A read of the PINT_PINSTATE.PIQ20 bit returns the pin state.
19 (R/NW)	PIQ19	Pin Interrupt 19 State. A read of the PINT_PINSTATE.PIQ19 bit returns the pin state.
18 (R/NW)	PIQ18	Pin Interrupt 18 State. A read of the PINT_PINSTATE.PIQ18 bit returns the pin state.
17 (R/NW)	PIQ17	Pin Interrupt 17 State. A read of the PINT_PINSTATE.PIQ17 bit returns the pin state.
16 (R/NW)	PIQ16	Pin Interrupt 16 State. A read of the PINT_PINSTATE.PIQ16 bit returns the pin state.
15 (R/NW)	PIQ15	Pin Interrupt 15 State. A read of the PINT_PINSTATE.PIQ15 bit returns the pin state.

Table 13-35: PINT_PINSTATE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/NW)	PIQ14	Pin Interrupt 14 State. A read of the PINT_PINSTATE.PIQ14 bit returns the pin state.
13 (R/NW)	PIQ13	Pin Interrupt 13 State. A read of the PINT_PINSTATE.PIQ13 bit returns the pin state.
12 (R/NW)	PIQ12	Pin Interrupt 12 State. A read of the PINT_PINSTATE.PIQ12 bit returns the pin state.
11 (R/NW)	PIQ11	Pin Interrupt 11 State. A read of the PINT_PINSTATE.PIQ11 bit returns the pin state.
10 (R/NW)	PIQ10	Pin Interrupt 10 State. A read of the PINT_PINSTATE.PIQ10 bit returns the pin state.
9 (R/NW)	PIQ9	Pin Interrupt 9 State. A read of the PINT_PINSTATE.PIQ9 bit returns the pin state.
8 (R/NW)	PIQ8	Pin Interrupt 8 State. A read of the PINT_PINSTATE.PIQ8 bit returns the pin state.
7 (R/NW)	PIQ7	Pin Interrupt 7 State. A read of the PINT_PINSTATE.PIQ7 bit returns the pin state.
6 (R/NW)	PIQ6	Pin Interrupt 6 State. A read of the PINT_PINSTATE.PIQ6 bit returns the pin state.
5 (R/NW)	PIQ5	Pin Interrupt 5 State. A read of the PINT_PINSTATE.PIQ5 bit returns the pin state.
4 (R/NW)	PIQ4	Pin Interrupt 4 State. A read of the PINT_PINSTATE.PIQ4 bit returns the pin state.
3 (R/NW)	PIQ3	Pin Interrupt 3 State. A read of the PINT_PINSTATE.PIQ3 bit returns the pin state.
2 (R/NW)	PIQ2	Pin Interrupt 2 State. A read of the PINT_PINSTATE.PIQ2 bit returns the pin state.
1 (R/NW)	PIQ1	Pin Interrupt 1 State. A read of the PINT_PINSTATE.PIQ1 bit returns the pin state.
0 (R/NW)	PIQ0	Pin Interrupt 0 State. A read of the PINT_PINSTATE.PIQ0 bit returns the pin state.

Pint Request Register

The PINT_REQ register indicates interrupt request status for pin interrupts. When set, an interrupt request is pending. When cleared, there is no interrupt request pending.

Both the PINT_REQ and PINT_LATCH registers indicate whether an interrupt request is latched on the respective pin. The PINT_LATCH register is a latch that operates regardless of the interrupt masks. Bits of the PINT_REQ register depend on the mask register. The PINT_REQ register is a logical AND of the PINT_LATCH register and the interrupt mask.

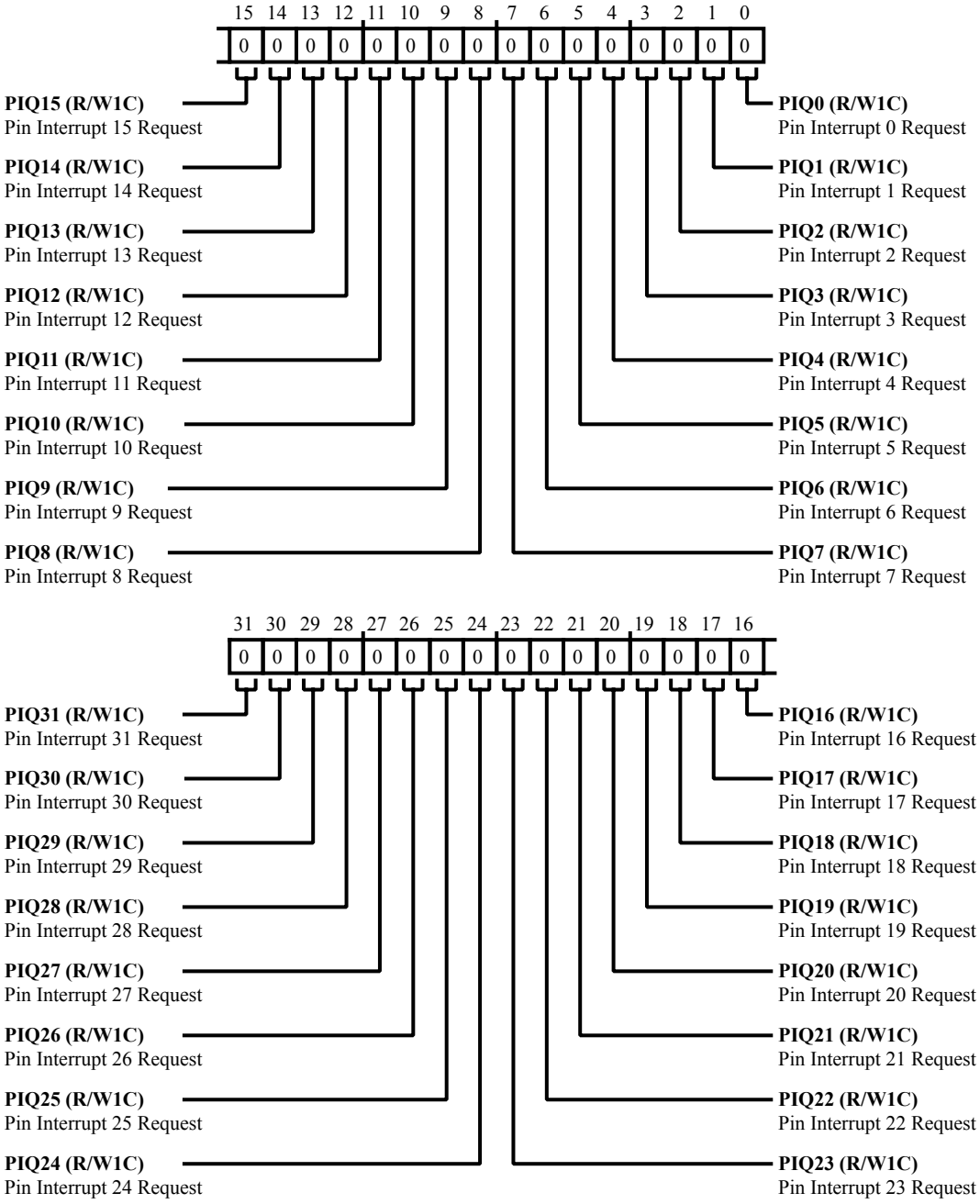

Figure 13-34: PINT_REQ Register Diagram

Table 13-36: PINT_REQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Request. If the `PINT_REQ.PIQ31` bit is set, a request is pending.

Table 13-36: PINT_REQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/W1C)	PIQ30	Pin Interrupt 30 Request. If the PINT_REQ.PIQ30 bit is set, a request is pending.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Request. If the PINT_REQ.PIQ29 bit is set, a request is pending.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Request. If the PINT_REQ.PIQ28 bit is set, a request is pending.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Request. If the PINT_REQ.PIQ27 bit is set, a request is pending.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Request. If the PINT_REQ.PIQ26 bit is set, a request is pending.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Request. If the PINT_REQ.PIQ25 bit is set, a request is pending.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Request. If the PINT_REQ.PIQ24 bit is set, a request is pending.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Request. If the PINT_REQ.PIQ23 bit is set, a request is pending.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Request. If the PINT_REQ.PIQ22 bit is set, a request is pending.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Request. If the PINT_REQ.PIQ21 bit is set, a request is pending.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Request. If the PINT_REQ.PIQ20 bit is set, a request is pending.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Request. If the PINT_REQ.PIQ19 bit is set, a request is pending.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Request. If the PINT_REQ.PIQ18 bit is set, a request is pending.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Request. If the PINT_REQ.PIQ17 bit is set, a request is pending.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Request. If the PINT_REQ.PIQ16 bit is set, a request is pending.
15 (R/W1C)	PIQ15	Pin Interrupt 15 Request. If the PINT_REQ.PIQ15 bit is set, a request is pending.

Table 13-36: PINT_REQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W1C)	PIQ14	Pin Interrupt 14 Request. If the PINT_REQ.PIQ14 bit is set, a request is pending.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Request. If the PINT_REQ.PIQ13 bit is set, a request is pending.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Request. If the PINT_REQ.PIQ12 bit is set, a request is pending.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Request. If the PINT_REQ.PIQ11 bit is set, a request is pending.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Request. If the PINT_REQ.PIQ10 bit is set, a request is pending.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Request. If the PINT_REQ.PIQ9 bit is set, a request is pending.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Request. If the PINT_REQ.PIQ8 bit is set, a request is pending.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Request. If the PINT_REQ.PIQ7 bit is set, a request is pending.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Request. If the PINT_REQ.PIQ6 bit is set, a request is pending.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Request. If the PINT_REQ.PIQ5 bit is set, a request is pending.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Request. If the PINT_REQ.PIQ4 bit is set, a request is pending.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Request. If the PINT_REQ.PIQ3 bit is set, a request is pending.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Request. If the PINT_REQ.PIQ2 bit is set, a request is pending.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Request. If the PINT_REQ.PIQ1 bit is set, a request is pending.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Request. If the PINT_REQ.PIQ0 bit is set, a request is pending.

ADSP-CM40x PADS Register Descriptions

Pads Controller (PADS) contains the following registers.

Table 13-37: ADSP-CM40x PADS Register List

Name	Description
PADS_PCFG0	Peripheral Configuration0 Register

Peripheral Configuration0 Register

The PADS_PCFG0 register provides several configuration options for various peripherals.

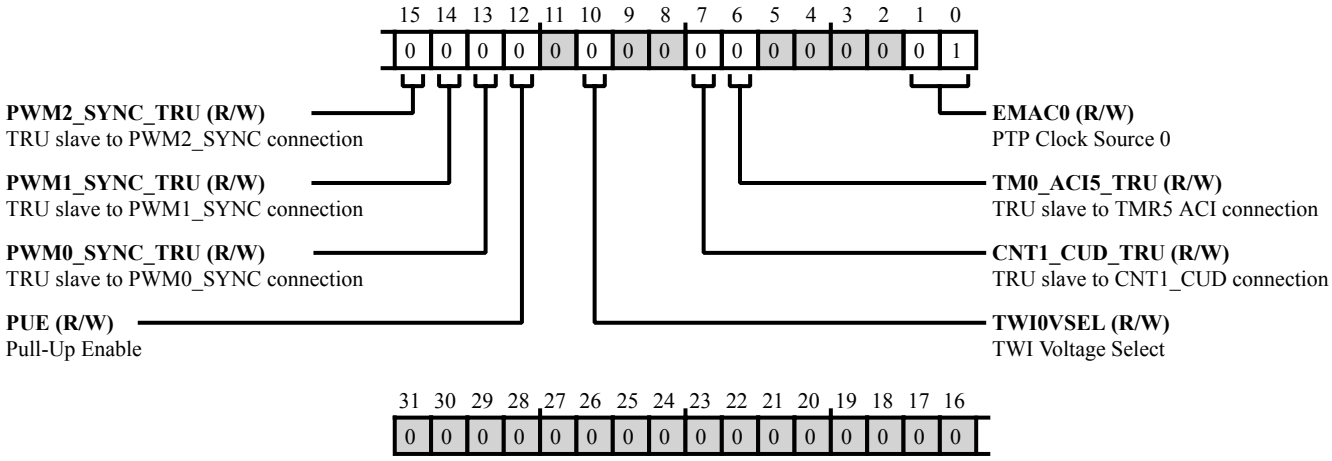

Figure 13-35: PADS_PCFG0 Register Diagram

Table 13-38: PADS_PCFG0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PWM2_SYNC_TRU	TRU slave to PWM2_SYNC connection.	
		0	default GPIO connection to PWM2_SYNC
		1	tru slave 56 connected to PWM2_SYNC
14 (R/W)	PWM1_SYNC_TRU	TRU slave to PWM1_SYNC connection.	
		0	default GPIO connection to PWM1_SYNC
		1	tru slave 53 connected to PWM1_SYNC
13 (R/W)	PWM0_SYNC_TRU	TRU slave to PWM0_SYNC connection.	
		0	default GPIO connection to PWM0_SYNC
		1	tru slave 50 connected to PWM0_SYNC
12 (R/W)	PUE	Pull-Up Enable. The PADS_PCFG0.PUE bit overrides the input enable and enables the pull-up on the AFE pads.	
10 (R/W)	TWI0VSEL	TWI Voltage Select. The PADS_PCFG0.TWI0VSEL bit selects the drive/tolerate voltage for the FIXME pins.	

Table 13-38: PADS_PCFG0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	CNT1_CUD_TRU	TRU slave to CNT1_CUD connection.	
		0	default gpio connection to CNT1 CUD
		1	tru slave 8 connected to CNT1_CUD
6 (R/W)	TM0_ACI5_TRU	TRU slave to TMR5 ACI connection.	
		0	default pinmux connection to TMR5 ACI
		1	tru slave7 connected to TMR5 ACI
1:0 (R/W)	EMAC0	PTP Clock Source 0. The `PADS_PCFG0.EMAC0` selects the clock source for the PTP Block in EMAC0.	
		0	EMAC0_RMII CLK
		1	SYSCLK
		2	External Clock
		3	SYSCLK

14 General-Purpose Timer (TIMER)

The general-purpose timer (GP Timer) module serves as a collection of system timers that support various system-level functions. These functions include:

- Synchronized PWM waveform output capability
- External signal capture
- External event count
- General time-base functionality

Additionally, various interrupts can be generated upon completion of timer events. Moreover, GP timers can act both as trigger masters and trigger slaves.

GP Timer Features

Each timer can be individually configured in any of these modes:

- Pin interrupt capture mode
- Windowed watchdog mode
- Pulse-width count and capture (WDTH_CAP) mode
- External Event (EXT_CLK) mode
- Pulse-width modulation (PWM_OUT) mode

Other features include:

- Synchronous operation
- Consistent management of period and pulse width values
- Autobaud detection for UART module (where available)
- Graceful bit pattern termination when stopping
- Support for center-aligned PWM patterns
- Error detection on implausible pattern values

- All read and write accesses to 32-bit registers are atomic
- Every timer has its dedicated interrupt request output

NOTE: Unused timers can function as edge-sensitive pin interrupts. Each timer has a `TIMER_TMR[n]_CFG.EMURUN` bit which controls whether to run or stop the timer during emulation.

ADSP-CM40x TIMER Register List

The General-Purpose Timer block (TIMER) provides timers that may be used for external event capture and measurement, system timing, and PWM waveform generation. A set of registers governs TIMER operations. For more information on TIMER functionality, see the TIMER register descriptions.

Table 14-1: ADSP-CM40x TIMER Register List

Name	Description
TIMER_BCAST_DLY	Broadcast Delay Register
TIMER_BCAST_PER	Broadcast Period Register
TIMER_BCAST_WID	Broadcast Width Register
TIMER_DATA_ILAT	Data Interrupt Latch Register
TIMER_DATA_IMSK	Data Interrupt Mask Register
TIMER_ERR_TYPE	Error Type Status Register
TIMER_RUN	Run Register
TIMER_RUN_CLR	Run Clear Register
TIMER_RUN_SET	Run Set Register
TIMER_STAT_ILAT	Status Interrupt Latch Register
TIMER_STAT_IMSK	Status Interrupt Mask Register
TIMER_STOP_CFG	Stop Configuration Register
TIMER_STOP_CFG_CLR	Stop Configuration Clear Register
TIMER_STOP_CFG_SET	Stop Configuration Set Register
TIMER_TMR[n]_CFG	Timer n Configuration Register
TIMER_TMR[n]_CNT	Timer n Counter Register
TIMER_TMR[n]_DLY	Timer n Delay Register
TIMER_TMR[n]_PER	Timer n Period Register
TIMER_TMR[n]_WID	Timer n Width Register
TIMER_TRG_IE	Trigger Slave Enable Register
TIMER_TRG_MSK	Trigger Master Mask Register

ADSP-CM40x TIMER Interrupt List

Table 14-2: ADSP-CM40x TIMER Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
50	TIMER0_STAT	TIMER0 Status	Level	
51	TIMER0_TMR0	TIMER0 Timer 0 Expiration or Event	Level	
52	TIMER0_TMR1	TIMER0 Timer 1 Expiration or Event	Level	
53	TIMER0_TMR2	TIMER0 Timer 2 Expiration or Event	Level	
54	TIMER0_TMR3	TIMER0 Timer 3 Expiration or Event	Level	
55	TIMER0_TMR4	TIMER0 Timer 4 Expiration or Event	Level	
56	TIMER0_TMR5	TIMER0 Timer 5 Expiration or Event	Level	
57	TIMER0_TMR6	TIMER0 Timer 6 Expiration or Event	Level	
58	TIMER0_TMR7	TIMER0 Timer 7 Expiration or Event	Level	

ADSP-CM40x TIMER Trigger List

Table 14-3: ADSP-CM40x TIMER Trigger List Masters

Trigger ID	Name	Description	Sensitivity
2	TIMER0_TMR0_MST	TIMER0 Timer 0 Expiration or Event	Edge
3	TIMER0_TMR1_MST	TIMER0 Timer 1 Expiration or Event	Edge
4	TIMER0_TMR2_MST	TIMER0 Timer 2 Expiration or Event	Edge
5	TIMER0_TMR3_MST	TIMER0 Timer 3 Expiration or Event	Edge
6	TIMER0_TMR4_MST	TIMER0 Timer 4 Expiration or Event	Edge
7	TIMER0_TMR5_MST	TIMER0 Timer 5 Expiration or Event	Edge
8	TIMER0_TMR6_MST	TIMER0 Timer 6 Expiration or Event	Edge
9	TIMER0_TMR7_MST	TIMER0 Timer 7 Expiration or Event	Edge

Table 14-4: ADSP-CM40x TIMER Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
2	TIMER0_TMR0_SLV0	TIMER0 Timer 0 Slave	None
3	TIMER0_TMR1_SLV0	TIMER0 Timer 1 Slave	None
4	TIMER0_TMR2_SLV0	TIMER0 Timer 2 Slave	None
5	TIMER0_TMR3_SLV0	TIMER0 Timer 3 Slave	None
6	TIMER0_TMR4_SLV0	TIMER0 Timer 4 Slave	None

Table 14-4: ADSP-CM40x TIMER Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
7	TIMER0_TMR5_SLV0	TIMER0 Timer 5 Slave (default configuration) or Timer0 Alternate Capture Input	None
8	TIMER0_TMR6_SLV0	TIMER0 Timer 6 Slave (default configuration) or CNT1_UD Input	None
9	TIMER0_TMR7_SLV0	TIMER0 Timer 7 Slave	None

Internal Interface

The processor core always accesses the timer registers through the MMR access bus. Hardware ensures that all read and write operations from and to 32-bit timer registers are atomic. Every timer has its dedicated data interrupt request. There is also one common timer status and error interrupt request output that connects to the system event controller. Whenever a data interrupt is generated, a data trigger master pulse is also driven out, if enabled. Each timer has an individual trigger input line, and each timer can be either started or stopped as a trigger slave.

In total, the GP timer module can have up to (N + 1) interrupt output lines and N data trigger lines.

External Interface

Each GP timer module can support up to 16 individual timers. However, most processors have less than this number. The exact number of timers available on a given processor is available in the data sheet for the processor.

Every timer has one main input/output signal (TIMER_TMR[n]) and, usually, one auxiliary input pin, used as an alternate capture input (TIMER_ACI[n]). Each timer can either run with a time base of SYSCLK or can reference an external clock on one of two TIMER_ACLK[n] pins. The TMR_ALT_CLK0 signal maps to individual alternate clock (TIMER_ACLK[n]) pins for one or more timers. For instance, a TM_ACLK3 pin would provide an alternate site to supply an external signal that would serve as reference clock for TMR3. Likewise, the TMR_ALT_CLK1 signal from each timer unit connects together internally to provide a single global timer clock pin (TIMER_CLK) for the GP timer module. It is used as an additional time base.

In the ***Timer Alternate Input and Alternate Clock*** table, column 4 is the alternative capture input list and columns 6 and 7 are alternative clock input0 and alternative clock input1, respectively.

Table 14-5: Timer Alternate Input and Alternate Clock

Timer Group	Timer	TMn_TMRm	TMn_ACIm	Peripheral connection	TMn_ACLKm	TMn_ACLKm
			Alternate Capture Input	on same GPIO:	Auxiliary Clock 0	Auxiliary Clock 1
0	0	PA_12	PA_04	UART0_RXb	PA_09	PA_00(TM0_CLK)
0	1	PA_13	PA_06	CAN0_RX	PA_10	PA_00
0	2	unused(0)	PA_01		PA_07	PA_00
0	3	unused(0)	PA_02		PA_08	PA_00

Table 14-5: Timer Alternate Input and Alternate Clock (Continued)

Timer Group	Timer	TMn_TMRm	TMn_ACIm	Peripheral connection	TMn_ACLKm	TMn_ACLKm
			Alternate Capture Input	on same GPIO:	Auxiliary Clock 0	Auxiliary Clock 1
0	4	unused(0)	PA_03		PA_05	PA_00
0	5	unused(0)	M0 Trigger SlaveTIMER0_ACI5		AuxCLk	PA_00
0	6	AuxClk	M0 Trigger SlaveTIMER0_ACI6		SYS_CLKIN0	PA_00
0	7	unused(0)	M0 Trigger Slave TIMER0_ACI7		SYS_CLKIN1	PA_00
1	0	PE_14, PB_14	PE_12	CAN1_RX	SYS_CLKIN0	PC_06(TM1_CLK)
1	1	PE_15, PB_15	M4 Trigger Slave TIMER1_ACI1		SYS_CLKIN0	PC_06
1	2	PB_13	PC_08	UART3_RXb	AuxClk	PC_06
1	3	PC_10	CNT_TO		PC_15	PC_06
1	4	PE_04	PC_09	UART2_RXb	PC_00	PC_06
1	5	PF_06	PE_09	UART1_RXb	PE_08	PC_06
1	6	PE_02	PF_05	UART4_RXb	SYS_CLKIN0	PC_06
1	7	PC_12	TIMER1_ACI7 M4 Trigger Slave		SYS_CLKIN1	PC_06

GP Timer Operating Modes

The following sections provide information on the various operating modes of the GP timer.

General Operation

The core of every timer is a 32-bit counter that can be interrogated through the read-only `TIMER_TMR[n]_CNT` register. Once the module enables a timer, it loads the timer `TIMER_TMR[n]_CNT` register with a starting value.

A timer can operate in one of several different modes, configured through the `TIMER_TMR[n]_CFG` register for that timer. These modes are: PWMOUT, EXTCLK, WIDCAP, WATCHDOG, PININT, and IDLE. The ***Timer Mode Descriptions*** table summarizes the modes.

Table 14-6: Timer Mode Descriptions

Timer Mode	Description
PWMOUT	Generates single or continuous PWM waveforms with programmable pulse width, period, and delay
EXTCLK	Counts edges of an externally applied waveform
WIDCAP	Captures pulse width or period of an externally applied waveform
WATCHDOG	Monitors pulse width or period of an external signal and compares against a window of acceptable values, optionally generating an interrupt when it falls inside or outside of that window
PININT	Can generate an interrupt on an active edge applied to a timer pin
IDLE	Idle; no activity

Period, Width and Delay Register Interaction

When the timer is started, writes to the buffer registers are immediately copied through to the double-buffered period, pulse width, and delay registers. These values are then ready for use in the first timer period. When a timer is already running, software can write new values to the `TIMER_TMR[n]_PER`, `TIMER_TMR[n]_WID`, and `TIMER_TMR[n]_DLY` registers. The written values are buffered and do not update into the registers until the end of the current period. (The update occurs when the value in the `TIMER_TMR[n]_CNT` register equals the value in the `TIMER_TMR[n]_PER` register).

If new values are not written to these registers, the value from the previous period is reused. Writes to these registers are atomic; it is not possible for the high word to be written without the low word also being written. Values written to the period, pulse width, and delay registers are always stored in the buffer registers. Reads from the same register always return the current, active value of period, pulse width, or delay value. Written values are not readback until they become active.

The usage of the `TIMER_TMR[n]_PER`, `TIMER_TMR[n]_WID`, and `TIMER_TMR[n]_DLY` registers varies, depending on the mode of the timer specified by the `TIMER_TMR[n]_CFG.TMODE` bits. See the ***Usage of the Period, Width, and Delay Registers in Different Timer Modes*** table for more information.

Table 14-7: Usage of the Period, Width, and Delay Registers in Different Timer Modes

Timer Mode	Period	Width	DELAY
IDLE	Not writable	Not writable	Not writable
WATCHDOG	Can be updated on-the-fly. New value takes effect either upon timer start or when an asserting edge on the input signal is sensed.	Read-only. Retains value of last measured width or period of the input signal.	Can be updated on-the-fly. New value takes effect either upon timer start or when an asserting edge on the input signal is sensed.
WIDCAP	Read-only. Period value captured at the appropriate time and updated from its buffer register simultaneously with the Width register.	Read-only. Width value captured at the appropriate time and updated from its buffer register simultaneously with the Period register.	Not used

Table 14-7: Usage of the Period, Width, and Delay Registers in Different Timer Modes (Continued)

Timer Mode	Period	Width	DELAY
PWMOUT	Can be updated on-the-fly. New value takes effect either upon timer start or at the end of the current period. A write followed by immediate read returns the current operating values.	Can be updated on-the-fly. New value takes effect either upon timer start or at the end of the current period. A write followed by immediate read returns the current operating values.	Can be updated on-the-fly. New value takes effect either upon timer start or at the end of the current period. A write followed by immediate read returns the current operating values.
EXTCLK	Can be updated on-the-fly.	Not used	Not used
PININT	Not used	Not used	Not used

NOTE: If any of the period, pulse width, and delay registers are not used, then programs cannot write into that register. For example, in WIDCAP mode, the delay registers are not used. So, the program is not allowed to write any value to the TIMER_TMR[n]_DLY register. To prevent undesired operation, program the TIMER_TMR[n]_CFG.TMODE bits before programming the period, width, or delay registers.If a program changes the TIMER_TMR[n]_CFG.TMODE bits from a status register to writable register (for example in PWMOUT mode), hardware does not clear these registers. These values are automatically overwritten by new values specified by software.In PWMOUT mode with small periods, there may not be enough time between updates from the buffer registers to write these registers. The next period can use one old value and one new value. To prevent (width + pulse delay) > period errors, write the width and delay registers before the period register when decreasing the values. Write the period register before the width and delay registers when increasing the value.

Single-Pulse PWMOUT Mode

In single-pulse PWMOUT mode, the timer generates a single pulse on the TIMER_TMR[n] pin. This mode is frequently used to implement a precise delay, often with generating an output trigger. The timer module uses the value in the TIMER_TMR[n]_DLY register to control the assertion of a pulse. The value in the TIMER_TMR[n]_WID register defines the pulse width. The TIMER_TMR[n]_PER is not used and cannot be written in this mode. After completion of the pulse, the timer is automatically stopped, and optionally generates an interrupt. The timer uses the TIMER_TMR[n]_CFG.PULSEHI bit to control pulse polarity.

The timer can be configured to generate a data interrupt after satisfying various conditions specified by the TIMER_TMR[n]_CFG.IRQMODE bits.

It is not necessary to clear the relevant TIMER_RUN bit to stop the timer cleanly. At the end of the pulse, the timer stops automatically and the corresponding TIMER_RUN bit is cleared. To generate multiple discrete pulses (as opposed to a continuous PWM waveform), write a 1 to the appropriate TIMER_RUN bit, and wait for the timer to stop. Then, write another 1 to the same TIMER_RUN bit.

Continuous PWMOUT Mode

In continuous PWMOUT mode, the timer generates a repetitive pulse with a well-defined period, duty cycle, and pulse position. The TIMER_TMR[n]_DLY, TIMER_TMR[n]_PER, and TIMER_TMR[n]_WID registers are

programmed with the values of the required PWM pulse. After the timer is started, the counter counts towards the value programmed in the TIMER_TMR[n]_PER register. Initially, the TIMER_TMR[n] pin remains in a deasserted state. The pin toggles to an asserted state when the value in the TIMER_TMR[n]_CNT register equals the value in the TIMER_TMR[n]_DLY register.

The timer can control the assertion sense of the TIMER_TMR[n] pin with the TIMER_TMR[n]_CFG.PULSEHI bit. The TIMER_TMR[n] pin holds this value for the number of clock cycles specified in the TIMER_TMR[n]_WID register. Then, the pin deasserts and holds this value until the completion of the programmed period. The same waveform is generated repeatedly until the timer is disabled.

The timer can be configured to generate a data interrupt after satisfying any of various conditions specified by the TIMER_TMR[n]_CFG.IRQMODE bits.

It is important to guarantee that the programmed period is greater than or equal to the sum of width and delay. Similarly, delay must be less than period. Violating either of these criteria results in an unpredictable waveform on the TIMER_TMR[n] pin until the situation is rectified by writing proper values to these registers.

The maximum frequency possible to generate on the TIMER_TMR[n] pin is achieved by setting TIMER_TMR[n]_PER to 2 and TIMER_TMR[n]_WID to 1. This operation makes the TIMER_TMR[n] pin toggle each SYSCLK clock cycle (assuming the timer is configured to clock internally), producing a duty cycle of 50%.

When the TIMER_STOP_CFG.TMR[nn] bit of a timer is 0, the timer treats a stop operation as a stop-is-pending condition. When terminated with this setting, the timer automatically completes the current waveform and then stops cleanly, remaining in a deasserted state. This funtionality prevents truncation of the current pulse and unwanted PWM patterns at the TIMER_TMR[n] pin. The processor can determine when the timer stops running by polling the corresponding TIMER_RUN.TMR[nn] bit until it reads 0 or by waiting for the last interrupt (if enabled).

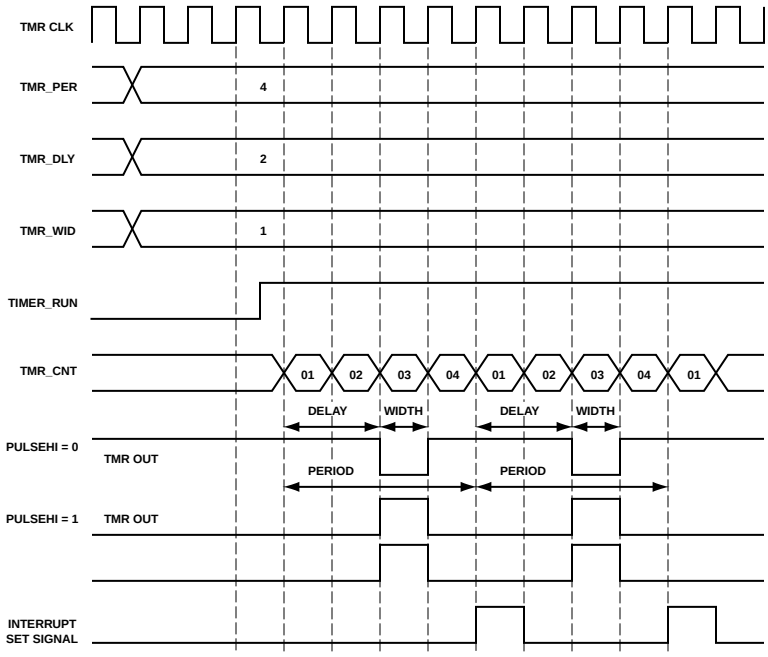

Figure 14-1: Signal Generation in Continuous PWMOUT Mode

The TIMER_TMR[n]_CFG register cannot be reconfigured until after the timer stops and the TIMER_RUN register reads 0.

Programs can force a timer to stop immediately in PWMOUT mode by writing a 1 to the TIMER_STOP_CFG register followed by writing a 1 to the TIMER_RUN_CLR register. (Or, a program can stop a timer by writing a 0 to the appropriate TIMER_RUN.TMR[nn] bit). This operation stops the timer whether the pending stop is waiting for the end of the current period or the end of the current pulse width. The timer can use this feature to regain immediate control of a timer during an error recovery sequence.

Use this feature carefully, as it can corrupt the PWM pattern generated at the TIMER_TMR[n] pin, though after such a stop the pin deasserts automatically. Each timer samples its TIMER_RUN.TMR[nn] bit at the end of each period. It stops cleanly at the end of the first period after the TIMER_RUN.TMR[nn] bit is low. A timer that is disabled and then restarted (before the end of the current period), continues to run as if nothing happened. Typically, the program disables a PWMOUT timer and then waits for it to stop itself.

Width Capture (WIDCAP) Mode

The timer uses WIDCAP mode, often called capture mode, to measure pulse widths on the TIMER_TMR[n] or TIMER_ACI[n] inputs. The polarity (active high or low) of the input signal can be selected with the TIMER_TMR[n]_CFG.PULSEHI bit. The ***Timer Signal Flow in Width Capture Mode*** figure shows the control signal flow for WIDCAP_CAP mode.

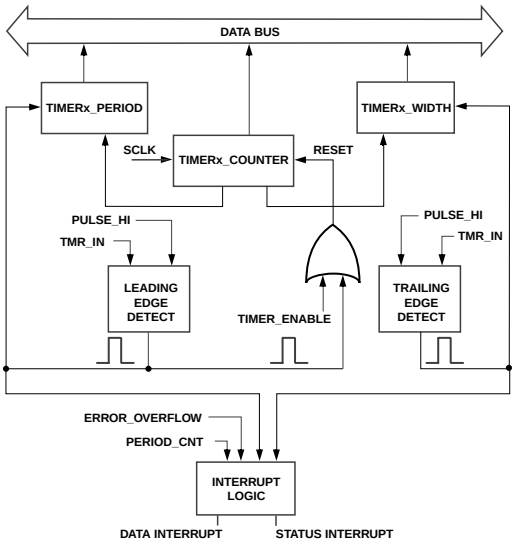

Figure 14-2: Timer Signal Flow in Width Capture Mode

In this mode, the timer uses the TIMER_TMR[n]_CFG.TINSEL bit to select between the TIMER_TMR[n] or TIMER_ACI[n] input. The internally clocked timer is used to determine the period and pulse width of the externally applied rectangular waveforms.

NOTE: In WIDCAP_CAP mode, when TMR_AUX_IN input is selected, a number of the timers sense internal signals from the GP counter-module through their alternate input. (These internal signals appear as “TO

GP TIMER TMR_AUX_IN (IF ENABLED)" in the GP Counter Block Diagram in the Counter chapter). This feature lets the timer capture the period between counter-events and supports autobaud detection for the UARTs of the processor.

Table 14-8: ADSP-CM40x WIDCAP_CAP Mode Alternate Inputs from Counter

Timer Alternate Input	Counter-Timer Output
TMR6_ACI	CNT0_TO
TMR7_ACI	CNT1_TO
TACI5	UART0 Rx
TACI2	UART1 Rx
TACI3	UART2 Rx

When a timer is enabled in this mode, the timer resets the count in its `TIMER_TMR[n]_CNT` register to 0x0000 0001. It does not start counting until it detects a leading edge on the selected input pin.

When the timer detects the first leading edge, it starts incrementing. When it detects a trailing edge of a waveform, it captures the current 32-bit value of its `TIMER_TMR[n]_CNT` register into its width buffer register. At the next leading edge, the timer transfers the current 32-bit value of its `TIMER_TMR[n]_CNT` register into its period buffer register. The `TIMER_TMR[n]_CNT` register is reset to 0x0000 0001 again, and the timer continues counting and capturing until it is disabled.

In this mode, programs can measure both the pulse width and the pulse period of a waveform. The timer does not use the `TIMER_TMR[n]_DLY` register in this mode. The timer uses the `TIMER_TMR[n]_CFG.PULSEHI` bit to control the definition of leading edge and trailing edge of the `TIMER_TMR[n]/TIMER_ACI[n]` pin.

In WIDCAP mode, the following events always occur at the same time as one unit:

1. The `TIMER_TMR[n]_PER` register is updated from the period buffer register.
2. The `TIMER_TMR[n]_WID` register is updated from the width buffer register.
3. The `TIMER_DATA_ILAT.TMR[nn]` bit is set (if enabled).
4. A timer data trigger pulse is generated (if enabled).

The `TIMER_TMR[n]_CFG.TMODE` bit 0 controls the point in time at which this set of events is executed. Taken together, these four events are called a measurement report. The `TIMER_STAT_ILAT` register is not set at a measurement report. A measurement report occurs, at most, once per input signal period. The current `TIMER_TMR[n]_CNT` value is always copied to the width buffer and period buffer registers at the trailing and leading edges of the input signal, respectively. But, these values are not visible to software. A measurement report event samples the captured values into visible registers and sets the timer interrupt to signal that the `TIMER_TMR[n]_PER` and the `TIMER_TMR[n]_WID` registers are ready to be read.

When the `TIMER_TMR[n]_CFG.TMODE` bit =b#1011, the measurement report occurs just after the width buffer register captures its value at a falling edge. Then, the `TIMER_TMR[n]_WID` register reports the pulse width measured in the pulse that has ended, but the `TIMER_TMR[n]_PER` register reports the pulse period measured at the end

of the previous period. If only the first trailing edge has occurred, then the first period value has not yet been measured at the first measurement report. So, the period value is not valid. A read of the TIMER_TMR[n]_PER value in this case returns 0. See the ***Example of Width Capture Deasserted Mode (TMODE=b#1011)*** figure for more information.

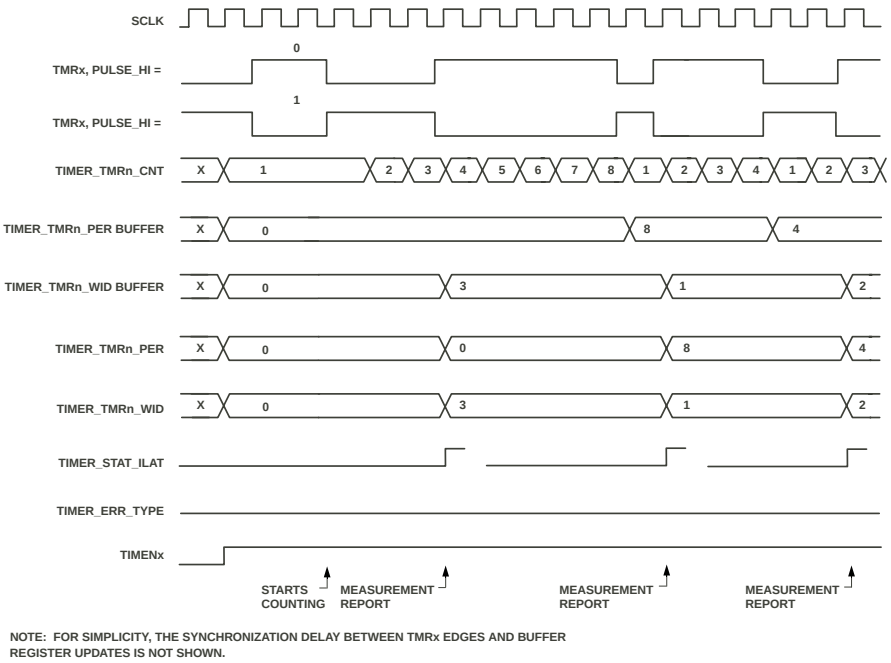

Figure 14-3: Example of Width Capture Deasserted Mode (TMODE=b#1011)

When the TIMER_TMR[n]_CFG.TMODE bit =b#1010, the measurement report occurs just after the period buffer register captures its value at a leading edge. Then, the TIMER_TMR[n]_PER and TIMER_TMR[n]_WID registers report the pulse period and pulse width measured in the period that has ended. Refer to the ***Example of Width Capture Asserted Mode (TMODE=b#1010)*** figure for more information.

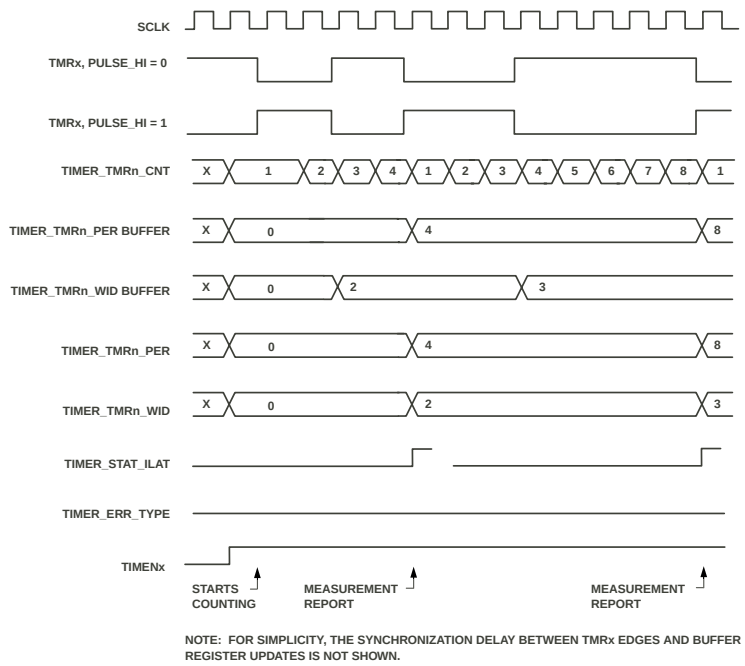

Figure 14-4: Example of Width Capture Asserted Mode (TMODE=b#1010)

To measure the pulse width of a waveform that has only one leading edge and one trailing edge, set `TIMER_TMR[n]_CFG.TMODE` = b#1011. If `TIMER_TMR[n]_CFG.TMODE` = b#1010 for this case, no period value is captured in the period buffer register. Instead, the timer genererates an error report interrupt (if enabled) when the `TIMER_TMR[n]_CNT` range is exceeded and the counter wraps around. In this case, both the `TIMER_TMR[n]_PER` and `TIMER_TMR[n]_WID` registers read 0 (because no measurement report occurred to copy the value captured in the width buffer register to the `TIMER_TMR[n]_WID` register).

If using the `TIMER_TMR[n]_CFG.TMODE` bit =b#1010 mode to measure the width of a single pulse, programs can disable the timer after taking the interrupt that ends the measurement interval. If desired, restart the timer as appropriate in preparation for another measurement. This procedure prevents the timer from free-running after the width measurement and logging errors generated by the timer count overflowing.

Width Capture Mode Overflow

A timer status interrupt (when enabled) is generated when the `TIMER_TMR[n]_CNT` register wraps around from 0xFFFF FFFF to 0 in the absence of a leading edge. At that point, the `TIMER_STAT_ILAT` bit is set and the `TIMER_ERR_TYPE` bits change to indicate a count overflow due to a period greater than the range of the counter. This indication is referred to as an error report. A data interrupt in WIDCAP mode indicates that a new measurement is ready to be read (a measurement report). Similarly, an interrupt on the timer status interrupt line (shared interrupt for all timers) indicates an overflow error when generated in this mode.

The `TIMER_TMR[n]_PER` and `TIMER_TMR[n]_WID` registers are never updated at the time an overflow error is signaled. If the timer overflows and the `TIMER_TMR[n]_CFG.TMODE` bit =b#1010, the `TIMER_TMR[n]_PER` and `TIMER_TMR[n]_WID` registers are not updated. If the timer overflows and the `TIMER_TMR[n]_CFG.TMODE` bit

=b#1011, the TIMER_TMR[n]_PER and TIMER_TMR[n]_WID registers are updated only if a trailing edge is detected at a previous measurement report.

Software can count the number of error reports between measurement report interrupts to measure input signal periods longer than 0xFFFF FFFF. Each error report interrupt adds a full 2^{32}SYSCLK counts to the total for the period, but the width is ambiguous. Ensure that if software monitors only the status interrupt, then status interrupts from all other timers are masked.

Refer to the ***Example Timing for Width Capture Followed by Period Overflow (TMR_CFG.TMODE=b#1010)*** figure. The period is 0x1 0000 0004, but the pulse width could be either 0x0 0000 0002 or 0x1 0000 0002.

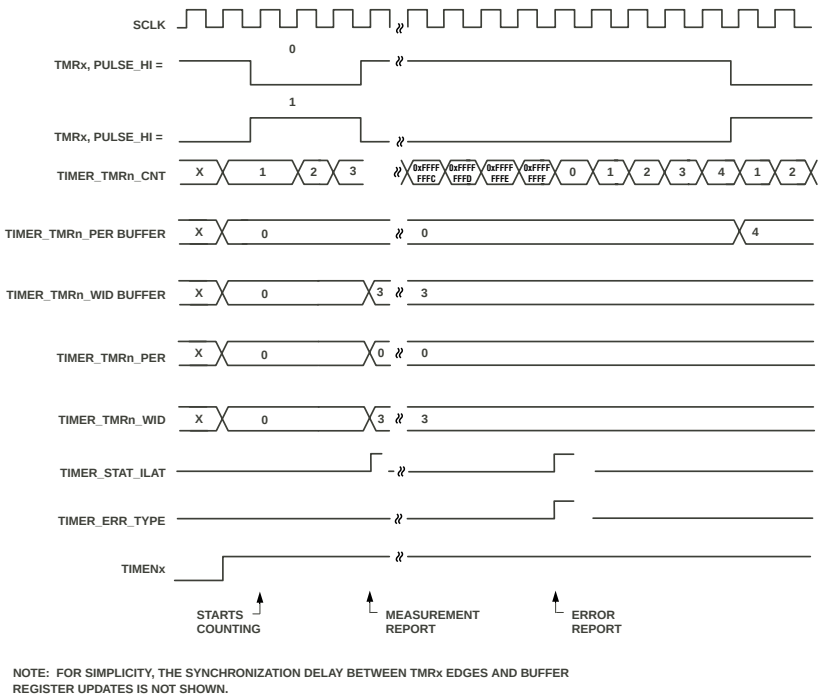

Figure 14-5: Example Timing for Width Capture Followed by Period Overflow (TMR_CFG.TMODE=b#1010)

The waveform applied to the TIMER_TMR[n] (or TIMER_ACI[n]) pin is not required to have a 50% duty cycle. The minimum input low time is little more than one SYSCLK period. The minimum input high time is a little more than one SYSCLK period. (Refer to the product data sheet for details). The maximum TIMER_TMR[n] input frequency is less than SYSCLK/2, with a 50% duty cycle. Under these conditions, the WIDCAP mode timer measures: period =2 and pulse width =1.

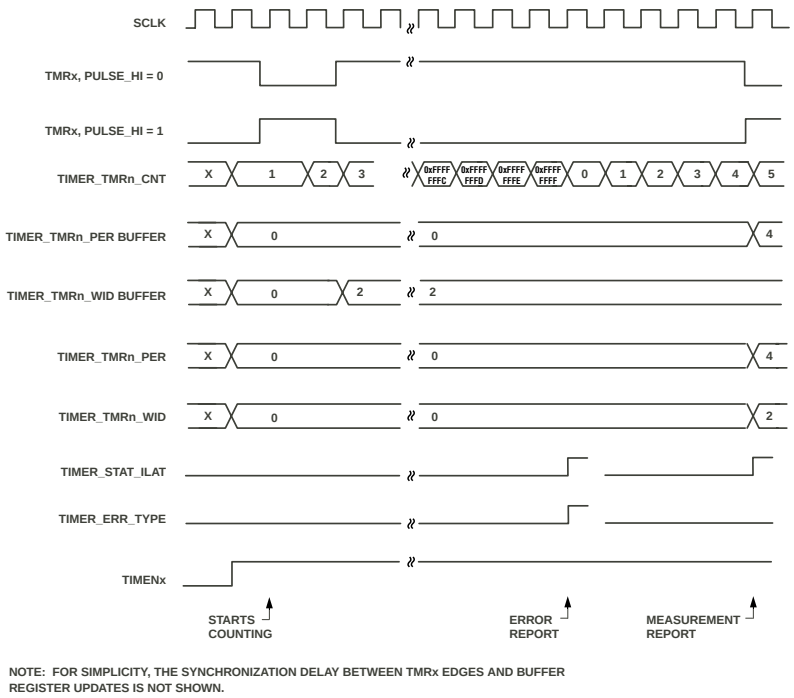

Figure 14-6: Example Timing for Width Capture Followed by Period Overflow (TMR_CFG.TMODE=b#1011)

Windowed Watchdog (WATCHDOG) Modes

In windowed watchdog (WATCHDOG) modes, a timer can take inputs from either the `TIMER_TMR[n]` pin or the `TIMER_ACI[n]` pin. With this mode, the timer can monitor pulse width (width watchdog mode) or pulse period (period watchdog mode) on the input line. It also compares the measured value against a minimum required value and maximum allowed value and generates an interrupt appropriately. The timer uses the `TIMER_TMR[n]_CFG.PULSEHI` bit to select polarity of the input signal.

The waveform applied to the input pin in watchdog mode is not required to have a 50% duty cycle. The minimum input pulse low time, high time, and total period specifications are available in the product data sheet.

Windowed Watchdog Width Mode

In windowed watchdog width mode, the timer counter monitors the pulse width of an input signal on either the `TIMER_TMR[n]` pin or one of the alternate clock pins (`TIMER_ACLK[n]`). Program the minimum pulse width (p_{MIN}) in the `TIMER_TMR[n]_DLY` register and the maximum pulse width (p_{MAX}) in the `TIMER_TMR[n]_PER` register. Both values are programmed in terms of number of clock cycles (SYSCLK or alternate clock). The timer can generate an interrupt if the de-asserting pulse edge occurs

- Inside the window (p_{MIN} < pulse width ≤ p_{MAX}), or
- Outside the window (pulse width ≤ p_{MIN} or pulse width > p_{MAX})

After enabling the timer in this mode, it always starts counting at the asserting edge of the input signal. Any pulse that is already active when the timer is enabled is ignored.

With the TIMER_TMR[n]_CFG.IRQMODE bit =b#11, the timer generates an interrupt if the timed pulse width exceeds p_{MAX}, or if the pulse width is less than p_{MIN}. After attaining p_{MAX}, the pulse stays at an active level, and the counter keeps on counting until it sees a de-asserting edge. When the input pulse is not active, the counter holds its current value until it again sees an asserting edge, or it restarts. An interrupt can also be generated for when the pulse occurs within the specified window condition, by setting TIMER_TMR[n]_CFG.IRQMODE =b#10.

In this mode, a trailing edge on the input pin triggers capturing of pulse width into the TIMER_TMR[n]_WID register. During the inactive portion of the input signal, the internal counter does not increment. The ***Watchdog Width Mode Timing*** figure shows the signal flow in this mode.

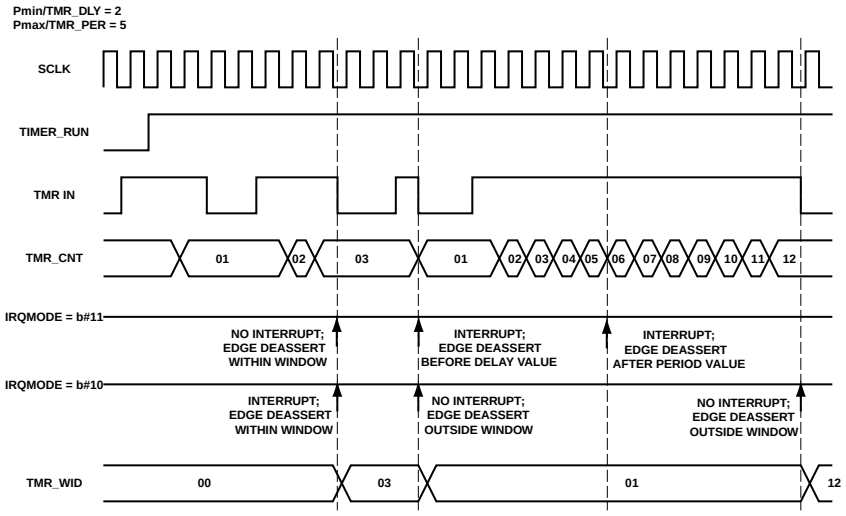

Figure 14-7: Watchdog Width Mode Timing

To check only the upper limit on pulse width (p_{MAX} but not p_{MIN}), then program p_{MIN} as 0 or 1. In such a case, it is better to use TIMER_TMR[n]_CFG.IRQMODE =b#11. With TIMER_TMR[n]_CFG.IRQMODE = b#10, a pulse width of 1 clock cycle results in an interrupt. For details, see the ***Windowed Watchdog Width Mode Interpretation*** table.

Table 14-9: Windowed Watchdog Width Mode Interpretation

Timer Delay	Timer Period	Incoming Pulse Width	IRQMODE= b#10	IRQMODE= b#11	Error Interrupt?
0 or 1	Anything ≥ 1	PW = 1	Interrupt at de-asserting edge of input signal	No Interrupt	No Error Interrupt
		PW ≤ TMR_PER	Interrupt at de-asserting edge of input signal	No Interrupt	No Error Interrupt
		PW > TMR_PER	No Interrupt	Interrupt when pulse width exceeds Pmax (Period Register) Value	No Error Interrupt

Table 14-9: Windowed Watchdog Width Mode Interpretation (Continued)

Timer Delay	Timer Period	Incoming Pulse Width	IRQMODE= b#10	IRQMODE= b#11	Error Interrupt?
> 1 but ≤ (Period -1)	Anything > 1	PW ≤ TMR_DLY	No Interrupt	Interrupt at De-asserting edge of input signal	No Error Interrupt
		TMR_DLY < PW≤ TMR_PER	Interrupt at de-asserting edge of input signal	No Interrupt	No Error Interrupt
		PW > TMR_PER	No Interrupt	Interrupt when pulse width exceeds Pmax (Period Register) Value	No Error Interrupt
≥ Period	-	PW ≤ TMR_PER	Undefined	Undefined	No Error Interrupt
	-	PW > TMR_PER	Undefined	Undefined	b#11 Error Type
-	0	-	Undefined	Undefined	b#10 Error Type

Windowed Watchdog Period Mode

In this mode, the timer monitors the number of clock cycles between two consecutive rising or falling edges of an input signal on either the `TIMER_TMR[n]` or `TIMER_ACI[n]` pin. Program the required minimum number of clock cycles (t_{MIN}) in the `TIMER_TMR[n]_DLY` register and the required maximum allowed number of clock cycles (t_{MAX}) in the `TIMER_TMR[n]_PER` register. Both values are programmed in terms of number of clock cycles (SYSCLK) or alternate time clock (`TIMER_ACLK[n]`). The timer can generate an interrupt when two consecutive occurrences of an active edge are:

- Within a specified window (t_{MIN} < Pulse Period ≤ t_{MAX}), or
- Outside a specified window (pulse width ≤ (t_{MIN} or t_{MAX} < pulse width)

When the `TIMER_TMR[n]_CFG.IRQMODE` bit =b#11 and the pulse period > t_{MAX} or is ≤ t_{MIN}, the timer generates an interrupt (if unmasked). After attaining the t_{MAX} value, the counter keeps on counting until it sees an active edge on the input line. An interrupt can also be generated for when the pulse occurs within the specified window condition, by setting `TIMER_TMR[n]_CFG.IRQMODE` =b#10. Refer to the ***Watchdog Period Mode Timing*** figure for timer functionality in period watchdog mode.

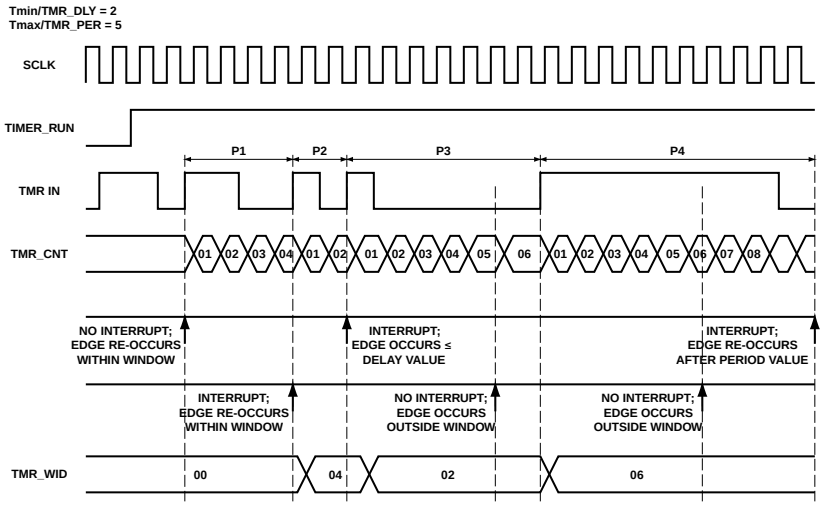

Figure 14-8: Watchdog Period Mode Timing

To check only the upper limit on period (the t_{MAX} value, not the t_{MIN} value), program t_{MIN} as 0 or 1. For details, refer to the ***Windowed Watchdog Period Mode Interpretation*** table.

Table 14-10: Windowed Watchdog Period Mode Interpretation

Timer Delay	Timer Period	Incoming Pulse Width	IRQMODE=b#10	IRQMODE =b#11	Error Interrupt?
0 or 1	Anything ≥ 2	Pulse Period ≤ TMR_PER	Interrupt at de-asserting edge of input signal	No Interrupt	No Error Interrupt
		Pulse Period > TMR_PER	No Interrupt	Interrupt when pulse period crosses Pmax (Period Register) value	No Error Interrupt
> 1 but ≤ Period -1	Anything ≥ 2	Pulse Period ≤ TMR_DLY	No Interrupt	Interrupt at de-asserting edge of input signal	No Error Interrupt
		TMR_DLY < Pulse Period ≤TMR_PER	Interrupt at de-asserting edge of input signal	No Interrupt	No Error Interrupt
		Pulse Period > TMR_PER	No Interrupt	Interrupt when pulse width exceeds Pmax (Period Register) value	No Error Interrupt
≥ Period	-	Pulse Period < TMR_PER	Undefined	Undefined	No Error Interrupt
		Pulse Period ≥ TMR_PER	Undefined	Undefined	b#11 Error Type

Table 14-10: Windowed Watchdog Period Mode Interpretation (Continued)

Timer Delay	Timer Period	Incoming Pulse Width	IRQMODE=b#10	IRQMODE =b#11	Error Interrupt?
-	0 or 1	-	Undefined	Undefined	b#10 Error Type

Pin Interrupt (PININT) Mode

In PININT mode, any active edges on either the TIMER_TMR[n] pin or the TIMER_ACI[n] pin can cause an edge-based interrupt, if enabled. (The timer uses the TIMER_TMR[n]_CFG.TINSEL register to select the pin). The event on the input pin can set the TIMER_DATA_ILAT.TMR[nn] bit and issue a system interrupt request. Program the TIMER_TMR[n]_CFG.PULSEHI bit to change active edge polarity.

Since the interrupt is generated in the SYSCLK clock domain, the width of the input signal must be more than one SYSCLK period. Along with generating the interrupt, the timer also generates a trigger pulse (configured using the TIMER_TRG_MSK register). Due to the configuration of polarity, glitches can cause the generation of an undesired interrupt at the input. To avoid this problem, programs must ensure that interrupts are unmasked only after configuring the desired polarity.

External Clock (EXTCLK) Mode

The timer uses EXTCLK mode, sometimes referred to as the counter mode, to count external events (signal edges), on either the TIMER_TMR[n] or TIMER_ACI[n] input pin. The timer works as a counter clocked by an external source (the signal at the pin), which can be asynchronous to SYSCLK. The current count in the TIMER_TMR[n]_CNT register represents the number of leading-edge events detected. The TIMER_TMR[n]_PER register is programmed with the value of the maximum timer external count before stopping or issuing an interrupt or trigger.

The TIMER_TMR[n]_CFG.PULSEHI bit determines the polarity of the leading edge on the input pin. The timer uses the TIMER_TMR[n]_CFG.TINSEL bit to select whether the event is counted on the TIMER_TMR[n] or on the TIMER_ACI[n] pin. The TIMER_STAT_ILAT.TMR[nn] and TIMER_ERR_TYPE bits are set if:

- TIMER_TMR[n]_CNT wraps around from 0xFFFF FFFF to 0, or
- the period = 0 at startup, or
- TIMER_TMR[n]_CNT register rolls over (from count = period to count = 0x1).

The TIMER_TMR[n]_WID and TIMER_TMR[n]_DLY registers are unused in this mode and must not be written.

The ***EXTCLK Mode Control Flow*** figure shows a flow diagram for EXTCLK mode.

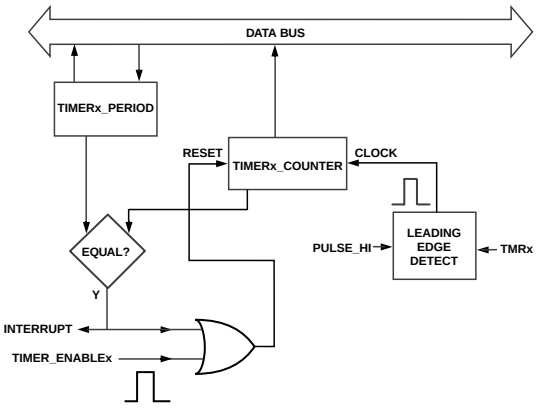

Figure 14-9: EXTCLK Mode Control Flow

The waveform applied to the input pin is not required to have a 50% duty cycle. The minimum input pulse low time, high time, and total period specifications are available in the product data sheet. Program the period to any value from 1 to (2^{32} – 1), inclusive.

After the timer has started, it resets the `TIMER_TMR[n]_CNT` register to 0x0 and then waits for the first leading edge on the input pin. This edge causes `TIMER_TMR[n]_CNT` to be incremented to the value 0x1, and every subsequent leading edge increments it by one. After the `TIMER_TMR[n]_CNT` register reaches the value programmed in the `TIMER_TMR[n]_PER` register, the corresponding `TIMER_DATA_ILAT` bit is set, and an interrupt and trigger are both generated (if enabled). The next leading-edge reloads the `TIMER_TMR[n]_CNT` register with 0x1, and the timer continues counting until it is disabled.

Trigger Slave Mode

The Trigger Slave feature includes a programmable option that counts the trigger pulses for a particular TRU (Trigger Routing Unit) master. This is accomplished by internally routing the trigger pulses to the Alternate Capture Input (`TIMER_ACI5`) signal of Timer 5 and to the `CNT_UD` input signal of the GP counter. Configure the PADS_PCONFIG0[6] and PADS_PCONFIG0[7] bits to enable count of trigger pulses by Timer 5 and Counter 1 respectively. This is shown in the ***Trigger Slave Mode*** figure and explained below.

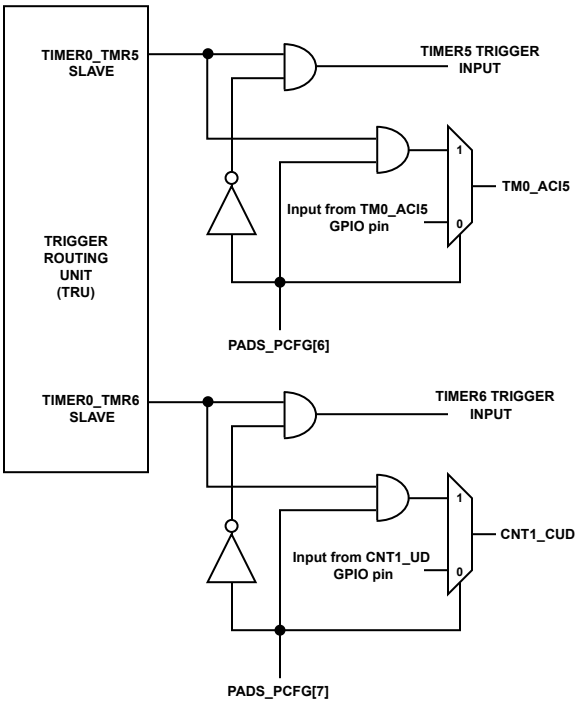

Figure 14-10: Trigger Slave Mode

If the PADS_PCONFIG0[6] bit is set (=1), the TIMER_TMR5_SLV0 slave trigger is routed to the TIMER_ACI5 input. With this configuration the input to TIMER_ACI5 is not routed from GPIO, it is routed from the TIMER_TMR5_SLV0 slave trigger pulses. Also the TIMER_TMR5_SLV0 slave trigger does not start or stop the Timer 5 as in normal slave operation.

If the PADS_PCONFIG0[6] bit is cleared (=0), the TIMER_TMR5_SLV0 slave trigger is used to start or stop the Timer 5 and the TIMER_ACI5 input is routed from the GPIO pins.

If the PADS_PCONFIG0[7] bit is set (=1), TIMER_TMR6_SLV0 slave trigger is routed to the CNT_UD input of GP Counter 1. With this configuration the input to CNT_UD is not routed from GPIO, it is routed from the TIMER_TMR6_SLV0 slave trigger pulses. Also the TIMER_TMR6_SLV0 slave trigger does not start or stop the Timer 6 as in normal slave operation.

If the PADS_PCONFIG0[7] bit is cleared (=0), the TIMER_TMR6_SLV0 slave trigger is used to start or stop Timer 6. The CNT_UD input of GP Counter 1 is routed from GPIO pins.

By default the PADS_PCONFIG0[6] and PADS_PCONFIG0[7] bits are cleared (=0) and Timer 5 and Timer 6 function as normal trigger slaves to start or stop the timers and the CNT_UD and TIMER_ACI5 input is routed from the GPIO pins.

To enable the TIMER_TMR5_SLV0 slave trigger pulses to be counted by Timer 5 for a particular TRU master, set the PADS_PCONFIG0[6] bit. Timer 5 can now be configured in Pin Interrupt (PININT) Mode or External Clock (EXTCLK) Mode to count the pulses on TIMER_ACI5 input.

To enable the TIMER_TMR6_SLV0 slave trigger pulses to be counted by the GP Counter for a particular TRU master, set the PADS_PCONFIG0[7] bit. GP Counter 1 can now be configured in Up/Down Counter Mode to count the pulses on the CNT_UD input.

GP Timer Programming Concepts

Using the features, operating modes, and event control for the GP timer to their greatest potential requires an understanding of some GP timer-related concepts.

Setting Up Constantly Changing Timer Conditions

This task shows how to use different period, pulse width, and delay settings for each of the first three timer periods after the timer starts.

1. Program the first set of TIMER_TMR[n]_PER, TIMER_TMR[n]_WID, and TIMER_TMR[n]_DLY register values.
2. Enable the timer using the TIMER_RUN register.
3. Immediately program the second set of TIMER_TMR[n]_PER, TIMER_TMR[n]_WID, and TIMER_TMR[n]_DLY register values, as needed.
4. Wait for the first timer interrupt.
5. Program the third set of TIMER_TMR[n]_PER, TIMER_TMR[n]_WID, and TIMER_TMR[n]_DLY register values.

Each new setting is then programmed when the preceding timer interrupt is received.

Configuring, Enabling, and Disabling One or More Timers

1. Configure the relevant timers for the operating mode and other properties using the TIMER_TMR[n]_CFG register.
2. Write a 1 to the representative TIMER_RUN.TMR[nn] bit. Or, use the TIMER_RUN_SET register to avoid disturbing the settings of other timers that are not going through configuration.

   The timer is enabled and operating.
3. To stop one or more timers, first program the TIMER_STOP_CFG register to determine whether to stop immediately or gracefully upon receiving a stop command.

   ADDITIONAL INFORMATION: PWMOUT modes are the only modes where a timer can be configured for graceful termination.
4. Write a 0 to the representative TIMER_RUN.TMR[nn] bits to stop the timer according to their TIMER_STOP_CFG settings. Alternately, write a 1 to the appropriate TIMER_RUN_CLR.TMR[nn] bits to avoid disturbing the settings of other timers that are not terminating.

   The timers stop.

Configuring Timer Data and Status Interrupts

1. Configure the TIMER_TMR[n]_CFG.IRQMODE bit field with the desired interrupt properties.
2. Unmask the interrupt source at the system event controller.
3. Set the TIMER_TMR[n]_CFG.IRQMODE field but leave the interrupt masked at the system level to poll the TIMER_DATA_ILAT.TMR[nn] bit of the timer without generating an interrupt.
4. Use the TIMER_STAT_IMSK register to generate interrupt requests by overflow or error conditions (incorrect programming values). The timer uses the TIMER_STAT_ILAT.TMR[nn] bits to report interrupt errors, when the timer status interrupt source is unmasked at the system event controller.
5. To poll the TIMER_STAT_ILAT.TMR[nn] bit of the timer without generating an interrupt, unmask the corresponding bit in the TIMER_STAT_IMSK register, but leave the interrupt masked at the system level.

Using the Timer Broadcast Feature

The broadcast feature provides a means to update period, width, and delay registers simultaneously across more than one timer.

1. Enable the appropriate broadcast bits (TIMER_TMR[n]_CFG.BPEREN, TIMER_TMR[n]_CFG.BWIDEN are TIMER_TMR[n]_CFG.BDLYEN) for the timers involved in the broadcast. The use of these bits depends on which broadcast registers the timer uses (TIMER_BCAST_PER, TIMER_BCAST_WID, or TIMER_BCAST_DLY).
2. Program the TIMER_BCAST_PER register (for example), to broadcast the period setting across the multiple timers enabled.

   The enabled timers load their TIMER_TMR[n]_PER registers with the value specified in the TIMER_BCAST_PER register.
3. Repeat Step 2 as needed for the TIMER_BCAST_WID and TIMER_BCAST_DLY register settings.

Timer Illegal States

The following sections use these definitions:

- Startup. The first clock period during which the timer counter is running after the timer is started by writing the TIMER_RUN register.
- Rollover. The time when the current count in TIMER_TMR[n]_CNT matches the value in TIMER_TMR[n]_PER and the counter is reloaded with the value 1.
- Overflow. The timer counter was incremented instead of doing a rollover when it was holding the maximum count value of 0xFFFF FFFF. The counter does not have a large enough range to express the next greater value and so it erroneously loads a new value of 0x0000 0000.
- Unchanged. No new error.

When the TIMER_ERR_TYPE register is designated unchanged, it displays the previously reported error code orb# 00 when there has been no error since this timer was enabled.

When the TIMER_STAT_ILAT register is unchanged, it reads 0 when there has been no error or overflow since this timer was enabled. Or, it reads 0 if software has performed a W1C to clear any previous error. If software has not acknowledged a previous error, the TIMER_STAT_ILAT register reads 1. Software can read the TIMER_STAT_ILAT register to check for errors. If a particular bit of a timer is set in this register, software can then read the TIMER_ERR_TYPE register for more information. Once detected, software can W1C the appropriate TIMER_STAT_ILAT bit to acknowledge the error.

Read the following tables as:

- In mode __ at event __,
- if TIMER_TMR[n]_PER is __ and TIMER_TMR[n]_WID is __ and TIMER_TMR[n]_DLY is __,
- then TIMER_ERR_TYPE is __ and TIMER_STAT_ILAT is __.

Startup error conditions do not prevent the timer from starting. Similarly, overflow and rollover error conditions do not stop the timer. Illegal cases can cause unwanted behavior of the TIMER_TMR[n] pin.

NOTE: For PININT mode, the timer does not use error functionality.

Continuous PWMOUT Mode

Table 14-11: Startup Event

TMR_PER	TMR_DLY	MR_WID	TMR_WID + TMR_DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
≤ 1	Anything other than period[8]	Anything	Anything	b#10	Set
≥ 2	Anything including 0, excluding TMR_PER value	Anything including 0	≤ PERIOD	Unchanged	Unchanged
	Anything including 0	Anything including 0	> PERIOD	Unchanged[9] (Detected at rollover)	Unchanged (Detected at rollover)
	Anything	Anything	> 2^{32} - 1	b#11	Set
	=Period	=0	=Period	No error	Unchanged (Detected at rollover)

Table 14-12: Rollover Event

TMR_PER	TMR_DLY	TMR_WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
≥ 1	Anything	Anything	Anything	b#10[timer running at SYSCLK] b#11 [timer running at ALT_CLKx]	Set

Table 14-12: Rollover Event (Continued)

TMR_PER	TMR_DLY	TMR_WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
≥ 2	Anything including 0, excluding TMR_PER value	Anything including 0	≤PERIOD	Unchanged	Unchanged
	Anything including 0, excluding TMR_PER value	Anything >0	>PERIOD	b#11	Set
	Anything	Anything	> 2^{32}- 1	b#11	Set
	= Period[10]	=0	=Period	b#11	Set
	>Period	=0	>Period	Unchanged	Unchanged

Table 14-13: Overflow Event (On TMR_PER Register Programming Error Only)

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
Anything	Anything	Anything	Anything	b#01	Set

Single Pulse PWMOUT Mode

For single pulse PWMOUT mode, there are no rollover events.

Table 14-14: Startup Event

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE TIMER_STAT_ILAT (if enabled)	
N/A	Anything	== 0	Anything	b#11[11]	Set
N/A	Anything including 0	≥1	> 2^{32} -1	Unchanged	Unchanged
N/A	Anything including 0	≥1	> 2^{32} -1	b#11	Set

Table 14-15: Overflow Event (On another error, such as DELAY + WIDTH >= 2^{32} -1)

TMR_PER	TMR_DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STA_ILAT (if enabled)
Anything	Anything	Anything	Anything	b#01	Set

WIDCAP Mode

For WIDCAP mode, the `TIMER_TMR[n]_PER` and `TIMER_TMR[n]_WID` registers are read-only and the `TIMER_TMR[n]_DLY` register is not used. Therefore, no startup or rollover errors are possible.

Table 14-16: Overflow Event

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
Anything	N/A	Anything	N/A	b#01	Set

EXTCLK Mode

Table 14-17: Startup Event

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
=0	N/A	N/A	N/A	b#01	Set
≥1	N/A	N/A	N/A	Unchanged	Unchanged

Table 14-18: Rollover Event

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
=0	N/A	N/A	N/A	b#01	Set
≥1	N/A	N/A	N/A	Unchanged	Unchanged

Table 14-19: Overflow Event (On TMR_PER Register = 0 Only)

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
Anything	N/A	N/A	N/A	b#01	Set

WATCHDOG Events

Table 14-20: Startup Event

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
≤ Allowed MIN[12]	Anything < PERIOD	N/A	N/A	b#01	Set
> Allowed MIN	Anything < PERIOD	N/A	N/A	Unchanged	Unchanged
> Allowed MIN	Anything ≥ PERIOD	Refer to WATCHDOG Mode tables			

Table 14-21: Rollover Event

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
≤ Allowed MIN[10]	Anything < PERIOD	N/A	N/A	b#01	Set
> Allowed MIN	Anything	N/A	N/A	Unchanged	Unchanged

Table 14-21: Rollover Event (Continued)

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
> Allowed MIN	Anything ≥ PERIOD	Refer to WATCHDOG Mode tables			

Table 14-22: Overflow Event

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
Anything	Anything	N/A	N/A	b#01	Set

ADSP-CM40x TIMER Register Descriptions

General-Purpose Timer Block (TIMER) contains the following registers.

Table 14-23: ADSP-CM40x TIMER Register List

Name	Description
TIMER_BCAST_DLY	Broadcast Delay Register
TIMER_BCAST_PER	Broadcast Period Register
TIMER_BCAST_WID	Broadcast Width Register
TIMER_DATA_ILAT	Data Interrupt Latch Register
TIMER_DATA_IMSK	Data Interrupt Mask Register
TIMER_ERR_TYPE	Error Type Status Register
TIMER_RUN	Run Register
TIMER_RUN_CLR	Run Clear Register
TIMER_RUN_SET	Run Set Register
TIMER_STAT_ILAT	Status Interrupt Latch Register
TIMER_STAT_IMSK	Status Interrupt Mask Register
TIMER_STOP_CFG	Stop Configuration Register
TIMER_STOP_CFG_CLR	Stop Configuration Clear Register
TIMER_STOP_CFG_SET	Stop Configuration Set Register
TIMER_TMR[n]_CFG	Timer n Configuration Register
TIMER_TMR[n]_CNT	Timer n Counter Register
TIMER_TMR[n]_DLY	Timer n Delay Register
TIMER_TMR[n]_PER	Timer n Period Register
TIMER_TMR[n]_WID	Timer n Width Register
TIMER_TRG_IE	Trigger Slave Enable Register

Table 14-23: ADSP-CM40x TIMER Register List (Continued)

Name	Description
TIMER_TRG_MSK	Trigger Master Mask Register

Broadcast Delay Register

For timers with TIMER_TMR[n]_CFG.BDLYEN enabled, a write to the TIMER_BCAST_DLY register concurrently updates the delay (TIMER_TMR[n]_DLY) registers of only those timers. A read of the TIMER_BCAST_DLY register returns 0x00000000, and no bus error is generated. To read back a written value, read that TMR's TIMER_TMR[n]_DLY register.

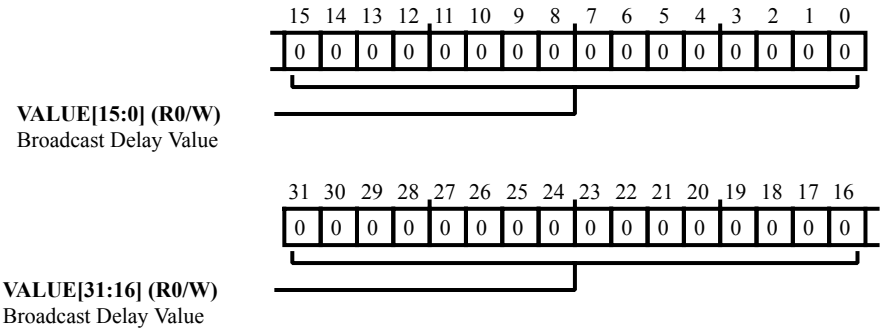

Figure 14-11: TIMER_BCAST_DLY Register Diagram

Table 14-24: TIMER_BCAST_DLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	VALUE	Broadcast Delay Value. A write to the TIMER_BCAST_DLY.VALUE bit field concurrently updates the delay (TIMER_TMR[n]_DLY) registers of only those timers. A read of the TIMER_BCAST_DLY.VALUE bit field returns 0x0000 0000, and no bus error is generated.

Broadcast Period Register

For timers with TIMER_TMR[n]_CFG.BPEREN enabled, a write to TIMER_BCAST_PER concurrently updates the period (TIMER_TMR[n]_PER) registers of only those timers. A read of TIMER_BCAST_PER returns 0x00000000, and no bus error is generated. To read back a written value, read that TMR's TIMER_TMR[n]_PER register.

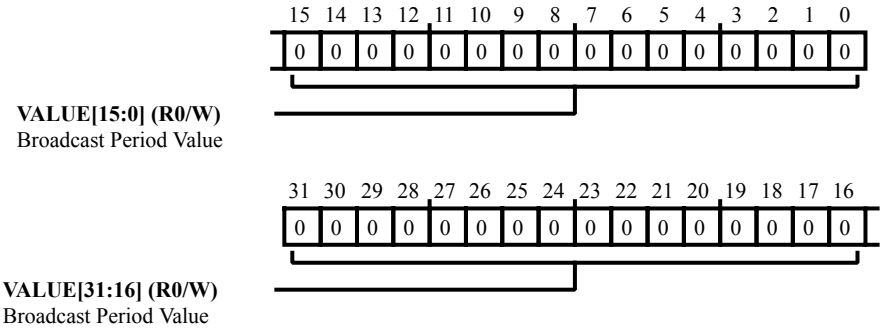

Figure 14-12: TIMER_BCAST_PER Register Diagram

Table 14-25: TIMER_BCAST_PER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	VALUE	Broadcast Period Value. A write to the TIMER_BCAST_PER.VALUE bit field concurrently updates the period (TIMER_TMR[n]_PER) registers of only those timers. A read of the TIMER_BCAST_PER.VALUE bit fields returns 0x0000 0000, and no bus error is generated.

Broadcast Width Register

For timers with TIMER_TMR[n]_CFG.BWIDEN enabled, a write to the TIMER_BCAST_WID register concurrently updates the' width (TIMER_TMR[n]_WID) registers of only those timers. A read of the TIMER_BCAST_WID register returns 0x00000000, and no bus error is generated. To read back a written value, read that TMR's TIMER_TMR[n]_WID register.

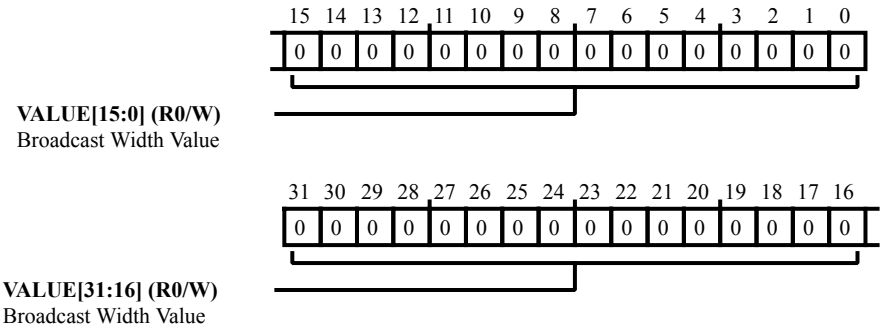

Figure 14-13: TIMER_BCAST_WID Register Diagram

Table 14-26: TIMER_BCAST_WID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	VALUE	Broadcast Width Value. A write to the TIMER_BCAST_WID.VALUE bit field concurrently updates the' width (TIMER_TMR[n]_WID) registers of only those timers. A read of the TIMER_BCAST_WID.VALUE bit field returns 0x0000 0000, and no bus error is generated.

Data Interrupt Latch Register

The TIMER_DATA_ILAT holds the latched interrupt status for interrupt requests that have been unmasked (enabled) by the TIMER_DATA_IMSK register and generated according to the conditions selected by the TIMER_TMR[n]_CFG.IRQMODE bits. If a bit in TIMER_DATA_ILAT is already set and the corresponding interrupt is masked in TIMER_DATA_IMSK, the latch holds its old value, leaving the interrupt asserted until it is reset by software with a W1C operation.

Note that interrupt service routines (ISRs) should clear the appropriate bits in TIMER_DATA_ILAT before returning from the ISR, to ensure that the interrupt is not re-issued. To make sure that no timer event is missed, the latch should be reset at the very beginning of the ISR when in EXTCLK mode.

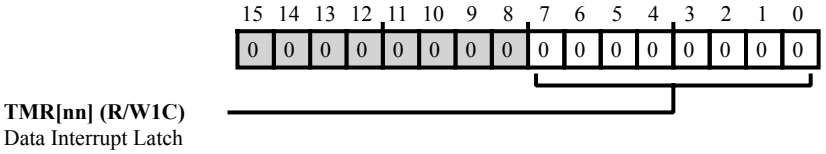

Figure 14-14: TIMER_DATA_ILAT Register Diagram

Table 14-27: TIMER_DATA_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W1C)	TMR[nn]	Data Interrupt Latch. For all TIMER_DATA_ILAT.TMR[nn] bits, status of =0 indicates no interrupt is latched, and status of =1 indicates a latched interrupt (indicating an unmasked interrupt request from a timer with a condition matching the one selected with corresponding TIMER_TMR[n]_CFG.IRQMODE bit has occurred).

Data Interrupt Mask Register

Each timer may generate a unique processor data interrupt request signal. The TIMER_DATA_IMSK register contains an interrupt mask for these requests, masking (disabling) or unmasking (enabling) the interrupts as programmed. The reset value of the TIMER_DATA_IMSK register is 0xFFFF, masking these interrupts after reset.

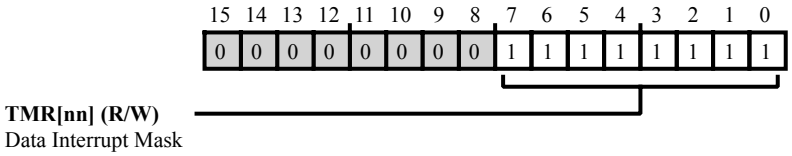

Figure 14-15: TIMER_DATA_IMSK Register Diagram

Table 14-28: TIMER_DATA_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMR[nn]	Data Interrupt Mask. For all TIMER_DATA_IMSK.TMR[nn] bits, write =0 unmasks (enables) the corresponding data interrupt request, and write =1 masks (disables) the corresponding data interrupt request.

Error Type Status Register

The TIMER_ERR_TYPE register contains Error Type status bits for each timer. These bits indicate the type of error (if any) in a running timer. This register is read-only. These status bits are cleared at reset and when a particular timer is enabled.

Each time an error interrupt is latched in the TIMER_STAT_ILAT register the corresponding TERRx bits in the TIMER_ERR_TYPE register are loaded with a code that identifies the type of error that was detected. This status value is held until the next error or until a particular timer is restarted. No bus error is generated if a write is performed on the TIMER_ERR_TYPE register.

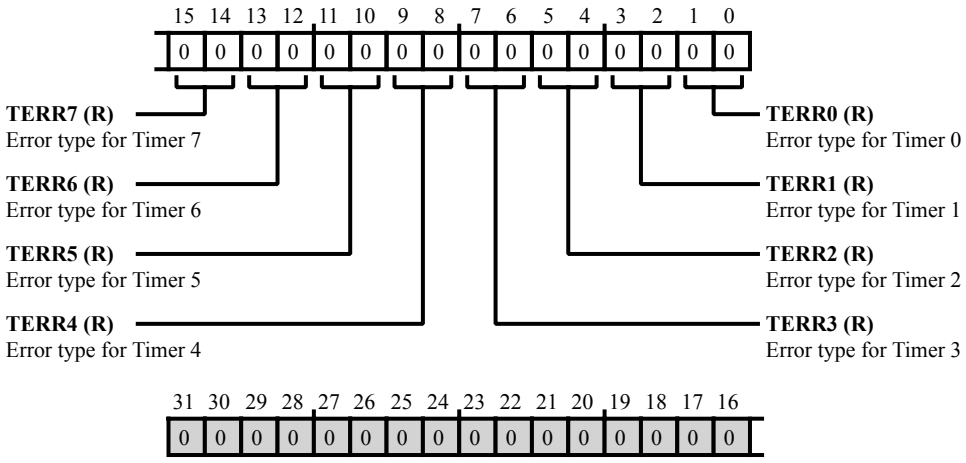

Figure 14-16: TIMER_ERR_TYPE Register Diagram

Table 14-29: TIMER_ERR_TYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:14 (R/NW)	TERR7	Error type for Timer 7.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error
13:12 (R/NW)	TERR6	Error type for Timer 6.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error

Table 14-29: TIMER_ERR_TYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/NW)	TERR5	Error type for Timer 5.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error
9:8 (R/NW)	TERR4	Error type for Timer 4.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error
7:6 (R/NW)	TERR3	Error type for Timer 3.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error
5:4 (R/NW)	TERR2	Error type for Timer 2.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error
3:2 (R/NW)	TERR1	Error type for Timer 1.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error
1:0 (R/NW)	TERR0	Error type for Timer 0.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error

Run Register

The TIMER_RUN allows all timers to be enabled simultaneously, permitting them to run synchronously. For each timer, there is a single start/stop control bit. Writing a 1 to this bit starts the corresponding timer; writing a 0 stops the timer with mechanism specified in the timer stop configuration TIMER_STOP_CFG register.

The start/stop control bits can be set/reset individually or in any combination. While starting or stopping one particular timer directly with this register, software must perform a read-modify write, so the bits corresponding to other timers remain unchanged. To avoid this need, software can use the TIMER_RUN_CLR register.

Reading the TIMER_RUN register shows the start status for the corresponding timer. A 1 indicates that the timer is running.

If a timer is in run state (corresponding run bit is =1), a software write of 1 in this bit does not have any effect on the timer state. The write does not result in restarting the timer.

Note that the TIMER_RUN register is not used in PININT mode. PININT mode starts as soon as the TIMER_TMR[n]_CFG.TMODE bits are set to 111.

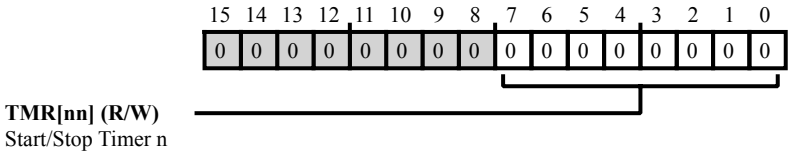

Figure 14-17: TIMER_RUN Register Diagram

Table 14-30: TIMER_RUN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMR[nn]	Start/Stop Timer n. For all TIMER_RUN.TMR[nn] bits, write =0 for stop, and write =1 for start. Read =1 when timer is running.

Run Clear Register

The TIMER_RUN_CLR register is an alias register, providing a mechanism to clear a specific start/stop bit in the TIMER_RUN register without affecting other bits in TIMER_RUN. To stop a particular timer, software must write a 1 into the corresponding TIMER_RUN_CLR bit. Writing a 0 has no effect. Because TIMER_RUN_CLR is a write-only register, the result of any write to this register must be checked by reading the TIMER_RUN register. A read of the TIMER_RUN_CLR returns 0x0000.

Note that the stopping mechanism of a timer may be abrupt or graceful (after completion of current waveform period) depending on the selection in the TIMER_STOP_CFG register.

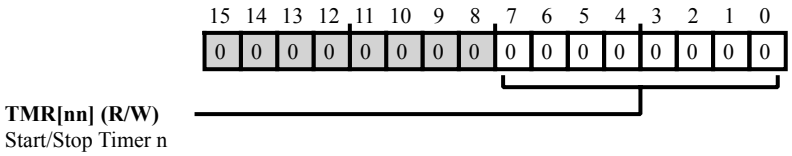

Figure 14-18: TIMER_RUN_CLR Register Diagram

Table 14-31: TIMER_RUN_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W1C)	TMR[nn]	RUN Clear Alias. For all TIMER_RUN_CLR.TMR[nn] bits, write =0 has no effect, and write =1 for stop (clearing the corresponding in start/stop bit in the TIMER_RUN register). Using TIMER_RUN_CLR to clear start/stop bits permits stopping specific timers without influencing run status of other timers.

Run Set Register

The TIMER_RUN_SET register is an alias register, providing a mechanism to set a specific start/stop bit in the TIMER_RUN register without affecting other bits in TIMER_RUN. To start a particular timer, software must write a 1 into the corresponding TIMER_RUN_SET bit. Writing a zero has no effect. For an example, to start timer 3 without affecting any other timer, write 0x0008 into TIMER_RUN_SET. Because TIMER_RUN_SET is a write-only register, the result of any write to this register must be checked by reading the TIMER_RUN register. A read of the TIMER_RUN_SET returns 0x0000.

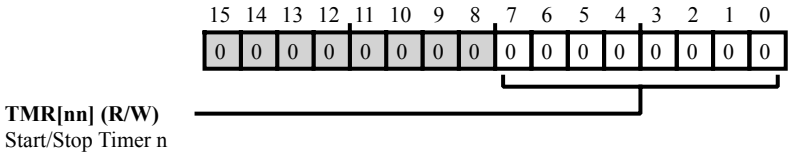

Figure 14-19: TIMER_RUN_SET Register Diagram

Table 14-32: TIMER_RUN_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W1S)	TMR[nn]	RUN Set Alias. For all TIMER_RUN_SET.TMR[nn] bits, write =0 has no effect, and write =1 for start (setting the corresponding start/stop bit in the TIMER_RUN register). Using TIMER_RUN_SET to set start/stop bits permits starting specific timers without influencing the run status of other timers.

Status Interrupt Latch Register

The TIMER_STAT_ILAT holds the latched interrupt status for error interrupts, indicating a timer overflow condition or indicating that prohibited programming has occurred for a timer. These interrupt status bits are sticky and are W1C. The bits in the TIMER_STAT_ILAT register provide information regarding each timer interrupt source.

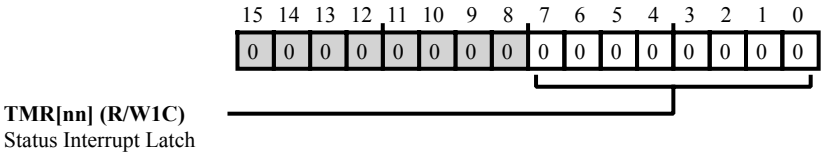

Figure 14-20: TIMER_STAT_ILAT Register Diagram

Table 14-33: TIMER_STAT_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W1C)	TMR[nn]	Status Interrupt Latch. For all TIMER_STAT_ILAT.TMR[nn] bits, status of 0 indicates no error interrupt is latched, and status of 1 indicates a timer counter overflow or programming error interrupt is latched.

Status Interrupt Mask Register

While each timer may generate a status interrupt request, these requests are OR'ed to generate a single status interrupt signal to the System Event Controller. The `TIMER_STAT_IMSK` register contains an interrupt mask for these requests, masking (disabling) or unmasking (enabling) the interrupts as programmed. The reset value of the `TIMER_STAT_IMSK` register is 0xFFFF, masking these interrupts after reset.

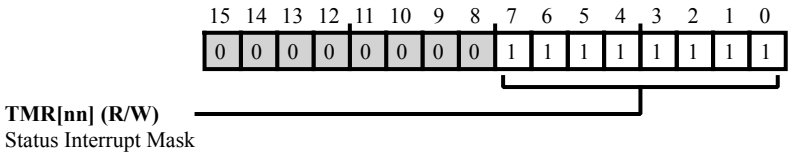

Figure 14-21: TIMER_STAT_IMSK Register Diagram

Table 14-34: TIMER_STAT_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMR[nn]	Status Interrupt Mask. For all `TIMER_STAT_IMSK.TMR[nn]` bits, write =0 unmasks (enables) the corresponding status interrupt request, and write =1 masks (disables) the corresponding status interrupt request.

Stop Configuration Register

The TIMER_STOP_CFG register selects the stop mode for each timer. Timers may be stopped abruptly (immediate halt - all modes) or gracefully in PWMOUT modes (single pulse and continuous). The halt is achieved through either a write =0 to the corresponding bit in TIMER_RUN or a write =1 to the corresponding bit in TIMER_RUN_CLR. A read of TIMER_STOP_CFG returns the last value written.

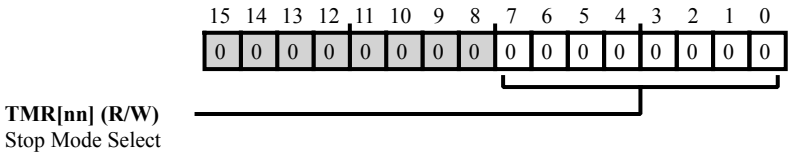

Figure 14-22: TIMER_STOP_CFG Register Diagram

Table 14-35: TIMER_STOP_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMR[nn]	Stop Mode Select. For all TIMER_STOP_CFG.TMR[nn] bits, write =0 for graceful termination (PWMOUT modes only), and write =1 for abrupt (immediate halt) on stop.

Stop Configuration Clear Register

This is an alias register, providing a mechanism to clear a specific bit in the TIMER_STOP_CFG register without affecting other bits in TIMER_STOP_CFG. To clear a bit in TIMER_STOP_CFG, software must write a 1 to the corresponding bit of TIMER_STOP_CFG_CLR register. Writing a zero has no effect. Because the TIMER_STOP_CFG_CLR register is a write-only register, the result of any write to this register must be checked by reading the TIMER_STOP_CFG register. A read of the TIMER_STOP_CFG_CLR register returns 0x0000.

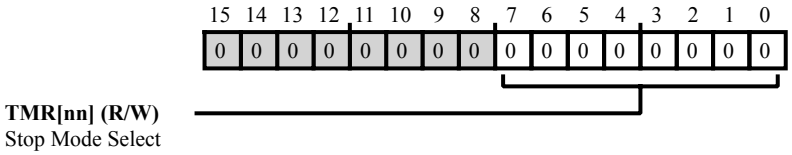

Figure 14-23: TIMER_STOP_CFG_CLR Register Diagram

Table 14-36: TIMER_STOP_CFG_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W1C)	TMR[nn]	STOP_CFG Clear Alias. For all TIMER_STOP_CFG_CLR.TMR[nn] bits, write =0 has no effect, and write =1 for graceful stop in PWMOUT modes (clearing the corresponding stop mode select bit in the TIMER_STOP_CFG register). Using TIMER_STOP_CFG_CLR to clear stop mode bits permits configuring specific timers without influencing the stop mode configuration of other timers.

Stop Configuration Set Register

This is an alias register, providing a mechanism to set a specific bit in the TIMER_STOP_CFG register without affecting other bits in TIMER_STOP_CFG. To set a bit in the TIMER_STOP_CFG register, software must write a 1 to the corresponding bit of the TIMER_STOP_CFG_SET register. Writing a zero has no effect. Because the TIMER_STOP_CFG_SET register is a write-only register, the result of any write to this register must be checked by reading the TIMER_STOP_CFG register. A read of the TIMER_STOP_CFG_SET register returns 0x0000.

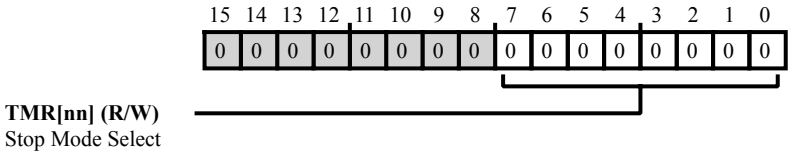

Figure 14-24: TIMER_STOP_CFG_SET Register Diagram

Table 14-37: TIMER_STOP_CFG_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W1S)	TMR[nn]	STOP_CFG Set Alias. For all TIMER_STOP_CFG_SET.TMR[nn] bits, write =0 has no effect, and write =1 for abrupt stop (setting the corresponding stop mode select bit in the TIMER_STOP_CFG register). Using TIMER_STOP_CFG_SET to set stop mode bits permits configuring specific timers without influencing the stop mode configuration of other timers.

Timer n Configuration Register

Each timer has a TIMER_TMR[n]_CFG register that specifies its operating mode. Only write to a TIMER_TMR[n]_CFG register when the corresponding timer is not running.

After disabling a timer operating in PWMOUT mode, verify that the timer has stopped running by checking the start/stop status of the timer in the TIMER_RUN register before writing to the timer's TIMER_TMR[n]_CFG register.

Note that a timer's TIMER_TMR[n]_CFG register may be read at any time.

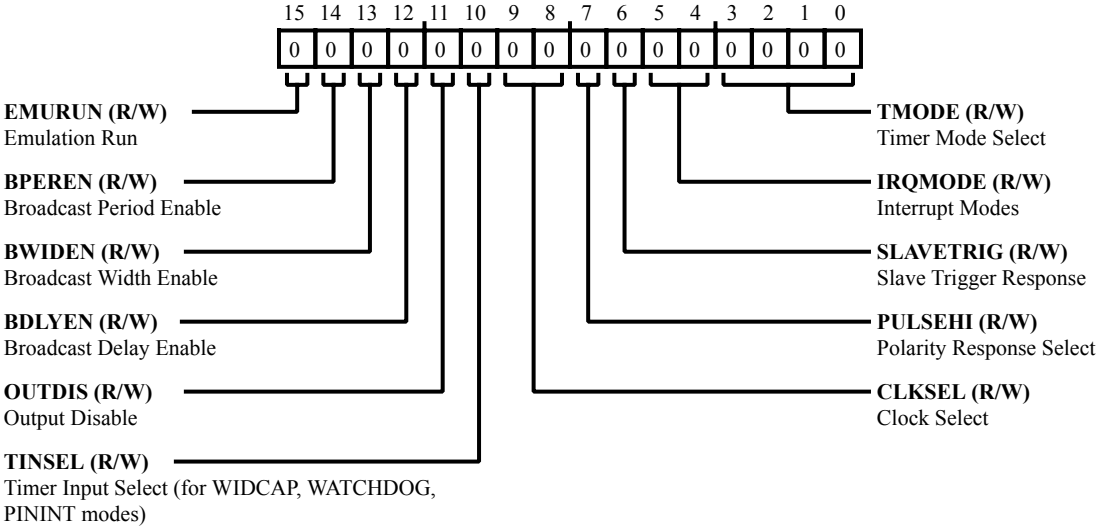

Figure 14-25: TIMER_TMR[n]_CFG Register Diagram

Table 14-38: TIMER_TMR[n]_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	EMURUN	Emulation Run. The TIMER_TMR[n]_CFG.EMURUN bit causes the timer to run (count) during emulation.	
		0	Stop Timer During Emulation
		1	Run Timer During Emulation
14 (R/W)	BPEREN	Broadcast Period Enable. The TIMER_TMR[n]_CFG.BPEREN bit enables updates to the TIMER_TMR[n]_PER register simultaneously across more than one timer.	
		0	Disable Broadcast to PER Register
		1	Enable Broadcast to PER Register

Table 14-38: TIMER_TMR[n]_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	BWIDEN	Broadcast Width Enable. The `TIMER_TMR[n]_CFG.BWIDEN` bit enables updates to the `TIMER_TMR[n]_WID` register simultaneously across more than one timer.	
		0	Disable Broadcast to WID Register
		1	Enable Broadcast to WID Register
12 (R/W)	BDLYEN	Broadcast Delay Enable. The `TIMER_TMR[n]_CFG.BDLYEN` bit enables updates to the `TIMER_TMR[n]_DLY` register simultaneously across more than one timer.	
		0	Disable Broadcast to DLY Register
		1	Enable Broadcast to DLY Register
11 (R/W)	OUTDIS	Output Disable. The `TIMER_TMR[n]_CFG.OUTDIS` bit enables or disables the timer pin output buffer.	
		0	Enable TMR pin output buffer
		1	Disable TMR pin output buffer
10 (R/W)	TINSEL	Timer Input Select (for WIDCAP, WATCHDOG, PININT modes).	
		0	Use TMR pin input
		1	Use TMR Alternate Capture Input
9:8 (R/W)	CLKSEL	Clock Select. The `TIMER_TMR[n]_CFG.CLKSEL` bit field selects the TIMER clock to use.	
		0	Use SYSCLK
		1	Use TMR_ALT_CLK0 as the TMR clock
		3	Use TMR_ALT_CLK1 as the TMR clock
7 (R/W)	PULSEHI	Polarity Response Select. The `TIMER_TMR[n]_CFG.PULSEHI` bit defines specific behaviors of the timer based on the operating mode. For more information, see the specific operating mode in the Programming Guidelines section.	
		0	Negative Response/Pulse Negative Edge Response or Negative Action Pulse on TMR pin
		1	Positive Response/Pulse Positive Edge Response or Positive Action Pulse on TMR pin

Table 14-38: TIMER_TMR[n]_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	SLAVETRIG	Slave Trigger Response. Note that the trigger pulse has no effect (to stop or start the timer) if the timer is already in the requested state.	
		0	Pulse stops timer if it is running
		1	Pulse starts timer if it is stopped
5:4 (R/W)	IRQMODE	Interrupt Modes. The TIMER_TMR[n]_CFG.IRQMODE bit field selects the interrupt request mode. Note that any mismatched combination of TIMER_TMR[n]_CFG.IRQMODE and TIMER_TMR[n]_CFG.TMODE results in no interrupt being generated. Also note that in WIDCAP modes, the position of the interrupt is controlled with the TIMER_TMR[n]_CFG.TMODE bit, and the TIMER_TMR[n]_CFG.IRQMODE bit is ignored.	
		0	Active Edge Mode The timer generates an interrupt at every active edge. The active edge polarity depends on the state of the TIMER_TMR[n]_CFG.PULSEHI bit). Valid for PININT mode only.
		1	Delay Expired Mode The timer generates an interrupt when the TIMER_TMR[n]_CNT value reaches the value in the TIMER_TMR[n]_DLY register. This mode is valid for all PWMOUT modes.
		2	Width Plus Delay Expired Mode The timer generates an interrupt when the TIMER_TMR[n]_CNT value reaches the value in the TIMER_TMR[n]_WID register plus the value in the TIMER_TMR[n]_DLY register. (PWMOUT modes only). The timer generates an interrupt if the de-asserting edge is within the specified window. (WATCHDOG modes only.)
		3	Period Expired Mode The timer generates an interrupt when the TIMER_TMR[n]_CNT value reaches the value in the TIMER_TMR[n]_PER register. (Continuous PWMOUT and EXTCLK modes only.) The timer generates an interrupt if the de-asserting edge is outside the specified window.(WATCHDOG modes only.)
3:0 (R/W)	TMODE	Timer Mode Select. The TIMER_TMR[n]_CFG.TMODE bit field selects the operating mode of each timer.	
		0-7	Idle Mode

Table 14-38: TIMER_TMR[n]_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		8	Period Watchdog Mode
		9	Width Watchdog Mode
		10	Measurement report at asserting edge of waveform
		11	Measurement report at de-asserting edge of waveform
		12	Continuous PWMOUT mode
		13	Single pulse PWMOUT mode
		14	EXTCLK mode
		15	PININT (pin interrupt) mode

Timer n Counter Register

The TIMER_TMR[n]_CNT register holds the current timer count. After enabling, the count is re-initialized to either 0x0 or 0x1, depending on the configuration and mode. The TIMER_TMR[n]_CNT is read only and may be read at any time (whether the timer is running or stopped). Reading the TIMER_TMR[n]_CNT register returns an atomic 32-bit value.

Depending on the timer operation mode, the counter increment can be clocked by a number of sources, including SYSCLK, the TMR or Alternate Capture input pins, TIMER_ACLK[n]. The counter retains its value after the timer is disabled.

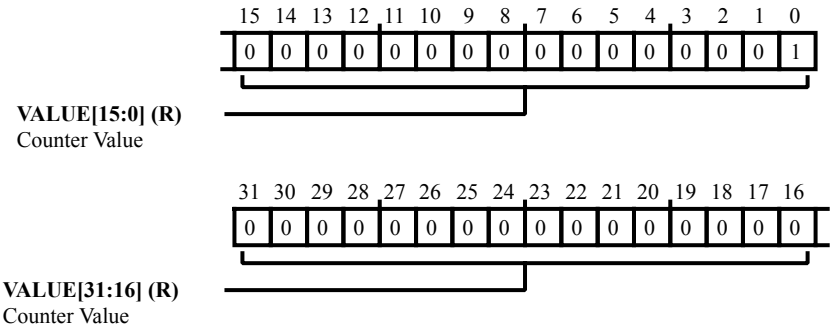

Figure 14-26: TIMER_TMR[n]_CNT Register Diagram

Table 14-39: TIMER_TMR[n]_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Counter Value. The TIMER_TMR[n]_CNT.VALUE bit field holds the current timer count.

Timer n Delay Register

The TIMER_TMR[n]_DLY register holds the delay value for the corresponding timer. This register's use is based on the selected timer mode.

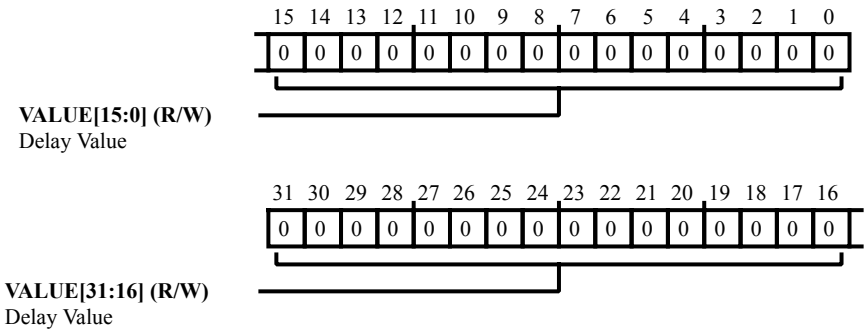

Figure 14-27: TIMER_TMR[n]_DLY Register Diagram

Table 14-40: TIMER_TMR[n]_DLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Delay Value. The TIMER_TMR[n]_DLY.VALUE bit field holds the delay value for the corresponding timer.

Timer n Period Register

The TIMER_TMR[n]_PER register holds the period value for the corresponding timer. This register's use is based on the selected timer mode.

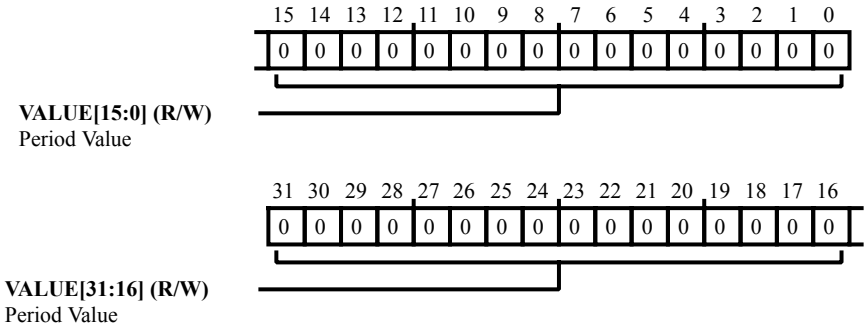

Figure 14-28: TIMER_TMR[n]_PER Register Diagram

Table 14-41: TIMER_TMR[n]_PER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Period Value. The TIMER_TMR[n]_PER.VALUE bit field holds the period value for the corresponding timer.

Timer n Width Register

The TIMER_TMR[n]_WID register holds the width value for the corresponding timer. This register's use is based on the selected timer mode.

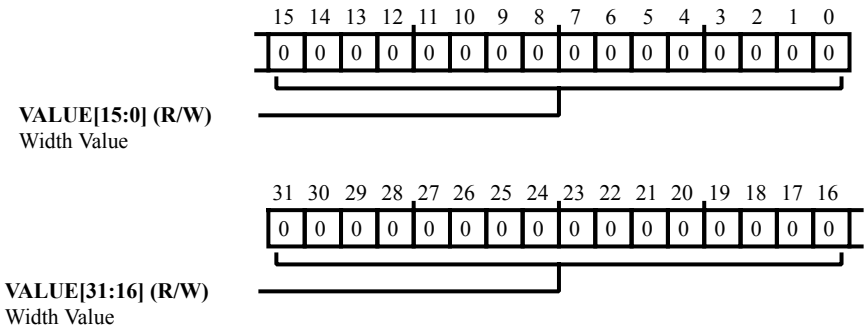

Figure 14-29: TIMER_TMR[n]_WID Register Diagram

Table 14-42: TIMER_TMR[n]_WID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Width Value. The TIMER_TMR[n]_WID.VALUE bit field holds the width value for the corresponding timer.

Trigger Slave Enable Register

As a trigger slave, each timer can generate a unique data trigger pulse signal. The `TIMER_TRG_IE` contains trigger input enable bits for these signals, disabling or enabling the triggers as programmed. The reset value of the `TIMER_TRG_IE` register is 0xFFFF, masking these triggers after reset.

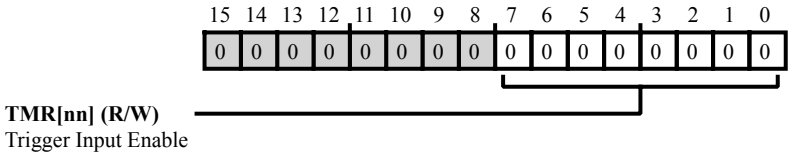

Figure 14-30: TIMER_TRG_IE Register Diagram

Table 14-43: TIMER_TRG_IE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMR[nn]	Trigger Input Enable. For all `TIMER_TRG_IE.TMR[nn]` bits, write =0 disables the corresponding trigger input, and write =1 enables the corresponding trigger input.

Trigger Master Mask Register

As a trigger master, each timer can generate a unique data trigger pulse signal. The TIMER_TRG_MSK register contains a trigger mask for these outputs, masking (disabling) or unmasking (enabling) the triggers as programmed. The reset value of the TIMER_TRG_MSK register is 0xFFFF, masking these triggers after reset.

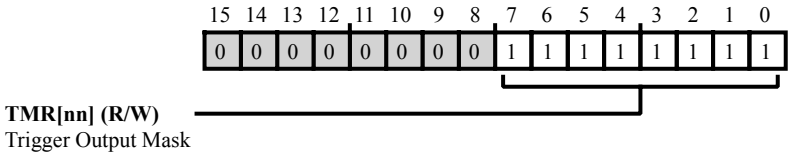

Figure 14-31: TIMER_TRG_MSK Register Diagram

Table 14-44: TIMER_TRG_MSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMR[nn]	Trigger Output Mask. For all TIMER_TRG_MSK.TMR[nn] bits, write =0 unmasks (enables) the corresponding data trigger output, and write =1 masks (disables) the corresponding data trigger output.

15 Capture Timer (CPTMR)

The Capture Timer (CPTMR) module measures the total on-time of the input signal between two triggers. The module performs this measurement by counting clock pulses during the on-time of the input signal. The total on-time is captured when the stop trigger is received.

The CPTMR is used in inverter applications to measure the total inverter on-time. The application can capture the total on-time of the inverter output between PWM cycles. The triggers required to start and stop the capture timer are received from the TRU, where the PWM is the master.

CPTMR Functional Description

The ADSP-CM40x has three capture timers. Each capture timer block can be enabled and configured to capture the total on-time of the `CPTMR_IN[n]` input pin between two trigger assertions (`CPTMR_CPT[n]_SLV0` or `CPTMR_CPT[n]_SLV1`). The core of the capture timer is a 32-bit counter that can be interrogated through the read-only `CPTMR_CNT[n]` register. The counter is reset to 0x1 when the start trigger is received. Between the start and stop triggers, the total on-time of the `CPTMR_IN[n]` input pin is captured in the `CPTMR_TON[n]` register.

CPTMR Block Diagram

The ***Capture Timer Block Diagram*** illustrates the block diagram of the CPTMR module. The block diagram is comprised of two primary sections:

Internal Interface. The core always accesses the capture timer registers through the peripheral bus interface. The capture timer has dedicated data and status interrupt request outputs that connect to the SEC module.

External Interface. The capture timer has an asynchronous input pin (`CPTMR_IN[n]`) and asynchronous trigger inputs (`CPTMR_CPT[n]_SLV0` and `CPTMR_CPT[n]_SLV1`). The requirement for minimum input pulse width timing on the external pins is $2 \times t_{SCLK}$, to guarantee that SYSCLK samples the asynchronous pulse.

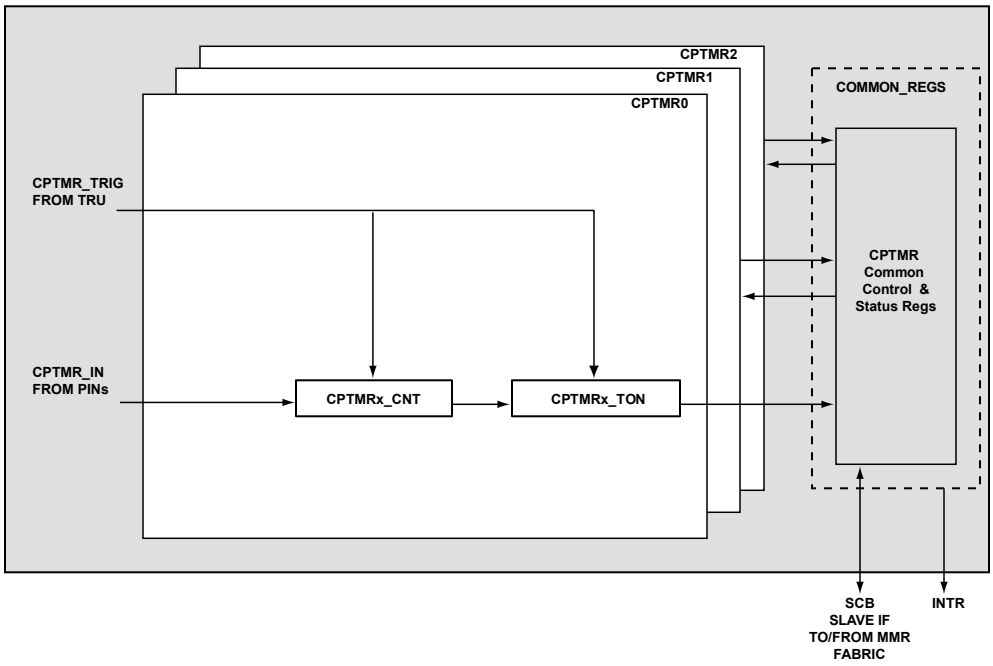

Figure 15-1: Capture Timer Block Diagram

ADSP-CM40x CPTMR Register List

The Capture Timer (CPTMR) measures total on-time of the input signal between two triggers.

Table 15-1: ADSP-CM40x CPTMR Register List

Name	Description
CPTMR_CFG[n]	Configuration Register
CPTMR_CNT[n]	Counter Register
CPTMR_DATA_ILAT	Data Interrupt Latch Status Register
CPTMR_DATA_IMSK	Data Interrupt Mask Register
CPTMR_DATA_IMSK_CLR	Data Interrupt Mask Clear Register
CPTMR_DATA_IMSK_SET	Data Interrupt Mask Set Register
CPTMR_RUN	Run Register
CPTMR_RUN_CLR	Run Clear Register
CPTMR_RUN_SET	Run Set Register
CPTMR_STAT_ILAT	Interrupt Latch Status Register
CPTMR_STAT_IMSK	Status Interrupt Mask Register
CPTMR_STAT_IMSK_CLR	Status Interrupt Mask Clear Register
CPTMR_STAT_IMSK_SET	Status Interrupt Mask Set Register

Table 15-1: ADSP-CM40x CPTMR Register List (Continued)

Name	Description
CPTMR_TON[n]	On-time Capture Register

ADSP-CM40x CPTMR Trigger List

Table 15-2: ADSP-CM40x CPTMR Trigger List Masters

Trigger ID	Name	Description	Sensitivity
None			

Table 15-3: ADSP-CM40x CPTMR Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
63	CPTMR0_CPT0_SLV0	CPTMR0 Capture Timer 0 Trigger 0	Level
64	CPTMR0_CPT0_SLV1	CPTMR0 Capture Timer 0 Trigger 1	Level
65	CPTMR0_CPT1_SLV0	CPTMR0 Capture Timer 1 Trigger 0	Level
66	CPTMR0_CPT1_SLV1	CPTMR0 Capture Timer 1 Trigger 1	Level
67	CPTMR0_CPT2_SLV0	CPTMR0 Capture Timer 2 Trigger 0	Level
68	CPTMR0_CPT2_SLV1	CPTMR0 Capture Timer 2 Trigger 1	Level

ADSP-CM40x CPTMR Interrupt List

Table 15-4: ADSP-CM40x CPTMR Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
129	CPTMR0_CPT0_ERR	CPTMR0 Status	Level	
130	CPTMR0_CPT1_ERR	CPTMR0 Status	Level	
131	CPTMR0_CPT2_ERR	CPTMR0 Status	Level	
132	CPTMR0_CPT0_MEAS	CPTMR0 Status	Level	
133	CPTMR0_CPT1_MEAS	CPTMR0 Status	Level	
134	CPTMR0_CPT2_MEAS	CPTMR0 Status	Level	

CPTMR Operation

The CPTMR is used to measure the total on-time of the input waveform between two trigger assertions. When the capture timer is enabled, the start trigger resets the count in the CPTMR_CNT[n] register to 1, and the timer starts counting and incrementing on the following SYSCLK rising edges, if the CPTMR_IN[n] input pin is asserted.

The CPTMR_IN[n] input pin is used directly or with inverted polarity, based on the CPTMR_CFG[n].TINPOL bit setting. Similarly, the capture timer trigger is used directly or with inverted polarity, based on the CPTMR_CFG[n].TRIGPOL bit setting. On the start trigger assertion, the contents of the CPTMR_CNT[n] register are transferred to the CPTMR_TON[n] register and the CPTMR_CNT[n] register is reset to 0x1. The capture timer counter continues counting and capturing until the timer is disabled.

When the 32-bit counter range exceeds 0xFFFFFFFF, the counter rolls over to 0, and an error report interrupt is generated. The capture timer status interrupt latch register (CPTMR_STAT_ILAT) bits indicate the occurrence of the overflow, if enabled in the status interrupt mask register (CPTMR_STAT_IMSK) bits. To measure on-time longer than 0xFFFF FFFF cycles, software can count the number of errors reported between measurement report interrupts. Since the reset value of the counter is 1, the first overflow error report interrupt adds 2^{32}- 1 SYSCLK cycles to the total on-time captured. As the counter rolls over to 0 after the overflow, 2^{32} SYSCLK cycles are added to the total on-time from the second overflow interrupt onward.

The timer data interrupt latch bits in the CPTMR_DATA_ILAT register are set (if enabled) to indicate an interrupt request from a particular capture timer instance. The on-time data interrupt is generated when the stop trigger is asserted. The bits in the CPTMR_DATA_ILAT and CPTMR_STAT_ILAT registers are sticky bits, and software has to explicitly clear them. Disabling the CPTMR module clears the CPTMR_DATA_ILAT and CPTMR_STAT_ILAT register bits along with the corresponding IRQ.

Since the timer can also accept external asynchronous inputs, the external input signal is shifted by close to 2 SYSCLK cycles to align the input edge with the SYSCLK edge. This synchronized input is then provided to the capture timer module for on-time calculations. There can be a maximum accuracy loss of 2 SYSCLK cycles in on-time calculations, but only if the input level is active and very near the leading edge of a trigger. The input given in the following figure is a synchronized signal going to the capture timer module.

Refer to the ***On-Time Capture Timing (TINPOL =0 and TRIGPOL =0)*** figure. For CPTMR_CFG[n].TINPOL =0 and CPTMR_CFG[n].TRIGPOL =0, the counter value in CPTMR_CNT[n] is reset to 1 at the assertion of the start trigger. Thereafter, the counter value increments with every SYSCLK rising edge if the CPTMR_IN[n] input is active. The CPTMR_TON[n] total on-time is then copied from the CPTMR_CNT[n] counter value. The CPTMR_CNT[n] count value is reset to 1 when the stop trigger is asserted. The operation continues this way for subsequent trigger assertions until the timer is disabled.

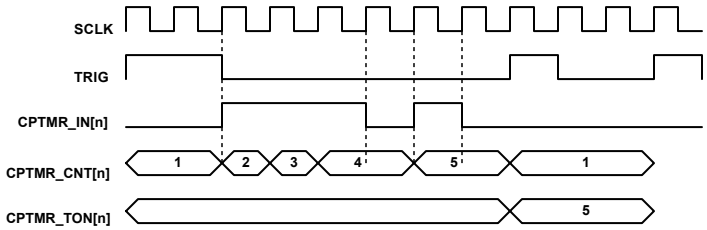

Figure 15-2: On-Time Capture Timing (TINPOL =0 and TRIGPOL =0)

CPTMR Interrupt Signals

The CPTMR is capable of signaling the core about its state and error conditions that occur during its operation, by providing status and data interrupts.

Data Interrupts

The capture timer can generate a processor data interrupt request signal if the data interrupt is unmasked by programming a 0 in the mask bit of the timer data interrupt mask register (CPTMR_DATA_IMSK). For a data interrupt to be serviced, clear the CPTMR_DATA_ILAT register with software using a W1C operation. The total on-time can be monitored when the data interrupt is generated at the assertion of the stop trigger.

Status Interrupts

The capture timer indicates a counter overflow condition through the status interrupt latch register (CPTMR_STAT_ILAT). The capture timer can generate a status interrupt for an overflow condition if the status interrupt is unmsasked by programming 0 in the mask bit of the status interrupt mask register (CPTMR_STAT_IMSK). The interrupt is serviced by performing a W1C operation to the CPTMR_STAT_ILAT register.

CPTMR Programming Model (ADSP-CM40x)

The processor contains three instances of the capture timers.

The CPTMR_IN[n] input signals to all timers come from external port pins, and the input triggers (CPTMR_CPT[n]_SLV0 or CPTMR_CPT[n]_SLV1) come from the TRU module. Use the following general procedure to configure and initiate the CPTMR module.

1. Configure the CPTMR with the following settings and the appropriate polarity of the trigger and input signals.
   - CPTMR_CFG[n].TRIG_INEN =1
   - CPTMR_CFG[n].TIN_INEN =1
2. Configure the TRU to assign a master to generate start and stop trigger for the capture timer.
3. Start the timer by configuring the CPTMR_RUN register.
4. Send trigger and input signals to the timer and monitor the total on-time when the measurement report is received (if CPTMR_DATA_IMSK.CPT[nn] bit =0).

ADSP-CM40x CPTMR Register Descriptions

Capture Timer (CPTMR) contains the following registers.

Table 15-5: ADSP-CM40x CPTMR Register List

Name	Description
CPTMR_CFG[n]	Configuration Register
CPTMR_CNT[n]	Counter Register
CPTMR_DATA_ILAT	Data Interrupt Latch Status Register
CPTMR_DATA_IMSK	Data Interrupt Mask Register

Table 15-5: ADSP-CM40x CPTMR Register List (Continued)

Name	Description
CPTMR_DATA_IMSK_CLR	Data Interrupt Mask Clear Register
CPTMR_DATA_IMSK_SET	Data Interrupt Mask Set Register
CPTMR_RUN	Run Register
CPTMR_RUN_CLR	Run Clear Register
CPTMR_RUN_SET	Run Set Register
CPTMR_STAT_ILAT	Interrupt Latch Status Register
CPTMR_STAT_IMSK	Status Interrupt Mask Register
CPTMR_STAT_IMSK_CLR	Status Interrupt Mask Clear Register
CPTMR_STAT_IMSK_SET	Status Interrupt Mask Set Register
CPTMR_TON[n]	On-time Capture Register

Configuration Register

The CPTMR_CFG[n] register contains bits that are used to configure various module options.

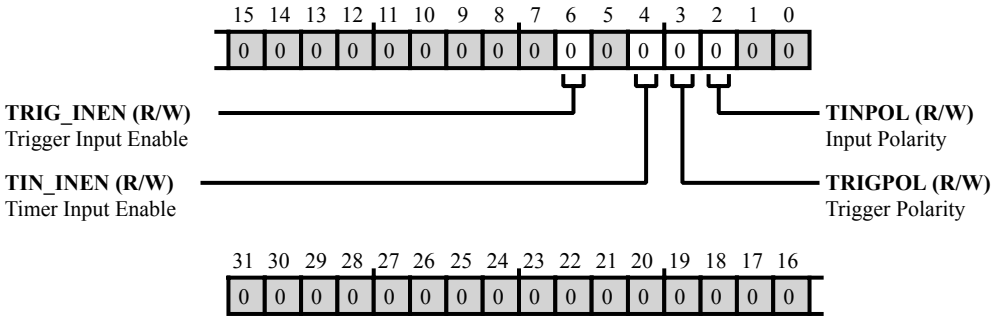

Figure 15-3: CPTMR_CFG[n] Register Diagram

Table 15-6: CPTMR_CFG[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W)	TRIG_INEN	Trigger Input Enable. The CPTMR_CFG[n].TRIG_INEN bit enables trigger input (=1).
4 (R/W)	TIN_INEN	Timer Input Enable. The CPTMR_CFG[n].TIN_INEN bit enables timer input (=1).
3 (R/W)	TRIGPOL	Trigger Polarity. The CPTMR_CFG[n].TRIGPOL bit selects whether a trigger is not inverted (active high/rising edge as leading edge, =0) or a trigger is inverted (active low/falling edge as leading edge, =1).
2 (R/W)	TINPOL	Input Polarity. The CPTMR_CFG[n].TINPOL bit selects whether input is not inverted (active high/rising edge as leading edge, =0) or input is inverted (active low/falling edge as leading edge, =1).

Counter Register

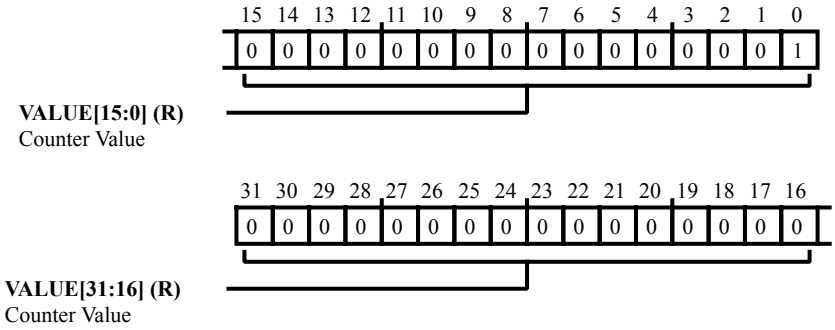

Figure 15-4: CPTMR_CNT[n] Register Diagram

Table 15-7: CPTMR_CNT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Counter Value.

Data Interrupt Latch Status Register

The CPTMR_DATA_ILAT register indicates data interrupt requests.

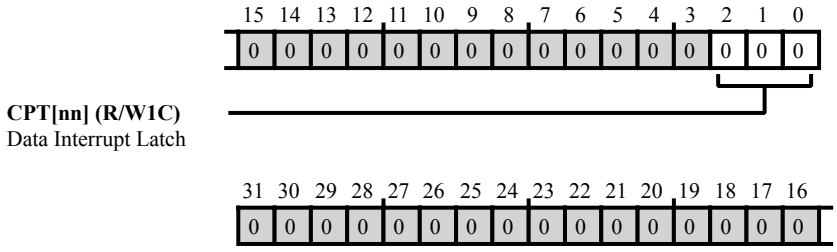

Figure 15-5: CPTMR_DATA_ILAT Register Diagram

Table 15-8: CPTMR_DATA_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W1C)	CPT[nn]	Data Interrupt Latch. A 1 in any of the CPTMR_DATA_ILAT.CPT[nn] bits indicates a data interrupt request from timer n.

Data Interrupt Mask Register

The CPTMR_DATA_IMSK register is used to mask data interrupts.

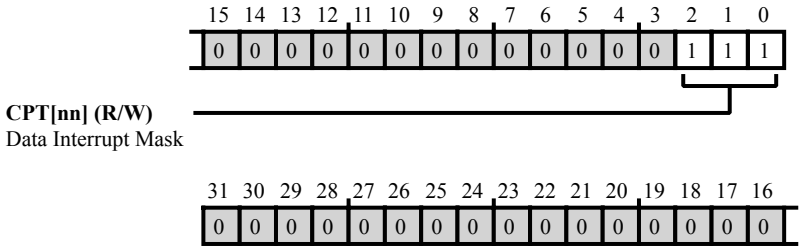

Figure 15-6: CPTMR_DATA_IMSK Register Diagram

Table 15-9: CPTMR_DATA_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W)	CPT[nn]	Data Interrupt Mask. Writing a 0 to any of the CPTMR_DATA_IMSK.CPT[nn] bits enables the data interrupts for timer n.

Data Interrupt Mask Clear Register

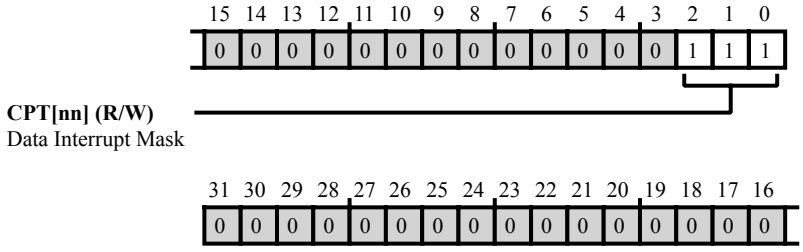

Figure 15-7: CPTMR_DATA_IMSK_CLR Register Diagram

Table 15-10: CPTMR_DATA_IMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W1C)	CPT[nn]	Data Interrupt Mask Clear. Writing a 1 to any of the `CPTMR_DATA_IMSK_CLR.CPT[nn]` bits enables the data interrupt for timer n. Writing a 0 has no effect.

Data Interrupt Mask Set Register

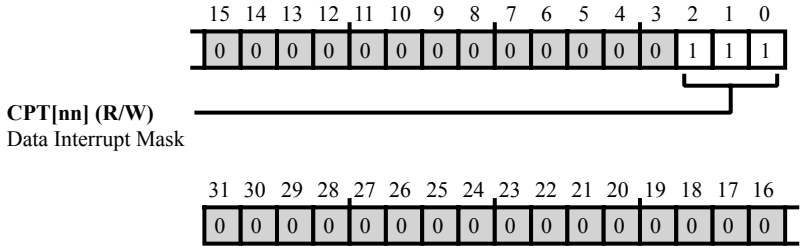

Figure 15-8: CPTMR_DATA_IMSK_SET Register Diagram

Table 15-11: CPTMR_DATA_IMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W1S)	CPT[nn]	Data Interrupt Mask Set. Writing a 1 to any of the `CPTMR_DATA_IMSK_SET.CPT[nn]` bits enables the data in-terrupt for timer n. Writing a 0 has no effect.

Run Register

Writing a 1 to bit n of the CPTMR_RUN register starts timer n. Writing a 0 to bit n halts timer n.

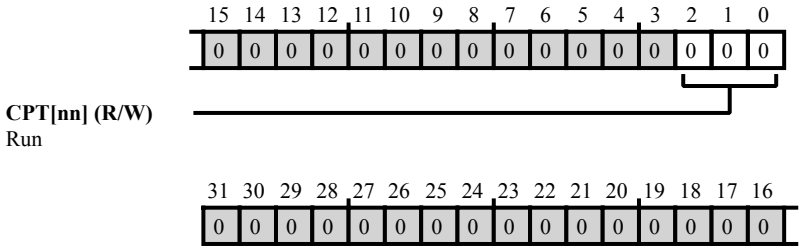

Figure 15-9: CPTMR_RUN Register Diagram

Table 15-12: CPTMR_RUN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W)	CPT[nn]	Run. Writing 1 to any of the CPTMR_RUN.CPT[nn] bits starts timer n. Writing 0 stops the timer.

Run Clear Register

Writing a 1 to bit n of the CPTMR_RUN_CLR register stops timer n. Writing a 0 has no effect.

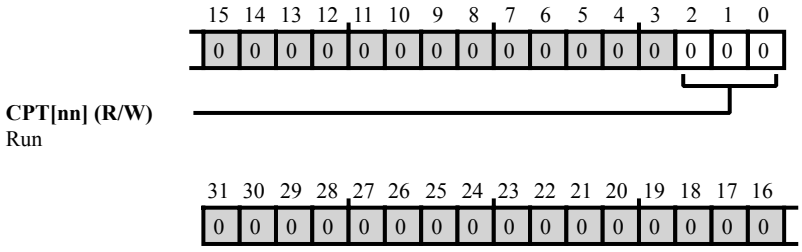

Figure 15-10: CPTMR_RUN_CLR Register Diagram

Table 15-13: CPTMR_RUN_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W1C)	CPT[nn]	Run Clear. Writing a 1 to any of the CPTMR_RUN_CLR.CPT[nn] bits stops timer n. Writing a 0 has no effect.

Run Set Register

Writing a 1 to bit n of the CPTMR_RUN_SET register starts timer n. Writing a 0 has no effect.

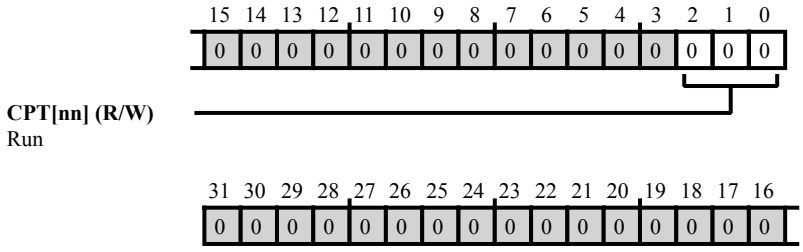

Figure 15-11: CPTMR_RUN_SET Register Diagram

Table 15-14: CPTMR_RUN_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W1S)	CPT[nn]	Run Set Register. Write a 1 to any of the CPTMR_RUN_SET.CPT[nn] bits to start timer n.

Interrupt Latch Status Register

The CPTMR_STAT_ILAT register indicates counter overflow errors.

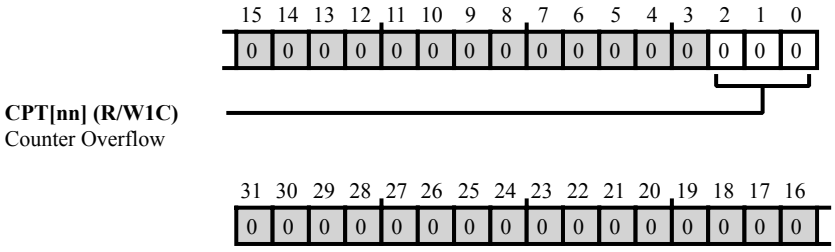

Figure 15-12: CPTMR_STAT_ILAT Register Diagram

Table 15-15: CPTMR_STAT_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W1C)	CPT[nn]	Counter Overflow. A 1 in any of the CPTMR_STAT_ILAT.CPT[nn] bits indicates an overflow error for timer n.

Status Interrupt Mask Register

The CPTMR_STAT_IMSK register is used to mask status interrupts.

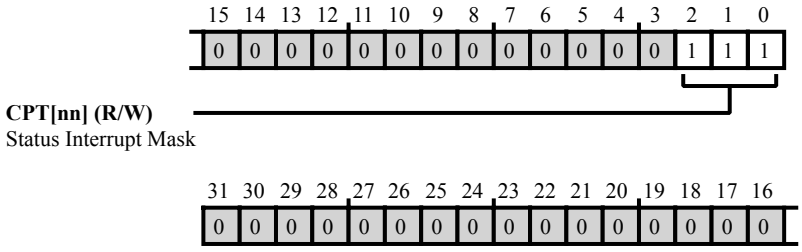

Figure 15-13: CPTMR_STAT_IMSK Register Diagram

Table 15-16: CPTMR_STAT_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W)	CPT[nn]	Status Interrupt Mask. Writing a 0 to any of the CPTMR_STAT_IMSK.CPT[nn] bits enables the status interrupt for timer n.

Status Interrupt Mask Clear Register

The CPTMR_STAT_IMSK_CLR register is used to unmask status interrupts.

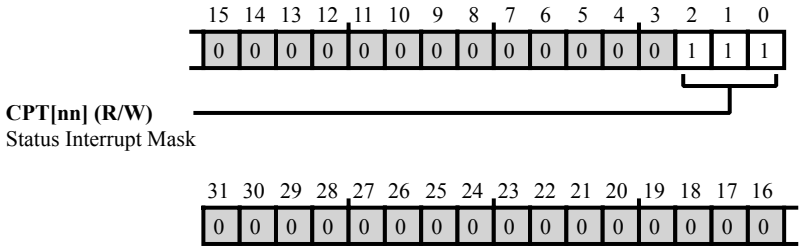

Figure 15-14: CPTMR_STAT_IMSK_CLR Register Diagram

Table 15-17: CPTMR_STAT_IMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W1C)	CPT[nn]	Status Interrupt Mask Clear. Writing a 1 to any of the CPTMR_STAT_IMSK_CLR.CPT[nn] bits enables the status interrupt for timer n. Writing a 0 has no effect.

Status Interrupt Mask Set Register

The CPTMR_STAT_IMSK_SET register is used to mask status interrupts.

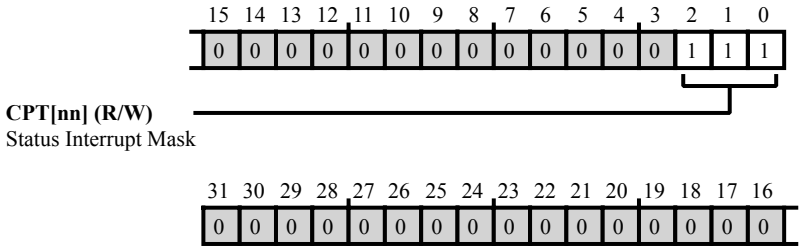

Figure 15-15: CPTMR_STAT_IMSK_SET Register Diagram

Table 15-18: CPTMR_STAT_IMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W1S)	CPT[nn]	Status Interrupt Mask Set. Writing a 1 to any of the CPTMR_STAT_IMSK_SET.CPT[nn] bits disables the status interrupt for timer n. Writing a 0 has no effect.

On-time Capture Register

The CPTMR_TON[n] register captures the total on-time of the input waveform.

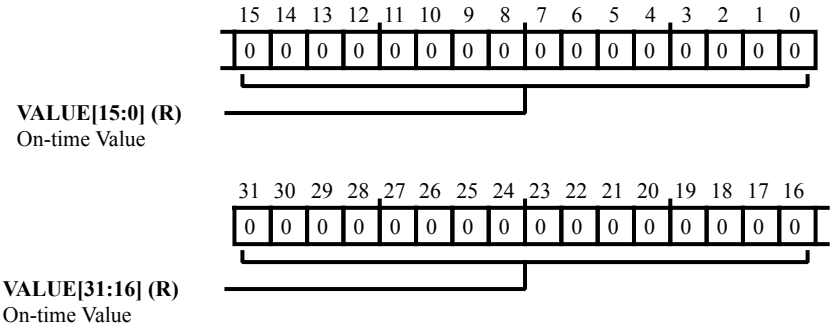

Figure 15-16: CPTMR_TON[n] Register Diagram

Table 15-19: CPTMR_TON[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	On-time Value.

16 Watchdog Timer (WDOG)

The processor has a 32-bit timer for each core that can implement a software watchdog function. A software watchdog can improve system reliability by generating an event to the processor core when the watchdog expires before software updates it. The system clock (SYSCLK) clocks the watchdog timers.

WDOG Features

The watchdog timer has the following features.

- 32-bit watchdog timers
- 8-bit disable bit pattern
- Can generate a general-purpose event for the core

Typically, the processor uses the watchdog timer to supervise stability of the system software. When used in this way, software reloads the watchdog timer in a regular manner so that the downward counting timer never expires (never becomes 0). An expiring timer then indicates that system software can be out of control. Based on the GP event generated by the WDOG, it is often better to reset and reboot the system using the reset control unit.

For easier debugging, the watchdog timer does not decrement (even if enabled) when the processor is in emulation mode.

WDOG Functional Description

When enabled, the 32-bit watchdog timer counts downward every SYSCLK cycle. When the count becomes 0, the expiry event is generated. This generates a GP event to the processor. When the event is generated, software must reset the core and the peripherals. The counter value can be read through the 32-bit `WDOG_STAT` register. The `WDOG_STAT` register cannot, however, be written directly. Rather, software writes the watchdog period value into the 32-bit `WDOG_CNT` register before the watchdog is enabled. Once the watchdog is started, the period value cannot be altered.

ADSP-CM40x WDOG Register List

The Watchdog Timer unit (WDOG) provides a software-based watchdog timer that can improve system reliability by generating an event to the processor core if the watchdog expires before being updated by software. A set of

registers governs WDOG operations. For more information on WDOG functionality, see the WDOG register descriptions.

Table 16-1: ADSP-CM40x WDOG Register List

Name	Description
WDOG_CNT	Count Register
WDOG_CTL	Control Register
WDOG_STAT	Watchdog Timer Status Register

ADSP-CM40x WDOG Interrupt List

Table 16-2: ADSP-CM40x WDOG Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
8	WDOG0_EXP	WDOG0 Expiration	Level	

WDOG Block Diagram

The *Watchdog Timer Block Diagram* figure shows the detailed block diagram for the watchdog timer.

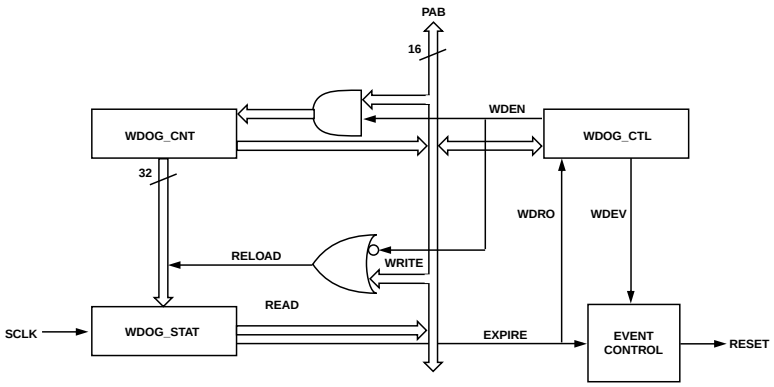

Figure 16-1: Watchdog Timer Block Diagram

Internal Interface

The system clock (SYSCLK) clocks the watchdog timer. The registers are accessed through the 16-bit peripheral MMR access bus. 32-bit read/write operations always access the 32-bit WDOG_CNT and WDOG_STAT registers. Hardware ensures that those accesses are atomic. When the counter expires, the GP expiration event is generated.

External Interface

The watchdog timer does not directly interact with any pins of the chip.

ADSP-CM40x WDOG Register Descriptions

Watchdog Timer Unit (WDOG) contains the following registers.

Table 16-3: ADSP-CM40x WDOG Register List

Name	Description
WDOG_CNT	Count Register
WDOG_CTL	Control Register
WDOG_STAT	Watchdog Timer Status Register

Count Register

The WDOG_CNT register holds the programmable, unsigned count value. A valid write to this register also pre-loads the WDOG counter. For added safety, the WDOG_CNT register can be updated only when the WDOG timer is disabled. A write to the WDOG_CNT register while the timer is enabled does not modify the contents of this register. This register must be accessed with 32-bit read/writes only.

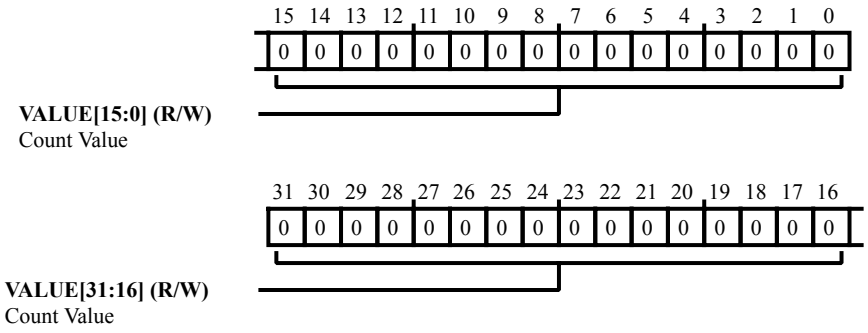

Figure 16-2: WDOG_CNT Register Diagram

Table 16-4: WDOG_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Count Value. The WDOG_CNT.VALUE bit field holds the programmable, unsigned count value.

Control Register

The WDOG_CTL register controls the watch dog timer. This register supports enabling/disabling the watch dog timer and supports checking the timer rollover status. Note that when the processor is in emulation mode, the watch dog timer counter will not decrement even if it is enabled.

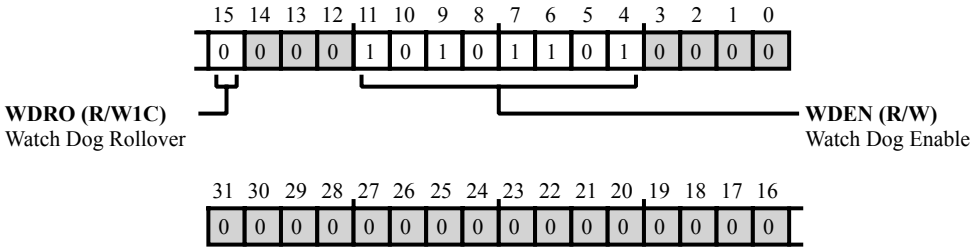

Figure 16-3: WDOG_CTL Register Diagram

Table 16-5: WDOG_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	WDRO	Watch Dog Rollover. Software can determine whether the timer has rolled over by interrogating the WDOG_CTL.WDRO status bit. This is a sticky bit that is set whenever the watch dog timer count reaches 0 and cleared only by disabling the watch dog timer and then writing a 1 to the bit.	
		0	WDT has not expired
		1	WDT has expired
11:4 (R/W)	WDEN	Watch Dog Enable. The WDOG_CTL.WDEN field is used to enable and disable the watch dog timer. Writing any value other than the disable value into this field enables the watch dog timer. This multi-bit disable key minimizes the chance of inadvertently disabling the watch dog timer.	
		173	173 = Counter Disabled All other values = counter enabled

Watchdog Timer Status Register

The WDOG_STAT contains the current count value of the watch dog timer. Reads of this register return the current count value. When the watch dog timer is enabled, WDOG_STAT is decremented by 1 on each SYSCLK cycle. When the count value reaches 0, the watch dog timer stops counting, and the expiry event is generated. The WDOG_STAT register is a 32-bit unsigned system MMR that must be accessed with 32-bit reads and writes.

Values cannot be stored directly in this register but are instead copied from the WDOG_CNT register. This copy process can happen in two ways:

- While the watchdog timer is disabled, writing the WDOG_CNT register pre-loads the WDOG_STAT register.
- While the watch dog timer is enabled, writing the WDOG_STAT register loads it with the value in WDOG_CNT
- While the watchdog timer is disabled, writing to the WDOG_STAT register also reloads it with the value in the WDOG_CNT register.

When the processor executes a write (of an arbitrary value) to WDOG_STAT, the value in WDOG_CNT is copied into WDOG_STAT. Typically, software sets the value of WDOG_CNT at initialization, then periodically writes to WDOG_STAT before the watch dog timer expires. This reloads the watchdog timer with the value from WDOG_CNT and prevents generation of the expiry event.

If the program does not reload the counter before SYSCLK x Count register cycles, an expiry event is generated, and the WDOG_CTL.WDRO bit is set. When this happens, the counter stops decrementing and remains at zero. If the counter is enabled with a zero loaded to the counter, the WDOG_CTL.WDRO bit is set immediately and the counter remains at zero and does not decrement.

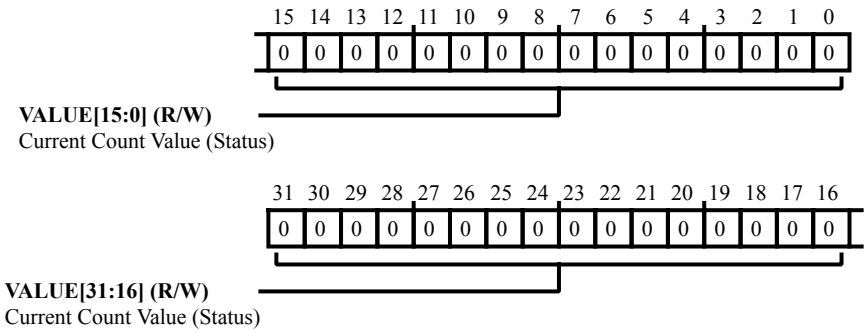

Figure 16-4: WDOG_STAT Register Diagram

Table 16-6: WDOG_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Current Count Value (Status). The WDOG_STAT.VALUE bit field contains the current count value of the watch dog timer.

17 General-Purpose Counter (CNT)

The GP counter converts pulses from incremental position encoders into data that is representative of the actual position of the pulse. This conversion is done by integrating (counting) pulses on one or two inputs. Since integration provides relative position, some devices also feature a zero-position input (zero marker). The GP counter can use the zero position input feature to establish a reference point for verifying that the acquired position does not drift over time. In addition, the GP counter can use the incremental position information to determine speed, if the time intervals are measured.

The GP counter provides flexible ways to establish position information. When used with the GP timer block, the GP counter can allow for the acquisition of coherent position or time-stamp information that enables speed calculation.

GP Counter Features

The GP counter includes the following features:

- 32-bit up or down counter
- Quadrature encode mode (Gray code)
- Binary encoder mode
- Alternative frequency-direction mode
- Timed direction and up or down counting modes
- Zero marker or push-button support
- Capture event timing in association with GP Timer
- Boundary comparison and boundary setting features
- M/N frequency scaling of the inputs CUD/CDG

GP Counter Functional Description

The ***GP Counter Block Diagram*** shows a block diagram of the GP counter. The CNT_UD and CNT_DG pins accept various forms of incremental inputs. The 32-bit counter processes the inputs. The GP counter uses the CNT_ZM pin to sense the pressing of a push button.

NOTE: When enabled, the GP counter requires 3 SYSCLK cycles of initialization before recognizing valid toggles on its input pins.

The three input pins can be filtered (debounced) before the GP counter evaluates them.

The GP counter also features a flexible boundary comparison. In all of the operating modes, the counter can be compared to an upper and lower limit. It takes various actions when reaching these limits.

The module can optionally generate an interrupt request to the system through its IRQ line. On many processors, a GP timer module can use an output to generate timestamps on certain events.

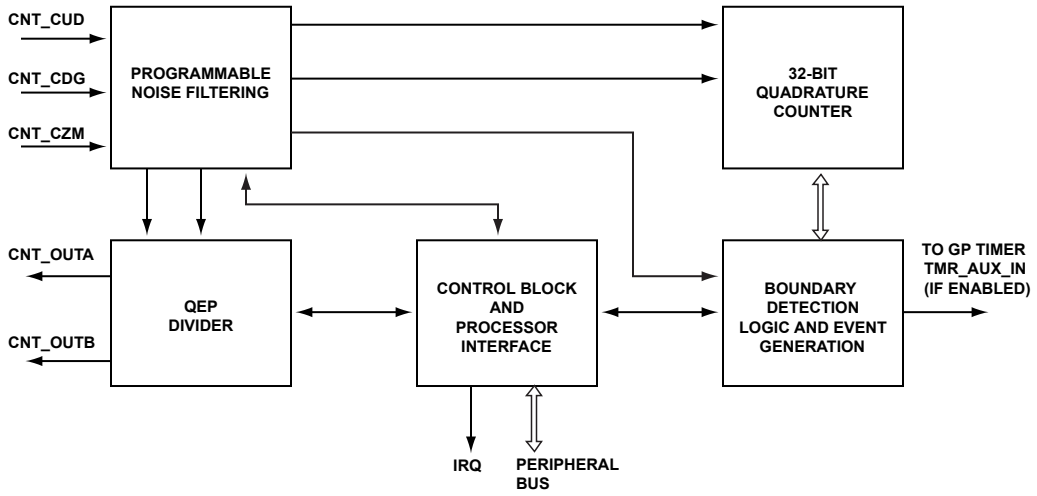

Figure 17-1: GP Counter Block Diagram

ADSP-CM40x CNT Register List

The GP Counter (CNT) provides support for manually controlled rotary controllers, such as the volume wheel on a radio device. This unit also supports industrial encoders.

The CNT converts pulses from incremental position encoders into data that is representative of the actual position. To complete this task, the CNT integrates (counts) pulses on one or two inputs. Because integration provides relative position, a zero position input (zero marker) is usually provided that establishes a reference point, verifying that the acquired position does not drift over time. The incremental position information may also be used to determine speed, if the relevant time intervals are measured. The CNT provides flexible ways to establish position information. When used in conjunction with the General-Purpose Timer (TIMER), the CNT allows acquisition of coherent position/time-stamp information, enabling speed calculation.

A set of registers govern CNT operations. For more information on CNT functionality, see the CNT register descriptions.

Table 17-1: ADSP-CM40x CNT Register List

Name	Description
CNT_CFG	Configuration Register
CNT_CMD	Command Register
CNT_CNTR	Counter Register
CNT_DEBNCE	Debounce Register
CNT_IMSK	Interrupt Mask Register
CNT_MAX	Maximum Count Register
CNT_MDIV	M Value for Divider
CNT_MIN	Minimum Count Register
CNT_NDIV	N Value for Divider
CNT_STAT	Status Register

ADSP-CM40x CNT Interrupt List

Table 17-2: ADSP-CM40x CNT Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
59	CNT0_STAT	CNT0 Count Status	Level	
60	CNT1_STAT	CNT1 Count Status	Level	
61	CNT2_STAT	CNT2 Count Status	Level	
62	CNT3_STAT	CNT3 Count Status	Level	

ADSP-CM40x CNT Trigger List

Table 17-3: ADSP-CM40x CNT Trigger List Masters

Trigger ID	Name	Description	Sensitivity
15	CNT0_STAT	CNT0 Counter Status	Level
16	CNT1_STAT	CNT1 Counter Status	Level
17	CNT2_STAT	CNT2 Counter Status	Level
18	CNT3_STAT	CNT3 Counter Status	Level

Table 17-4: ADSP-CM40x CNT Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
8	CNT1_UD	CNT1 Counter Up/Down Input (alternate configuration)	Level

GP Counter Operating Modes

The GP counter has the following five modes of operation.

1. Quadrature Encoder
2. Binary Encoder
3. Up/Down Counter
4. Direction Counter
5. Timed Direction

With the exception of the timed direction mode, the GP counter can operate with the GP timer block to capture additional timing information (timestamps) associated with events detected by this block.

Quadrature Encoder Mode

In this mode, the CNT_UD and CNT_DG inputs expect a quadrature-encoded signal that is interpreted as a two-bit Gray code. The order of transitions of the CNT_UD and CNT_DG inputs determines whether the counter increments or decrements. The CNT_CNTR register contains the number of transitions that have occurred as shown in the ***Quadrature Events and Counting Mechanism*** table. Optionally, an interrupt is generated when both inputs change within one SYSCLK cycle. Gray coding prohibits such transitions. Therefore, the CNT_CNTR register remains unchanged, and an error condition is signaled.

Table 17-5: Quadrature Events and Counting Mechanism

CNT_COUNTER Register Value	-4	-3	-2	-1	0	+1	+2	+3	+4
CDG, CUD Inputs	00	01	11	10	00	01	11	10	00

It is possible to reverse the count direction of the Gray-coded signal by enabling the polarity inverter of either the CNT_UD pin or the CNT_DG pin. Inverting both pins does not alter the behavior. The GP counter can enable this feature with the CNT_CFG.CDGINV and CNT_CFG.CUDINV bits.

As an example, the CNT_DG and CNT_UD inputs are 00 and the next transition is to 01. These inputs normally change the counter in increments as shown in the table. If the CNT_UD polarity is inverted, this condition generates a received input of 01 followed by 00. The results is a decrement of the counter, altering the behavior of the connected hardware.

Binary Encoder Mode

This mode is almost identical to quadrature encoder mode, with the exception that the CNT_UD: CNT_DG inputs expect a binary-encoded signal. The order of transitions of the CNT_UD and CNT_DG inputs determines whether the counter increments or decrements. The CNT_CNTR register contains the number of transitions that have occurred as shown in the ***Binary Events and Counting Mechanism*** table. Optionally, an interrupt is generated when the detected code steps by more than 1 (in binary arithmetic) within one SYSCLK cycle. Such transitions are erroneous. Therefore, the CNT_CNTR register remains unchanged, and an error condition is signaled.

Table 17-6: Binary Events and Counting Mechanism

CNT_COUNTER Register Value	-4	-3	-2	-1	0	+1	+2	+3	+4
CDG:CUD Inputs	00	01	10	11	00	01	10	11	00

Reversing the CNT_UD and CNT_DG pin polarity has a different effect in binary encoder mode than for the quadrature encoder mode. Inverting the polarity of the CNT_UD pin only, or inverting both the CNT_UD and CNT_DG pins, results in reversing the count direction.

Up/Down Counter Mode

In this mode, the counter increments or decrements at every active edge of the input pins. The GP counter uses the CNT_CFG.CUDINV bit to select an active edge and has the following results.

- If the GP counter module detects an active edge at the CNT_UD input, the counter increments.
- If the GP counter module detects an active edge at the CNT_DG input, the counter decrements.
- If simultaneous edges occur on the CNT_DG and CNT_UD pins, the counter remains unchanged, and both up-count and down-count events are signaled in the CNT_STAT register.

Direction Counter Mode

In this mode, the counter is incremented or decremented at every active edge of the CNT_DG input pin. The state of the CNT_UD input determines whether the counter increments or decrements. The GP counter uses the CNT_CFG.CUDINV bit to select the polarity.

If the GP counter detects an active edge at the CNT_DG input, the counter value changes by one in the selected direction.

Timed Direction Mode

In this mode, the counter is incremented or decremented at each SYSCLK cycle. The state of the CNT_UD input determines whether the counter increments or decrements. The GP counter uses the CNT_CFG.CUDINV bit to select the polarity. The CNT_DG pin can be used to gate the clock. The GP counter uses the CNT_CFG.CDGINV bit to select the polarity.

M/N Scaling

The GP counter provides programmable M/N frequency scaling of the CNT_UD and CNT_DG inputs. It supports frequency scaling only when the inputs are in quadrature encoded mode, with support for both incrementing and decrementing gray code, and with on-the-fly direction changing. The divided outputs are available on the GP counter output pins CNT_OUTA and CNT_OUTB.

To use M/N frequency scaling, set the CNT_CFG.DIVEN bit and program the M and N values in the CNT_MDIV and CNT_NDIV registers. With division enabled, the output frequency on the CNT_OUTA and CNT_OUTB pins is:

$f_{QOUT} = f_{QIN} \times (M/N)$

Where the frequency of the inputs on the CNT_UD and CNT_DG input pins is:

$f_{QIN}=1/t_{PERIOD}$

The ***M/N Scaling*** figure shows the M/N scaling implemented by the QEP dividers.

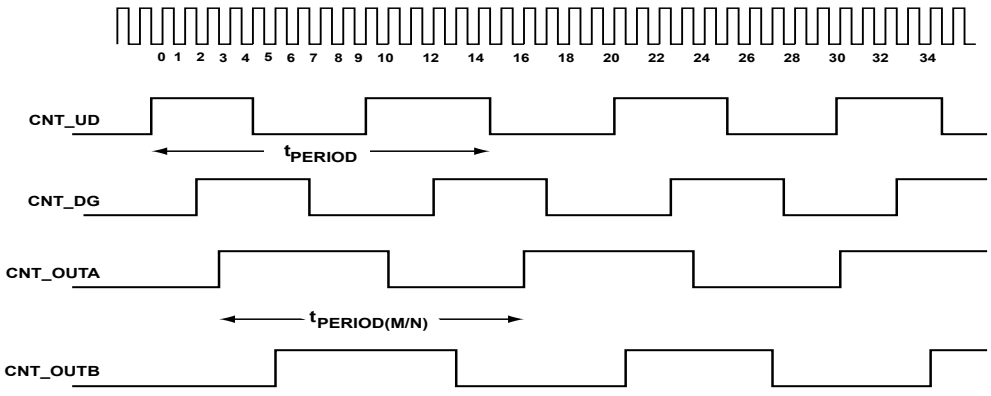

Figure 17-2: M/N Scaling

The divided outputs (CNT_OUTA and CNT_OUTB pins) start after 1 QEP input pulse period as shown in the figure (and 1 or 2 system clocks (SYSCLK) based on the implementation). If de-bouncing is enabled and the CNT_CFG.DIVNTV bit is not set, then the divided outputs start after 1 QEP pulse + the de-bouncing period (t_{FILTER}).

The ***M/N Scaling*** figure shows the full period scaled M/N (t_{PERIOD}/(M/N). However, the value for scaling is measured from the rising edge of either of the inputs (CNT_UD or CNT_DG) to the rising edge of the other input. This value gives the width of the QEP pulse and the value measured is scaled to create the M/N version. Each QEP pulse is continuously measured and the values are constantly updated to create the divided outputs. The QEP width seen on the output is not the exact M/N- scaled version of a measured QEP input width because the input frequency changes. There is not a one-to-one correspondence for the QEP measured-width and the actual QEP M/N output.

This difference is because the design constantly updates the measured values to the dividers. The output reflects a type of weighted-average of the multiple QEP pulses measured. The ***Scaled Outputs with Increasing Frequency at Inputs (Increasing Motor Speed)*** shows waveforms for an increasing input frequency.

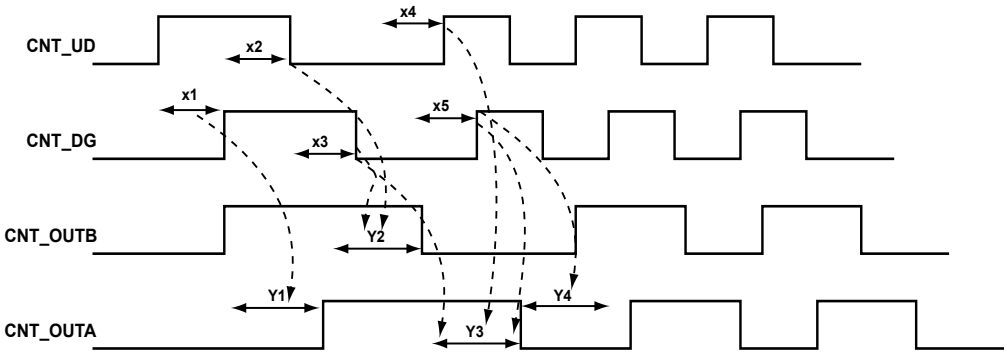

Figure 17-3: Scaled Outputs with Increasing Frequency at Inputs (Increasing Motor Speed)

In the ***Scaled Outputs with Increasing Frequency at Inputs*** figure X1, X2, X3, X4, X5 are the input QEP widths measured. (After X5 measurements, the widths remain constant. This period is also known as frequency stabilized) and Y1, Y2, Y3, Y4 are the QEP output widths (and all outputs keep to Y4 as the input frequency stabilized). Importantly,

- Y1 = X1/(M/N)
- Y2 = (Weighted Average of {X2,X3})/(M/N)
- Y3 = (Weighted Average of {X3,X4,X5})/(M/N)
- Y4 = X5/(M/N)

The weight applied for each measurement depends on when the measurement completes based on the outputs. It is difficult to predict the exact weight. Similar behavior occurs with the decreasing frequency of inputs.

Set the `CNT_CFG.DIVMODE` bit to avoid weighted averaging. Once the bit is set, the output values are:

- Y1 = X1/(M/N)
- Y2 = X2/(M/N)
- Y3 = X3/(M/N)
- Y4 = X5/(M/N)

The default behavior of the QEP dividers is weighted-averaging.

The maximum error in the input width measured by the QEP dividers is ± 1 SYSCLK cycles. The maximum error on the QEP divided outputs is ± N/M SYSCLK cycles.

NOTE: The count of output pulses created by this M/N scaling is not always M/N times the input pulses. The input pulse counts are scaled N/M without many errors for large values of M/N. For small values, there are errors. These errors can accumulate as the number of input pulses increases.

M/N Stop Detection

If there are no QEP input pulses after the last measured and translated QEP input pulse, then the QEP divider responds as if the input (and therefore, the motor) has stopped. Once the stop condition is detected, the QEP divider-outputs do not switch. A W1C status bit (`CNT_STAT.STP`) is set in the status register, and an interrupt is generated when the `CNT_IMSK.STP` bit is unmasked. Once the input QEP pulses restart, the QEP output pulses restart after a delay of 1 QEP pulse.

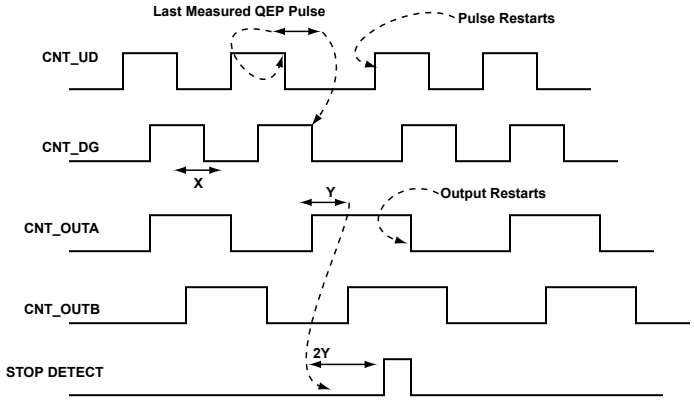

Figure 17-4: M/N Stop Detection

NOTE: The figure depicts an N/M ratio of 3/2. Y = X × N/M

If there are no QEP pulses seen during the period 2Y, then the QEP dividers assume that inputs have stopped. The QEP output does not toggle until a new input pulse is identified. Even if there are errors such as trembles or illegal Gray code that occur during this period, the stop condition is still detected. The error cases are not treated as a valid QEP pulse.

If the input width varies from one QEP pulse to the other more by more than 2Y, then a false stop is detected. Therefore, the maximum supported deceleration rate is from X to 2Y. This rate translates to a width-change of X to 2X × N/M between two QEP pulses. Using a worst case of N/M = 1, the maximum a pulse width can change is twice the size of the previous pulse. So, the allowable width of a QEP pulse following a 20 µs pulse is <40 µs (with N/M=1). Similarly, if there is a 1 ms QEP, the pulse that follows is within 2 ms. If not, a stop condition is detected at N/M=1.

The stop detection interrupt or status is also raised when a QEP pulse is wide and caused the internal counter to overflow. The internal counter-overflow occurs if the pulse is larger than 4,026,531,840 × M (or hex (0xF000_0000) × M) system clock cycles.

NOTE: If new QEP pulses are identified with a different direction after a STOP is detected, no direction-change interrupt occurs. (The dividers assume that the pulse is new start after the STOP detection). However, the QEP dividers start providing the output with the new direction.

M/N Error Condition

If the CNT_UD/CNT_DG does not follow the quadrature encoder mode, then the outputs continue to run in quadrature encoder mode using the QEP input width measured before the error condition. (The error is ignored). Although the output continues to run, the counter raises an illegal Gray code interrupt when the inputs are not in quadrature encode mode with CNT_CFG.CNTMODE set to QUAD_ENC mode.

To stop the QEP dividers on an error condition, disable the CNT_CFG.DIVEN bit to three-state the CNT_OUTA/CNT_OUTB divided outputs immediately.

M/N Restrictions

The following restrictions apply to the programmable M and N values.

a) M, N values cannot be changed on-the-fly (when the CNT_CFG.DIVEN bit =1).

b) $M<f_{SYSCLK}/f_{QIN(max)}$, where f_{QIN} is input frequency, f_{SYSCLK} is the frequency of the system clock.

c) N is greater than or equal to M ($n \geq M$)

d) N is greater than 1 ($N>1$)

e) M is greater than 1 ($M>1$)

If $M>f_{SYSCLK}/f_{QIN(max)}$, then an error status bit (CNT_STAT.MERR) is set and an interrupt is generated (if the interrupt enable bit (CNT_IMSK.MERR) is set). This sequence of events occurs each time the QEP input pulse frequency does not meet the requirement after the end of each QEP input pulse.

The ***Max M Values*** table shows the maximum M values for f_{SYSCLK} = 100 MHz.

Table 17-7: Max M Values

f_{QIN}(KHz)	M
6666.66	15
2000.0	50
333.33	300
166.67	600

GP Counter Programming Model

The following sections provide information for programming the interface.

GP Counter General Programming Flow

The following are general guidelines for configuring and enabling the GP counter.

1. Initialize (but do not enable) the GP Counter for the desired mode and settings through the CNT_CFG register.

2. Usually, events of interest are processed using interrupts rather than by polling status bits. In this case, clear all status bits and activate the interrupt generation requests with the CNT_IMSK register.
3. Configure interrupts at the system level to insure desired interrupt signaling to the system event controller.
4. If timing information is required, set up the relevant GP Timer in width capture mode.
5. Finally, enable interrupts and the GP Counter itself using the CNT_IMSK and CNT_CFG registers, respectively.

M/N Scaling Programming Guidelines

To use the QEP dividers, set both the CNT_CFG.EN and CNT_CFG.DIVEN bits. Note that the counter must be enabled at least two system clock (SYSCLK) cycles before enabling the rotary dividers (set CNT_CFG.EN at least 2 SYSCLK cycles before setting the CNT_CFG.DIVEN bit. To detect an illegal gray code error condition, program the CNT_CFG.CNTMODE for quadrature encode mode (QUAD_ENC).

The polarity bits (CNT_CFG.CDGINV and CNT_CFG.CUDINV) do not invert the QEP divider outputs, the QEP divider output polarity is the same as the input polarity. However the polarity bits cannot be changed on the fly when the rotary dividers are running (the polarity bits should not be changed when the CNT_CFG.DIVEN bit =1). If this occurs, it may be treated as an error or a direction change by the QEP dividers. Do not change the CNT_CFG bits on the fly when QEP dividers are enabled.

GP Counter Mode Configuration

The GP counter can use the CNT_ZM input pin to sense the zero marker output of a rotary device or to detect the pressing of a push button. There are four programming schemes, which are functional in all counter modes:

- Push-button mode
- Zero-marker-zeros-counter mode
- Zero-marker-error mode
- Zero-once mode

Configuring GP Counter Push-Button Operation

Use the following procedure to configure push-button operation:

1. Set CNT_IMSK.CZM to enable (unmask) the zero marker interrupt.
2. Select the active edge polarity through the CNT_CFG.CZMINV bit.
3. Proceed with any other desired configuration steps and enable the peripheral.

An active edge at the CNT_ZM input sets the CNT_IMSK.CZM bit.

Configuring Zero-Marker-Zeros-Counter Mode

The following provides information on configuring zero-marker-zeros-counter mode for the GP counter.

1. Set CNT_IMSK.CZMZ to enable CNT_CNTR. The zero marker interrupt zeroes the counter.
2. Set CNT_CFG.ZMZC to enable ZMZC mode.
3. Select the active edge polarity through the CNT_CFG.CZMINV bit.
4. Proceed with any other desired configuration steps and enable the peripheral.

This configuration causes an active level at the CNT_ZM pin to clear the CNT_CNTR register and keep it cleared until the CNT_ZM pin is deactivated. In addition, the CNT_STAT.CZMZ bit is set.

Configuring Zero-Marker-Error Mode

The GP counter uses this mode to detect discrepancies between counter-value and the zero marker output of certain rotary encoder devices.

1. Set the CNT_STAT.CZME bit to enable this mode.
2. Select the active edge of the CNT_ZM pin through the CNT_CFG.CZMINV bit.
3. Proceed with any other desired configuration steps and enable the peripheral.

When the GP counter detects an active edge at the CNT_ZM input pin, it compares the four LSBs of the CNT_CNTR register to zero. If they are not zero, the GP counter uses CNT_STAT.CZME bit to signal a mismatch.

Configuring Zero-Once Mode

The GP counter uses this mode to perform an initial reset of the counter-value when it detects an active zero marker. After that, the zero marker is ignored (the counter is no longer reset).

1. Set the CNT_CMD.W1ZMONCE bit to enable this mode.
2. Select the active edge of the CNT_ZM pin through the CNT_CFG.CZMINV bit.
3. Ensure that at least one of the following bits is enabled: CNT_IMSK.CZM, CNT_IMSK.CZME, CNT_IMSK.CZMZ.
4. Proceed with any other desired configuration steps and enable the peripheral.

The CNT_CNTR register and the CNT_CMD.W1ZMONCE bit are cleared on the next active edge of the CNT_ZM pin. Now the CNT_CMD.W1ZMONCE bit can be read to check whether the event has already occurred.

Configuring Boundary Auto-Extend Mode

In this mode, hardware modifies the boundary registers (CNT_MIN and CNT_MAX) whenever the CNT_CNTR value reaches either of them. The GP counter uses this mode to monitor the widest angle a thumb wheel even if the software did not generate interrupts.

1. Initialize CNT_CNTR with the desired value.
2. Set both CNT_MIN and CNT_MAX to this same value.

3. Configure the CNT_CFG.BNDMODE field for auto extend mode.
4. Proceed with any other desired configuration steps and enable the peripheral.

The CNT_MAX register is loaded with the current CNT_CNTR value when the latter increments beyond the CNT_MAX value. Similarly, the CNT_MIN register is loaded with the CNT_CNTR value when the latter decrements below the CNT_MIN value. The CNT_STAT.MAXC and CNT_STAT.MINC status bits are set when the CNT_CNTR value matches the respective boundary register value.

Configuring Boundary Capture Mode

In this mode, the CNT_CNTR value is latched into the CNT_MIN register at one detected edge of the CNT_ZM input pin, and latched into the CNT_MAX boundary register at the opposite edge.

1. To capture the CNT_ZM pin rising edge into CNT_MIN and the falling edge into CNT_MAX, program CNT_CFG.CZMINV for active high polarity. Conversely, to capture the CNT_ZM pin falling edge into CNT_MIN and the rising edge into CNT_MAX, program CNT_CFG.CZMINV for active low polarity.
2. Program the CNT_IMSK.MAXC and CNT_IMSK.MINC interrupt mask bits according to interrupt generation requirements.
3. Configure the CNT_CFG.BNDMODE field for boundary capture mode.
4. Proceed with any other desired configuration steps and enable the peripheral.

The CNT_STAT.MAXC and CNT_STAT.MINC status bits report the capture event, depending on how interrupt masks are configured.

Configuring Boundary Compare and Boundary Zero Modes

In these modes, the two boundary registers (CNT_MAX and CNT_MIN) are compared to the value in the CNT_CNTR register.

1. Program CNT_MAX and CNT_MIN registers with appropriate upper and lower range values.
2. Program the CNT_IMSK.MAXC and CNT_IMSK.MINC interrupt mask bits according to interrupt generation requirements.
3. Configure the CNT_CFG.BNDMODE field for boundary compare mode.
4. Proceed with any other desired configuration steps and enable the peripheral.

If after incrementing, CNT_CNTR = CNT_MAX, then the CNT_STAT.MAXC bit is set. Similarly if after decrementing, CNT_CNTR = CNT_MIN, then the CNT_STAT.MINC bit is set.

Additionally, for boundary zero mode, the counter-value in CNT_CNTR is set to zero. The CNT_STAT.MAXC and CNT_STAT.MINC bits are not set when software updates the CNT_MAX or CNT_MIN registers.

Configuring GP Counter Push-Button Operation

Use the following procedure to configure push-button operation:

1. Set CNT_IMSK.CZM to enable (unmask) the zero marker interrupt.
2. Select the active edge polarity through the CNT_CFG.CZMINV bit.
3. Proceed with any other desired configuration steps and enable the peripheral.

An active edge at the CNT_ZM input sets the CNT_IMSK.CZM bit.

GP Counter Programming Concepts

Using the features, operating modes, and event control for the GP counter to their greatest potential requires an understanding of some GP counter-related concepts. Some key aspects to consider are input noise filtering and capturing timing information.

CNT Input Noise Filtering

In all modes, the three input pins can be filtered to present clean signals to the GP counter logic. The GP counter uses the CNT_CFG.DEBEN bit to enable or disable this filtering. The ***Programmable Noise Filtering*** figure shows the filtering operation for the CNT_UD pin.

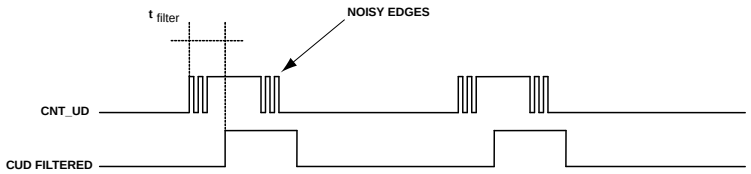

Figure 17-5: Programmable Noise Filtering

The CNT module implements the filtering mechanism using counters for each GP counter pin, where each counter is initialized from the CNT_DEBNCE.DPRESCALE field. When a transition is detected on a pin, the corresponding counter starts counting up to the programmed number of SYSCLK cycles. The state of the pin is latched after time t_{filter} and passed on to the GP counter logic.

The following formula determines the time t_{filter}, given SYSCLK and the CNT_DEBNCE.DPRESCALE value, where lower values of CNT_DEBNCE.DPRESCALE result in shorter debounce delays:

$$t_{filter} = 128 \times (2^{DPRESCALE} \times SYSCLK)$$

Capturing Counter Interval and CNT_CNTR Read Timing

When the count speed is low, it is often useful to capture the time elapsed since the last count event. Program the TIMER_TMR[n]_CFG register of the associated GP timer in a width capture mode with the following bit settings.

- TIMER_TMR[n]_CFG.PULSEHI = 0
- TIMER_TMR[n]_CFG.TMODE = b#1011

- TIMER_TMR[n]_CFG.TINSEL = 1

The ***Capture Intervals*** figure shows and the following list describe the operation of the GP counter and the GP timer in this mode.

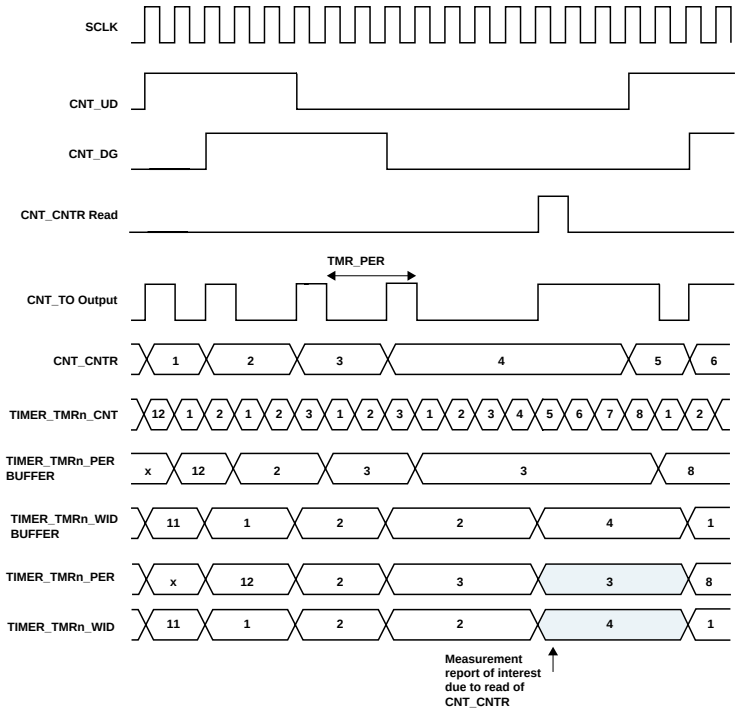

Figure 17-6: Capture Intervals

1. The CNT_TO signal generates a pulse every time a count event occurs. In addition, when the processor reads the CNT_CNTR register, the CNT_TO signal presents a pulse which is extended (high) until the next count event.
2. The GP timer updates its TIMER_TMR[n]_PER register with the period (measured from falling edge to falling edge, because TIMER_TMR[n]_CFG.PULSEHI = 0) of the CNT_TO signal.
3. The TIMER_TMR[n]_WID register is updated with the pulse width (the portion where CNT_TO is low, again because TIMER_TMR[n]_CFG.PULSEHI = 0).
4. Both registers are updated at every rising edge of the CNT_TO signal (because TIMER_TMR[n]_CFG.TMODE= b#011).

The TIMER_TMR[n]_PER register contains the period between the last two count events. The TIMER_TMR[n]_WID register contains the time since the last count event and the read of the CNT_CNTR register, both measured in SYSCLK cycles.

Read the CNT_CNTR register to latch the two time measurements, providing a coherent triplet of information to calculate speed and position.

NOTE: Speed restrictions apply to the use of the CNT_TO signal. Therefore, programs must not operate at count event rates that are high. For instance, if CNT_CNTR is incremented or decremented every SYSCLK cycle (timed direction mode), the CNT_TO signal is not valid.

Capturing Time Interval Between Successive Counter Events

When the required timing information is the interval between successive count events, program the associated timer in a width capture mode. Set the TIMER_TMR[n]_CFG bit of TIMER_TMR[n]_CFG.PULSEHI = 1, TIMER_TMR[n]_CFG.TMODE = b#1010 and TIMER_TMR[n]_CFG.TINSEL = 1. Typically, this information is sufficient if the speed of GP counter events does not to reach low values.

The ***Period Register Timing*** figure shows the operation of the GP counter and the GP timer in this mode.

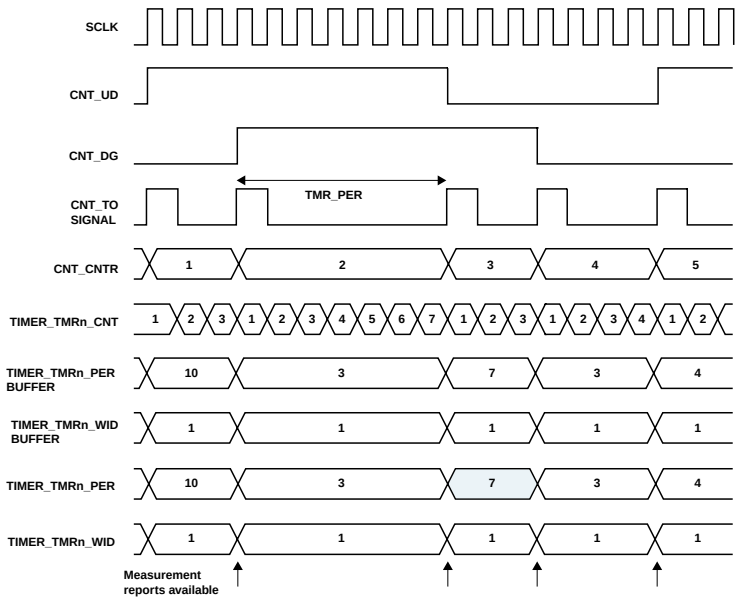

Figure 17-7: Period Register Timing

The CNT_TO signal generates a pulse every time a count event occurs. The GP timer updates its TIMER_TMR[n]_PER register with the period (measured from rising edge to rising edge) of the CNT_TO signal. The TIMER_TMR[n]_PER register is updated at every rising edge of the CNT_TO signal and contains the number of SYSCLK cycles that have elapsed since the previous rising edge.

Incidentally, the TIMER_TMR[n]_WID register is also updated at the same time, but is generally of no interest in this mode of operation. If no reads of the CNT_CNTR register occur between counter events, the TIMER_TMR[n]_WID register only contains the width of the CNT_TO pulse. If a read of CNT_CNTR has occurred between events, the TIMER_TMR[n]_WID register contains the time between the read of CNT_CNTR and the next event.

This mode can also be used with TIMER_TMR[n]_CFG.PULSEHI = 0. In this case, the period of CNT_TO is measured between falling edges. It results in the same values as in the previous case, only the latching occurs one SYSCLK cycle later.

GP Counter Event Control

11 events can be signaled to the processor using status information and optional interrupt requests. The GP counter uses the respective bits in the CNT_IMSK register to enable the interrupts. It uses dedicated bits in the CNT_STAT

register to report events. When an interrupt from the GP counter is acknowledged, the application software is responsible for correct interpretation of the events. It is recommended to logically AND the content of the CNT_IMSK and CNT_STAT registers to identify pending interrupts.

Perform a write-one-to-clear (W1C) operation to the CNT_STAT register to clear the interrupt requests. Hardware does not clear the status bits automatically, unless the counter module is disabled.

The following sections describe the events associated with the GP counter.

Illegal Gray and Binary Code Events

When illegal transitions occur in quadrature encoder or binary encoder modes, the CNT_STAT.IC bit is set. If enabled by the CNT_STAT.IC bit, the counter module generates an interrupt request. Set the CNT_STAT.IC bit only in the quadrature encoder or binary encoder modes.

Up/Down Count Events

The GP counter uses the CNT_STAT.UC bit to indicate whether the counter has been incremented. Similarly, the CNT_STAT.DC bit reports decrements. The two events are independent. For instance, if the counter increments by one and then decrements by two, both bits remain set, even though the resulting counter-value shows a decrement by one.

In up/down counter mode, hardware can detect simultaneous active edges on the CNT_UD and CNT_DG inputs. In that case, the CNT_CNTR remains unchanged, but both the CNT_STAT.UC and CNT_STAT.DC bits are set. Interrupt requests for these events can be enabled through the CNT_IMSK.UC and CNT_IMSK.DC bits. Use this feature carefully when the counter is clocked at high rates. This suggestion is especially critical when the counter operates in DIR_TMR mode, as interrupts are generated every SYSCLK cycle.

These events can also be used for more push buttons, when GP counter features are unnecessary. When up/down counter mode is enabled, the GP counter can use these count events to report interrupts from push buttons that connect to the CNT_UD and CNT_DG inputs.

Zero-Count Events

The CNT_STAT.CZERO status bit indicates that the CNT_CNTR has reached a value equal to 0x0000 0000 after an increment or decrement. This bit is not set when the counter value is set to zero by a write to CNT_CNTR or by setting the CNT_CMD.W1LCNTZERO bit. If enabled by the CNT_IMSK.CZERO bit, the GP counter module generates an interrupt request.

Overflow Events

There are two status bits that indicate whether the signed counter-register has overflowed from a positive to a negative value or conversely. The CNT_STAT.COV31 bit reports that the 32-bit CNT_CNTR register has either incremented from 0x7FFF FFFF to 0x8000 0000, or decremented from 0x8000 0000 to 0x7FFF FFFF.

If enabled by the CNT_IMSK.COV31 bit, an interrupt request is generated. Similarly, in applications where only the lower 16 bits of the counter are of interest, the CNT_STAT.COV15 status bit reports counter transitions from

0xXXXX 7FFF to 0xXXXX 8000, or from 0xXXXX 8000 to 0xXXXX 7FFF. If enabled by the CNT_IMSK.COV15 bit, an interrupt request is generated.

Boundary Match Events

The CNT_STAT.MINC and CNT_STAT.MAXC status bits report boundary events as described in Configuring Boundary Capture Mode. These bits are not set if the software updates the CNT_CNTR, CNT_MAX, or CNT_MIN registers or writes to the CNT_CMD register. The CNT_IMSK.MINC and CNT_IMSK.MAXC bits enable interrupt generation on boundary events.

Zero Marker Events

The CNT_STAT.CZM, CNT_STAT.CZME, and CNT_STAT.CZMZ bits are associated with zero marker events, as described in Configuring GP Counter Push-Button Operation. Each of these events can optionally generate an interrupt request, when enabled by the corresponding CNT_IMSK.CZM, CNT_IMSK.CZME and CNT_IMSK.CZMZ bits.

ADSP-CM40x CNT Register Descriptions

CNT (CNT) contains the following registers.

Table 17-8: ADSP-CM40x CNT Register List

Name	Description
CNT_CFG	Configuration Register
CNT_CMD	Command Register
CNT_CNTR	Counter Register
CNT_DEBNCE	Debounce Register
CNT_IMSK	Interrupt Mask Register
CNT_MAX	Maximum Count Register
CNT_MDIV	M Value for Divider
CNT_MIN	Minimum Count Register
CNT_NDIV	N Value for Divider
CNT_STAT	Status Register

Configuration Register

The CNT_CFG register configures counter modes, configures input pins, and enable the CNT.

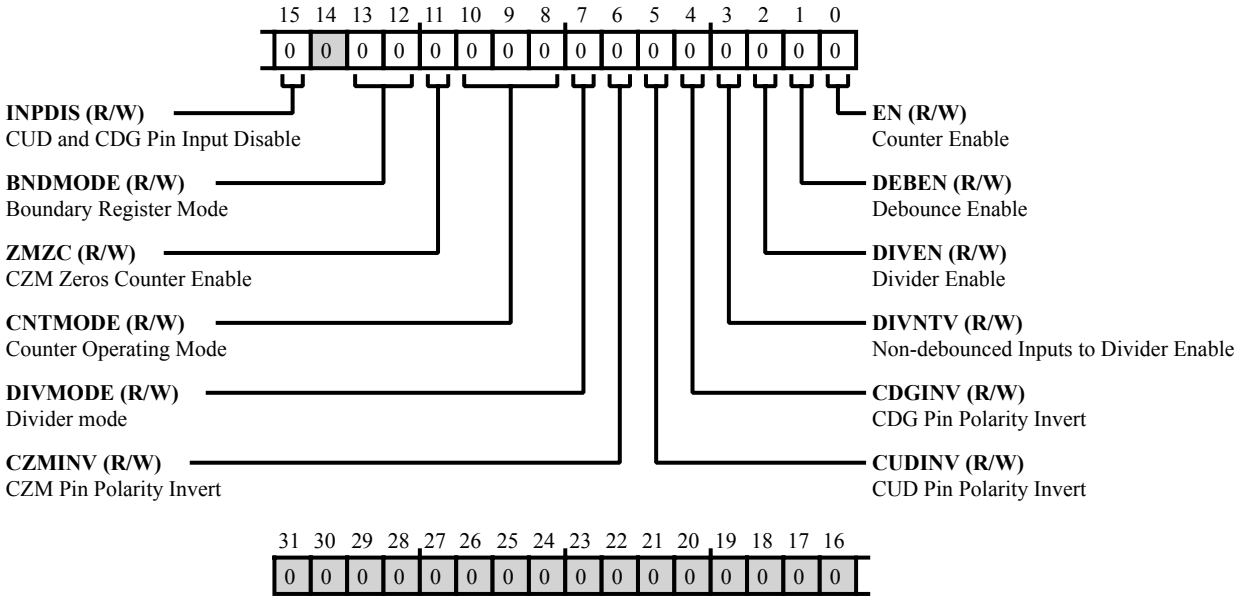

Figure 17-8: CNT_CFG Register Diagram

Table 17-9: CNT_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	INPDIS	CUD and CDG Pin Input Disable. The CNT_CFG.INPDIS disables or enables the CNT_UD input pin and the CNT_DG pin.	
		0	Enable
		1	Pin Input Disable
13:12 (R/W)	BNDMODE	Boundary Register Mode. The CNT_CFG.BNDMODE selects the mode for the CNT_MIN and CNT_MAX boundary registers.	
		0	BND_COMP Boundary compare mode
		1	BND_ZERO Boundary zero mode
		2	BND_CAPT Boundary capture mode
		3	BND_AEXT Boundary auto-extend mode

Table 17-9: CNT_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	ZMZC	CZM Zeros Counter Enable. The CNT_CFG.ZMZC enables or disables level sensitive - Active CNT_ZM pin operation to zero the CNT_CNTR.	
		0	Disable
		1	Enable
10:8 (R/W)	CNTMODE	Counter Operating Mode. The CNT_CFG.CNTMODE selects the operating mode for the CNT_UD input pin and the CNT_DG pin.	
		0	QUAD_ENC Quadrature encoder mode
		1	BIN_ENC Binary encoder mode
		2	UD_CNT Rotary counter mode
		4	DIR_CNT Direction counter mode
		5	DIR_TMR Direction timer mode
7 (R/W)	DIVMODE	Divider mode.	
		0	Weighted Average Division
		1	Non Weighted Division
6 (R/W)	CZMINV	CZM Pin Polarity Invert. The CNT_CFG.CZMINV selects the polarity for the CNT_ZM pin. This polarity must be configured before the counter is enabled. It must not change on-the-fly while the counter is enabled.	
		0	Active High, Rising Edge
		1	Active Low, Falling Edge
5 (R/W)	CUDINV	CUD Pin Polarity Invert. The CNT_CFG.CUDINV selects the polarity for the CNT_UD pin. This polarity must be configured before the counter is enabled. It must not change on-the-fly while the counter is enabled.	
		0	Active High, Rising Edge
		1	Active Low, Falling Edge
4 (R/W)	CDGINV	CDG Pin Polarity Invert. The CNT_CFG.CDGINV selects the polarity for the CNT_DG pin. This polarity must be configured before the counter is enabled. It must not change on-the-fly while the counter is enabled.	
		0	Active High, Rising Edge
		1	Active Low, Falling Edge

Table 17-9: CNT_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	DIVNTV	Non-debounced Inputs to Divider Enable. The CNT_CFG.DIVNTV bit enables non-debounced inputs to the divider.	
		0	Disable
		1	Enable
2 (R/W)	DIVEN	Divider Enable. The CNT_CFG.DIVEN bit enables the divider.	
		0	Disable
		1	Enable
1 (R/W)	DEBEN	Debounce Enable. The CNT_CFG.DEBEN enables or disables CNT input debounce filtering operation selected with the CNT_DEBNCE register.	
		0	Disable
		1	Enable
0 (R/W)	EN	Counter Enable. The CNT_CFG.EN enables or disables CNT operation.	
		0	Counter Disable
		1	Counter Enable

Command Register

The CNT_CMD register configures the CNT, enabling operations such as zeroing a counter register and copying or swapping boundary registers. These actions are taken by setting the appropriate bit.

Read operations from this register do not return meaningful values, with the exception of the CNT_CMD.W1ZMONCE bit, where a set bit indicates that the bit has been set by software before, but a zero marker event has not yet been detected on the CNT_ZM pin yet. For more information, see the CNT functional description.

The CNT_CNTR, CNT_MIN, and CNT_MAX registers can be initialized to zero by setting the CNT_CMD.W1LCNTZERO, CNT_CMD.W1LMINZERO, and CNT_CMD.W1LMAXZERO bits. In addition to clearing registers, CNT_CMD permits modifying the CNT_MIN and CNT_MAX boundary registers in a number of ways. The current counter value in CNT_CNTR can be captured and loaded into either of the two boundary registers to create new boundary limits. This operation is performed by setting the CNT_CMD.W1LMAXCNT and CNT_CMD.W1LMINCNT bits. Alternatively, the counter can be loaded from CNT_MAX or CNT_MIN using the CNT_CMD.W1LCNTMAX and CNT_CMD.W1LCNTMIN bits. It is also possible to transfer the current CNT_MAX value into CNT_MIN (or vice versa) through the CNT_CMD.W1LMINMAX and CNT_CMD.W1LMAXMIN bits.

Another counter operation is the ability to only have the zero marker clear the CNT_CNTR register once. For more information, see the CNT functional description.

It is possible for multiple actions to be performed simultaneously by setting multiple bits in the CNT_CMD register, but there are restrictions. The bits associated with each command have been grouped together such that all bits that involve a write to the CNT_CNTR, CNT_MAX, or CNT_MIN are located within bits 4-bit groups of the CNT_CMD register.

Note that a maximum of three commands can be issued at any one time, excluding the CNT_CMD.W1ZMONCE command. Also note that CNT_CMD.W1LCNTMIN, CNT_CMD.W1LCNTMAX, and CNT_CMD.W1LCNTZERO bits have to be used exclusively. Never set more than one of them at the same time.

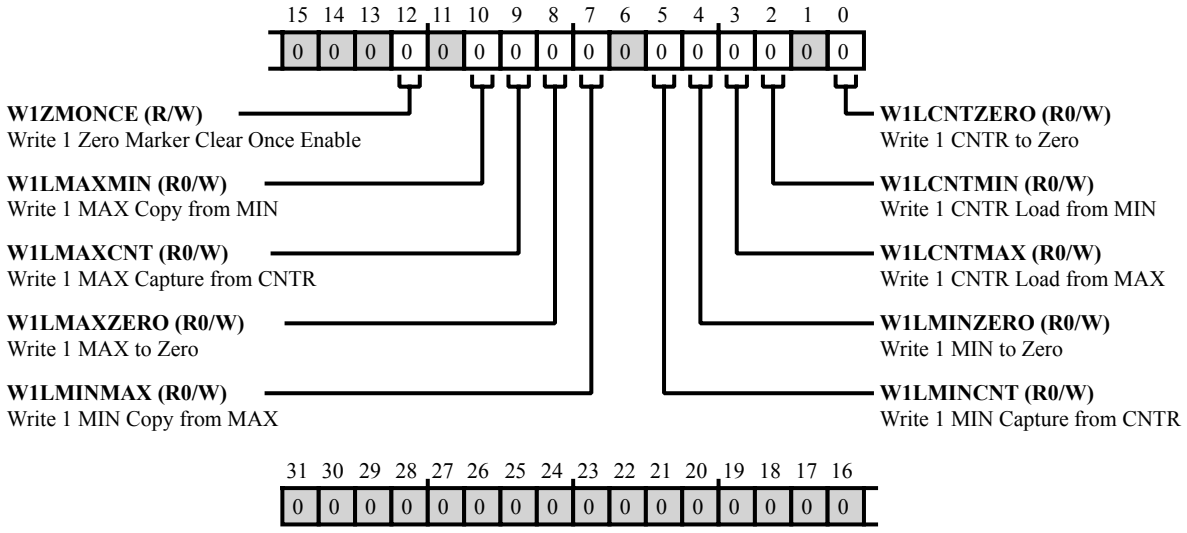

Figure 17-9: CNT_CMD Register Diagram

Table 17-10: CNT_CMD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W)	W1ZMONCE	Write 1 Zero Marker Clear Once Enable. The CNT_CMD.W1ZMONCE enables a single zero marker clear of CNT_CNTR. Reading a 1 in this bit indicates that the bit has been set by software before, but no zero marker event has been detected on the CNT_ZM pin yet.
10 (R0/W)	W1LMAXMIN	Write 1 MAX Copy from MIN. The CNT_CMD.W1LMAXMIN bit transfers the current CNT_MIN register value into CNT_MAX register.
9 (R0/W)	W1LMAXCNT	Write 1 MAX Capture from CNTR. The CNT_CMD.W1LMAXCNT bit loads the current value in the CNT_CNTR register into the CNT_MAX register to create a new boundary limit.
8 (R0/W)	W1LMAXZERO	Write 1 MAX to Zero. Writing a 1 to the CNT_CMD.W1LMAXZERO bit clears the CNT_MAX register.
7 (R0/W)	W1LMINMAX	Write 1 MIN Copy from MAX. The CNT_CMD.W1LMINMAX bit transfers the current CNT_MAX register value into CNT_MIN register.
5 (R0/W)	W1LMINCNT	Write 1 MIN Capture from CNTR. The CNT_CMD.W1LMINCNT bit loads the current value in the CNT_CNTR register into the CNT_MIN register to create a new boundary limit.
4 (R0/W)	W1LMINZERO	Write 1 MIN to Zero. Writing a 1 to the CNT_CMD.W1LMINZERO bit clears the CNT_MIN register.
3 (R0/W)	W1LCNTMAX	Write 1 CNTR Load from MAX. The CNT_CMD.W1LCNTMAX bit loads the current value in the CNT_MAX register into the CNT_CNTR register to create a new boundary limit.
2 (R0/W)	W1LCNTMIN	Write 1 CNTR Load from MIN. The CNT_CMD.W1LCNTMIN bit loads the current value in the CNT_MIN register into the CNT_CNTR register to create a new boundary limit.
0 (R0/W)	W1LCNTZERO	Write 1 CNTR to Zero. Writing a 1 to the CNT_CMD.W1LCNTZERO bit clears the CNT_CNTR register.

Counter Register

The CNT_CNTR register holds the 32-bit, twos-complement count value. It can be read and written at any time. Hardware ensures that reads and write are atomic, by providing respective shadow registers. This register can be accessed with either 32-bit or 16-bit operations. This allows use of the CNTas a 16-bit counter if sufficient for the application.

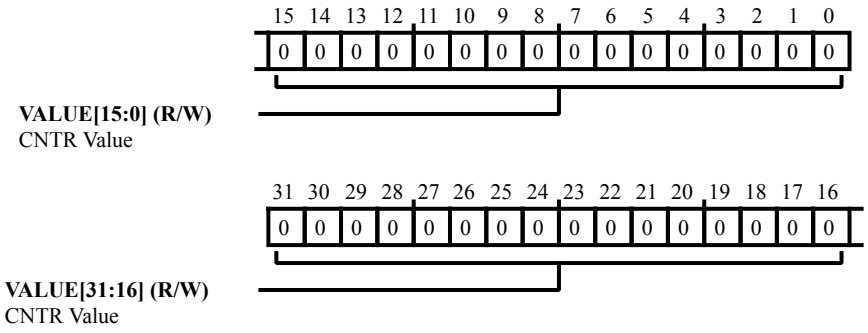

Figure 17-10: CNT_CNTR Register Diagram

Table 17-11: CNT_CNTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	CNTR Value. The CNT_CNTR.VALUE bit field holds the 32-bit, twos-complement count value.

Debounce Register

The CNT_DEBNCE register selects the noise filtering characteristic of the three input pins according to the formula:

t_{filter} = 128 x ($2^{DPRESCALE}$/ SYSCLK)

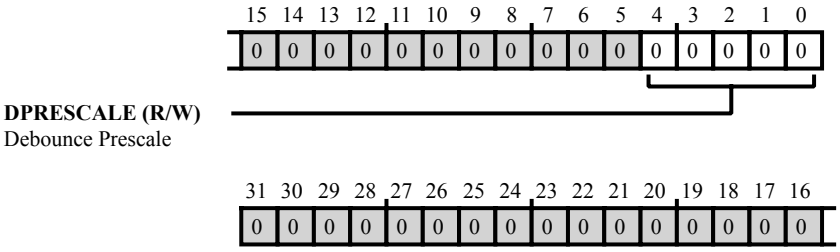

Figure 17-11: CNT_DEBNCE Register Diagram

Table 17-12: CNT_DEBNCE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4:0 (R/W)	DPRESCALE	Debounce Prescale. The CNT_DEBNCE.DPRESCALE selects the desired number of input filtering cycles (and resulting input debounce time) in multiples of SYSCLK.	
		0	1x cycles = 128 SYSCLK cycles
		1	2x cycles
		2	4x cycles
		3	8x cycles
		4	16x cycles
		5	32x cycles
		6	64x cycles
		7	128x cycles
		8	256x cycles
		9	512x cycles
		10	1024x cycles
		11	2048x cycles
		12	4096x cycles
		13	8192x cycles
		14	16384x cycles
		15	32768x cycles
		16	65536x cycles

Table 17-12: CNT_DEBNCE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		17	131072x cycles
		18	Reserved from this value 10010 - 11111: Reserved
		31	Reserved till this value

Interrupt Mask Register

The CNT_IMSK register supports enabling (unmasking) interrupt request generation from each of the CNT events.

All bits in CNT_IMSK either disable/mask an interrupt (if bit cleared) or enable/unmask an interrupt (if bit set).

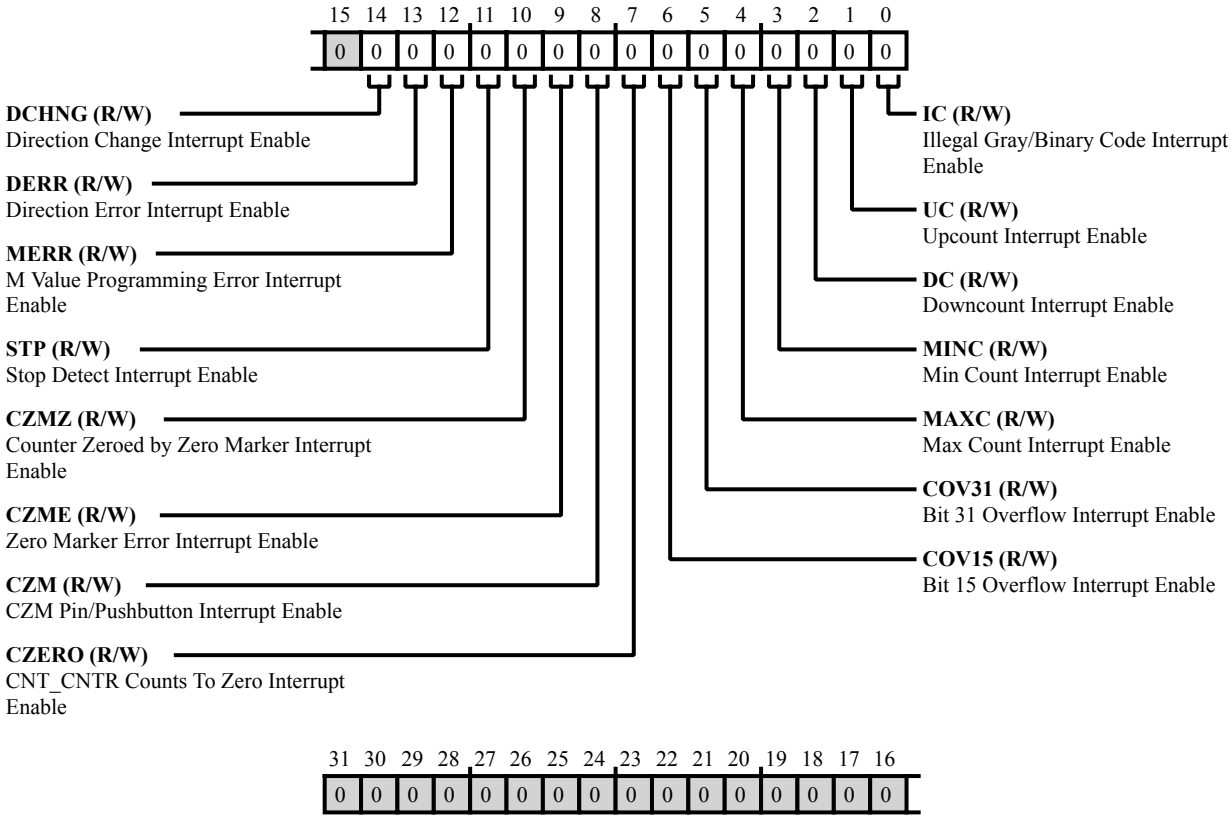

Figure 17-12: CNT_IMSK Register Diagram

Table 17-13: CNT_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	DCHNG	Direction Change Interrupt Enable. The CNT_IMSK.DCHNG bit enables (unmasks) the direction change interrupt.	
		0	Mask Interrupt
		1	Unmask Interrupt
13 (R/W)	DERR	Direction Error Interrupt Enable.	
		0	Mask Interrupt
		1	Unmask Interrupt

Table 17-13: CNT_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	MERR	M Value Programming Error Interrupt Enable. The CNT_IMSK.MERR bit enables (unmasks) the M value programming error interrupt.	
		0	Mask Interrupt
		1	Unmask Interrupt
11 (R/W)	STP	Stop Detect Interrupt Enable. The CNT_IMSK.STP bit enables (unmasks) the stop detect interrupt.	
		0	Mask Interrupt
		1	Unmask Interrupt
10 (R/W)	CZMZ	Counter Zeroed by Zero Marker Interrupt Enable. The CNT_IMSK.CZMZ bit enables (unmasks) the counter zeroed by zero marker interrupt.	
		0	Mask Interrupt
		1	Unmask Interrupt
9 (R/W)	CZME	Zero Marker Error Interrupt Enable. The CNT_IMSK.CZME bit enables (unmasks) the zero marker error interrupt.	
		0	Mask Interrupt
		1	Unmask Interrupt
8 (R/W)	CZM	CZM Pin/Pushbutton Interrupt Enable. The CNT_IMSK.CZM bit enables (unmasks) the CZM pin/pushbutton interrupt.	
		0	Mask Interrupt
		1	Unmask Interrupt
7 (R/W)	CZERO	CNT_CNTR Counts To Zero Interrupt Enable. The CNT_IMSK.CZERO bit enables (unmasks) the counts to zero interrupt.	
		0	Mask Interrupt
		1	Unmask Interrupt
6 (R/W)	COV15	Bit 15 Overflow Interrupt Enable. The CNT_IMSK.COV15 bit enables (unmasks) the bit 15 overflow interrupt.	
		0	Mask Interrupt
		1	Unmask Interrupt

Table 17-13: CNT_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	COV31	Bit 31 Overflow Interrupt Enable. The CNT_IMSK.COV31 bit enables (unmasks) the bit 31 overflow interrupt.	
		0	Mask Interrupt
		1	Unmask Interrupt
4 (R/W)	MAXC	Max Count Interrupt Enable. The CNT_IMSK.MAXC bit enables (unmasks) the max count interrupt.	
		0	Mask Interrupt
		1	Unmask Interrupt
3 (R/W)	MINC	Min Count Interrupt Enable. The CNT_IMSK.MINC bit enables (unmasks) the min count interrupt.	
		0	Mask Interrupt
		1	Unmask Interrupt
2 (R/W)	DC	Downcount Interrupt Enable. The CNT_IMSK.DC bit enables (unmasks) the down count interrupt.	
		0	Mask Interrupt
		1	Unmask Interrupt
1 (R/W)	UC	Upcount Interrupt Enable. The CNT_IMSK.UC bit enables (unmasks) the up count interrupt.	
		0	Mask Interrupt
		1	Unmask Interrupt
0 (R/W)	IC	Illegal Gray/Binary Code Interrupt Enable. The CNT_IMSK.IC bit enables (unmasks) the illegal Gray/Binary Code interrupt and should only be used in these modes.	
		0	Mask Interrupt
		1	Unmask Interrupt

Maximum Count Register

The CNT_MAX register holds the 32-bit, twos-complement, higher boundary value. It can be read and written at any time. Hardware ensures that reads and write are atomic, by providing respective shadow registers. This register can be accessed with either 32-bit or 16-bit operations. This allows for using the CNT as a 16-bit counter if sufficient for the application.

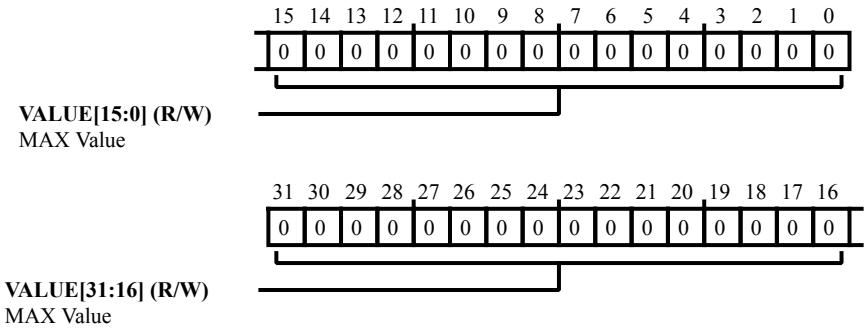

Figure 17-13: CNT_MAX Register Diagram

Table 17-14: CNT_MAX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MAX Value. The CNT_MAX.VALUE bit field holds the 32-bit, twos-complement, higher boundary value.

M Value for Divider

The CNT_MDIV register determines the M value for the M/N division of the QEP inputs that are provided on the CNT_UD/CNT_DG pins.

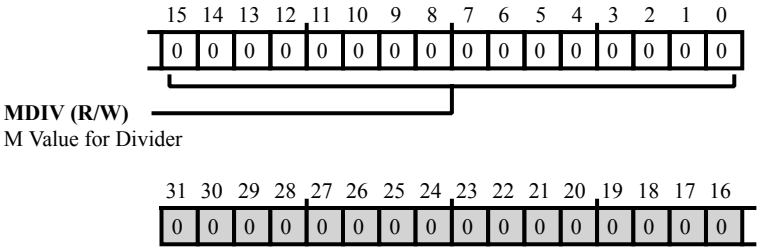

Figure 17-14: CNT_MDIV Register Diagram

Table 17-15: CNT_MDIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MDIV	M Value for Divider. The CNT_MDIV.MDIV bit field determines the M value for the M/N division of the QEP inputs that are provided on the CNT_UD/CNT_DG pins.

Minimum Count Register

The CNT_MIN register holds the 32-bit, twos-complement, lower boundary value. It can be read and written at any time. Hardware ensures that reads and write are atomic, by providing respective shadow registers. This register can be accessed with either 32-bit or 16-bit operations. This allows for using the CNT as a 16-bit counter if sufficient for the application.

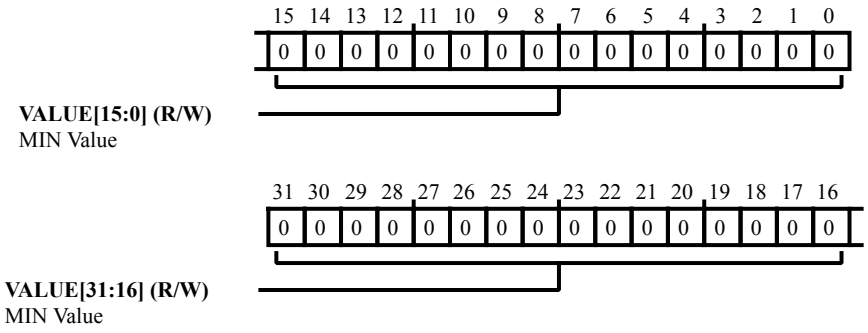

Figure 17-15: CNT_MIN Register Diagram

Table 17-16: CNT_MIN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MIN Value. The CNT_MIN.VALUE bit field holds the 32-bit, twos-complement, lower boundary value.

N Value for Divider

The CNT_NDIV register determines the N value for the M/N division of the QEP inputs that are provided on the CNT_UD/CNT_DG pins.

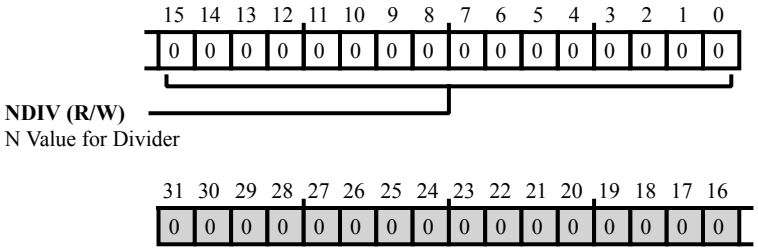

Figure 17-16: CNT_NDIV Register Diagram

Table 17-17: CNT_NDIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	NDIV	N Value for Divider. The CNT_NDIV.NDIV bit field determines the N value for the M/N division of the QEP inputs that are provided on the CNT_UD/CNT_DG pins.

Status Register

The CNT_STAT register provides status information for each of the CNT events as configured in the CNT_IMSK register. When a CNT event is detected, the corresponding bit in this register is set. It remains set until either software writes a 1 to the bit (write-1-to-clear) or the CNT is disabled.

All bits in CNT_STAT indicate either no interrupt pending (if bit cleared) or an interrupt pending (if bit set).

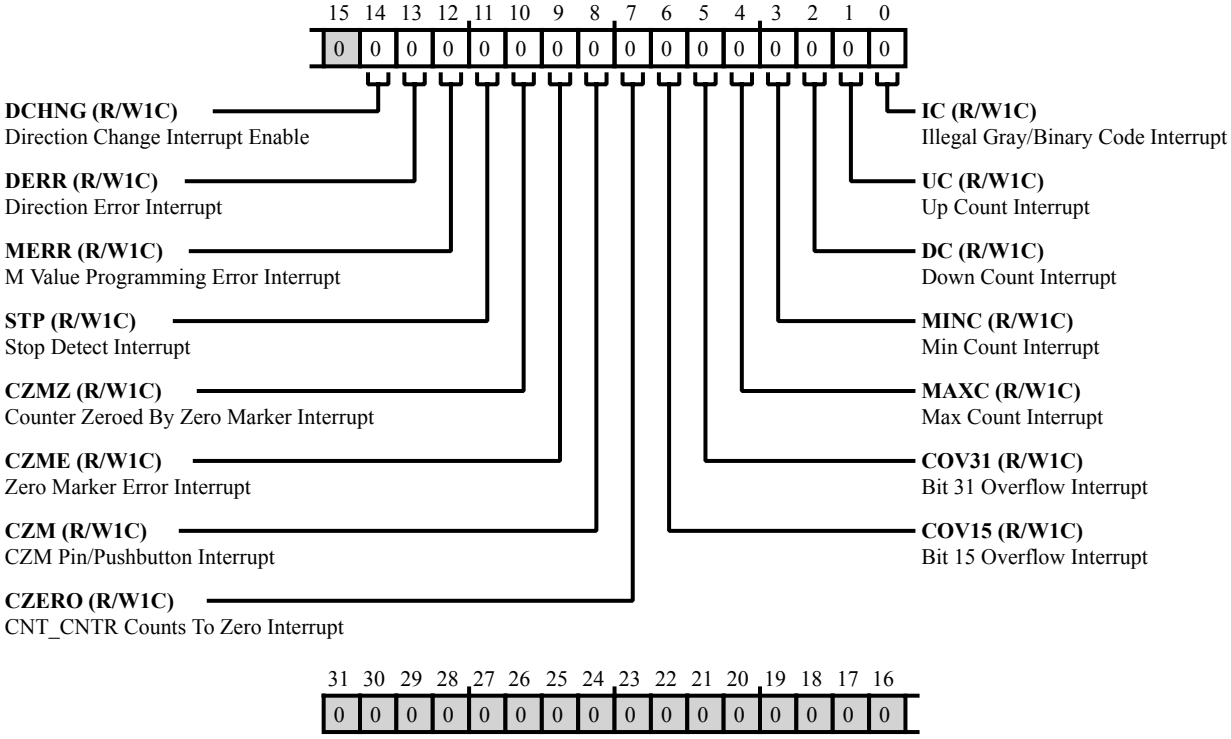

Figure 17-17: CNT_STAT Register Diagram

Table 17-18: CNT_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W1C)	DCHNG	Direction Change Interrupt Enable. The CNT_STAT.DCHNG bit is set if the direction change is valid. Direction change status and interrupt are generated only if QEP dividers are enabled.
13 (R/W1C)	DERR	Direction Error Interrupt.
12 (R/W1C)	MERR	M Value Programming Error Interrupt. The CNT_STAT.MERR bit indicates a M value programming error. This interrupt is generated when a M value greater than SYSCLK frequency/Q_{IN} frequency is programmed.

Table 17-18: CNT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W1C)	STP	Stop Detect Interrupt. The CNT_STAT.STP bit indicates a stop detect error. This interrupt is generated if the QEP pulse is wider than 2 X N/M. The internal counter overflow also causes STOP condition and it occurs if pulse was larger than 4,026,531,840 (0xF000_0000) M) system clock cycles.	
10 (R/W1C)	CZMZ	Counter Zeroed By Zero Marker Interrupt. The CNT_STAT.CZMZ bit indicates a Zero Marker error. If the CNT_CFG.ZMZC bit =1, this interrupt is generated when the CZMII latch reports a significant edge on the CZM input. Once cleared by software the CNT_STAT.CZM bit is not set again when the CZM input remains active without pulsing.	
		0	No error
		1	Error occurred
9 (R/W1C)	CZME	Zero Marker Error Interrupt. The CNT_STAT.CZME bit behaves similarly to the CNT_STAT.CZM bit, with the exception that CNT_STAT.CZME is not set on the CZM edge when the lower four bits of the CNT_CNTR are not zero. In many applications this indicates an error condition, as the zero marker might be out of sync with the counter.	
		0	No error
		1	Error occurred
8 (R/W1C)	CZM	CZM Pin/Pushbutton Interrupt. The CNT_STAT.CZM bit indicates a CZM pin/pushbutton error. This interrupt is generated when a significant edge is seen on the CZM pin, regardless what mode the counter is operating in. This is often used to sense push buttons (especially with the debouncing circuit enabled).	
		0	No error
		1	Error occurred
7 (R/W1C)	CZERO	CNT_CNTR Counts To Zero Interrupt. The CNT_STAT.CZERO bit indicates a counts to zero error. This error is generated when the CNT_CNTR register has incremented or decremented toward 0x0000.0000. The latch is not set when software writes to the CNT_CNTR register directly or when the counter is zeroed by writes to the CNT_CMD register.	
		0	No error
		1	Error occurred

Table 17-18: CNT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1C)	COV15	Bit 15 Overflow Interrupt. The CNT_STAT.COV15 bit indicates a bit 15 overflow error. This error is generated when the 16-bit 2s-complement CNT_CNTR register has incremented from 0xxxxx.7FFF to 0xxxxx.8000 or decremented from 0xxxxx.8000 to 0xxxxx.7FFF.	
		0	No error
		1	Error occurred
5 (R/W1C)	COV31	Bit 31 Overflow Interrupt. The CNT_STAT.COV31 bit indicates a bit 31 overflow error. This error is generated when the 32-bit 2s-complement CNT_CNTR register has incremented from 0x7FFF.FFFF to 0x8000.0000 or decremented from 0x8000.0000 to 0x7FFF.FFFF.	
		0	No error
		1	Error occurred
4 (R/W1C)	MAXC	Max Count Interrupt. The CNT_STAT.MAXC bit indicates a max count error. This interrupt is used in boundary compare (BND_COMP) mode. If after incrementing the CNT_CNTR register equals CNT_MAX, the CNT_STAT.MAXC bit is set.	
		0	No error
		1	Error occurred
3 (R/W1C)	MINC	Min Count Interrupt. The CNT_STAT.MINC bit indicates a min count error. This interrupt is used in boundary compare (BND_COMP) mode. If after decrementing the CNT_CNTR register equals CNT_MIN, the CNT_STAT.MINC bit is set.	
		0	No error
		1	Error occurred
2 (R/W1C)	DC	Down Count Interrupt. The CNT_STAT.DC bit indicates a down count error. This interrupt is generated when the CNT_CNTR register decrements.	
		0	No error
		1	Error occurred
1 (R/W1C)	UC	Up Count Interrupt. The CNT_STAT.UC bit indicates an up count interrupt. This interrupt is generated when the CNT_CNTR register increments.	

Table 17-18: CNT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W1C)	IC	Illegal Gray/Binary Code Interrupt. The CNT_STAT.IC bit indicates a illegal Gray/Binary Code interrupt and should only be used in these modes. In normal operation those codes can increment or decrement the CNT_CNTR register by one at a time. If the sensed inputs instruct the counter to increment or decrement by two, the CNT_STAT.IC bit is set. Hardware sets the CNT_STAT.IC bit in QUAD_ENC and BIN_ENC encoder modes only.	
		0	No error
		1	Error occurred

18 Pulse-Width Modulator (PWM)

The pulse-width modulator (PWM) module is a flexible and programmable waveform generator. With minimal CPU intervention, the PWM peripheral can generate complex waveforms for:

- Motor control
- Pulse-coded modulation (PCM)
- Digital-to-analog conversion (DAC)
- Power switching
- Power conversion.

The PWM module has four PWM pairs capable of 3-phase PWM generation for source inverters for AC induction and DC brushless motors.

PWM Features

Each PWM generation unit features:

- 16-bit center-based PWM generation unit
- Programmable PWM pulse width
- Single or double update modes
- Programmable dead time and switching frequency
- Twos-complement implementation which permits smooth transition to full-on and full-off states
- Dedicated asynchronous PWM shutdown signal

Functional Description

The following sections provide details on the functionality of the PWM.

- Architectural Concepts
- Timer Units

- Channel Timing Control Unit
- Output Disable and Cross-Over Modes
- Sync Operation Modes

ADSP-CM40x PWM Trigger List

Table 18-1: ADSP-CM40x PWM Trigger List Masters

Trigger ID	Name	Description	Sensitivity
19	PWM0_SYNC	PWM0 PWMTMR Group Trigger	Edge
20	PWM1_SYNC	PWM1 PWMTMR Group Trigger	Edge
21	PWM2_SYNC	PWM2 PWMTMR Group Trigger	Edge

Table 18-2: ADSP-CM40x PWM Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
48	PWM0_TRIP_TRIG0	PWM0 Trip Trigger Slave 0	None
49	PWM0_TRIP_TRIG1	PWM0 Trip Trigger Slave 1	None
50	PWM0_TRIP_TRIG2	PWM0 Trip Trigger Slave 2 (default configuration) or PWM0_SYNC	None
51	PWM1_TRIP_TRIG0	PWM1 Trip Trigger Slave 0	None
52	PWM1_TRIP_TRIG1	PWM1 Trip Trigger Slave 1	None
53	PWM1_TRIP_TRIG2	PWM1 Trip Trigger Slave 2 (default Configuration) or PWM1_SYNC	None
54	PWM2_TRIP_TRIG0	PWM2 Trip Trigger Slave 0	None
55	PWM2_TRIP_TRIG1	PWM2 Trip Trigger Slave 1	None
56	PWM2_TRIP_TRIG2	PWM2 Trip Trigger Slave 2 (default Configuration) or PWM2_SYNC	None

ADSP-CM40x PWM Interrupt List

Table 18-3: ADSP-CM40x PWM Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
12	PWM0_TRIP	PWM0 Trip Occurred	Level	
13	PWM1_TRIP	PWM1 Trip Occurred	Level	
14	PWM2_TRIP	PWM2 Trip Occurred	Level	
15	PWM0_SYNC	PWM0 PWMTMR Group Interrupt	Edge	

Table 18-3: ADSP-CM40x PWM Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
16	PWM1_SYNC	PWM1 PWMTMR Group Interrupt	Edge	
17	PWM2_SYNC	PWM2 PWMTMR Group Interrupt	Edge	

ADSP-CM40x PWM Register List

The Pulse-Width Modulator unit (PWM) includes multiple timers (providing period flexibility) and channels (providing mode, interrupt, and pulse shape flexibility), permitting a wide variety of PWM output options for motor control and other applications. A set of registers governs PWM operations. For more information on PWM functionality, see the PWM register descriptions.

Table 18-4: ADSP-CM40x PWM Register List

Name	Description
PWM_ACTL	Channel A Control Register
PWM_AH0	Channel A-High Duty-0 Register
PWM_AH0_HP	Channel A-High Heightened-Precision Duty-0 Register
PWM_AH1	Channel A-High Duty-1 Register
PWM_AH1_HP	Channel A-High Heightened-Precision Duty-1 Register
PWM_AH_DUTY0	Channel A-High Full Duty0 Register
PWM_AH_DUTY1	Channel A-High Full Duty1 Register
PWM_AL0	Channel A-Low Duty-0 Register
PWM_AL0_HP	Channel A-Low Heightened-Precision Duty-0 Register
PWM_AL1	Channel A-Low Duty-1 Register
PWM_AL1_HP	Channel A-Low Heightened-Precision Duty-1 Register
PWM_AL_DUTY0	Channel A-Low Full Duty0 Register
PWM_AL_DUTY1	Channel A-Low Full Duty1 Register
PWM_BCTL	Channel B Control Register
PWM_BH0	Channel B-High Duty-0 Register
PWM_BH0_HP	Channel B-High Heightened-Precision Duty-0 Register
PWM_BH1	Channel B-High Duty-1 Register
PWM_BH1_HP	Channel B-High Heightened-Precision Duty-1 Register
PWM_BH_DUTY0	Channel B-High Full Duty0 Register
PWM_BH_DUTY1	Channel B-High Full Duty1 Register
PWM_BL0	Channel B-Low Duty-0 Register

Table 18-4: ADSP-CM40x PWM Register List (Continued)

Name	Description
PWM_BL0_HP	Channel B-Low Heightened-Precision Duty-0 Register
PWM_BL1	Channel B-Low Duty-1 Register
PWM_BL1_HP	Channel B-Low Heightened-Precision Duty-1 Register
PWM_BL_DUTY0	Channel B-Low Full Duty0 Register
PWM_BL_DUTY1	Channel B-Low Full Duty1 Register
PWM_CCTL	Channel C Control Register
PWM_CH0	Channel C-High Pulse Duty Register 0
PWM_CH0_HP	Channel C-High Pulse Heightened-Precision Duty Register 0
PWM_CH1	Channel C-High Pulse Duty Register 1
PWM_CH1_HP	Channel C-High Pulse Heightened-Precision Duty Register 1
PWM_CHANCFG	Channel Configuration Register
PWM_CHOPCFG	Chop Configuration Register
PWM_CH_DUTY0	Channel C-High Full Duty0 Register
PWM_CH_DUTY1	Channel C-High Full Duty1 Register
PWM_CL0	Channel C-Low Pulse Duty Register 0
PWM_CL0_HP	Channel C-Low Pulse Duty Register 1
PWM_CL1	Channel C-Low Duty-1 Register
PWM_CL1_HP	Channel C-Low Heightened-Precision Duty-1 Register
PWM_CL_DUTY0	Channel C-Low Full Duty0 Register
PWM_CL_DUTY1	Channel C-Low Full Duty1 Register
PWM_CTL	Control Register
PWM_DCTL	Channel D Control Register
PWM_DH0	Channel D-High Duty-0 Register
PWM_DH0_HP	Channel D-High Pulse Heightened-Precision Duty Register 0
PWM_DH1	Channel D-High Pulse Duty Register 1
PWM_DH1_HP	Channel D High Pulse Heightened-Precision Duty Register 1
PWM_DH_DUTY0	Channel D-High Full Duty0 Register
PWM_DH_DUTY1	Channel D-High Full Duty1 Register
PWM_DL0	Channel D-Low Pulse Duty Register 0
PWM_DL0_HP	Channel D-Low Heightened-Precision Duty-0 Register
PWM_DL1	Channel D-Low Pulse Duty Register 1

Table 18-4: ADSP-CM40x PWM Register List (Continued)

Name	Description
PWM_DL1_HP	Channel D-Low Heightened-Precision Duty-1 Register
PWM_DLYA	Channel A Delay Register
PWM_DLYB	Channel B Delay Register
PWM_DLYC	Channel C Delay Register
PWM_DLYD	Channel D Delay Register
PWM_DL_DUTY0	Channel D-Low Full Duty0 Register
PWM_DL_DUTY1	Channel D-Low Full Duty1 Register
PWM_DT	Dead Time Register
PWM_ILAT	Interrupt Latch Register
PWM_IMSK	Interrupt Mask Register
PWM_STAT	Status Register
PWM_SYNC_WID	Sync Pulse Width Register
PWM_TM0	Timer 0 Period Register
PWM_TM1	Timer 1 Period Register
PWM_TM2	Timer 2 Period Register
PWM_TM3	Timer 3 Period Register
PWM_TM4	Timer 4 Period Register
PWM_TRIPCFG	Trip Configuration Register

Architectural Concepts

A clock, whose period is t_{SYSCLK}, drives the PWM controller. The PWM generator produces four pairs (four high-side and four low-side) of PWM signals on the eight PWM output pins. Each high and low pair constitutes a channel. For example, PWM_AL and PWM_AH make up channel A, and PWM_BL and PWM_BH make up channel B, and so on. Each pair of channel outputs references either a main timer or an independent timer. These timers operate on a switching frequency determined by the PWM_TM0 through PWM_TM4 registers. There are two duty registers for every PWM output. The registers enable generation of symmetrical or asymmetrical waveforms. The waveforms produce lower harmonic distortion in three-phase PWM inverters, with minimal CPU intervention.

Block Diagram

The ***PWM Block Diagram*** figure shows a block diagram that represents the main functional blocks of the PWM controller.

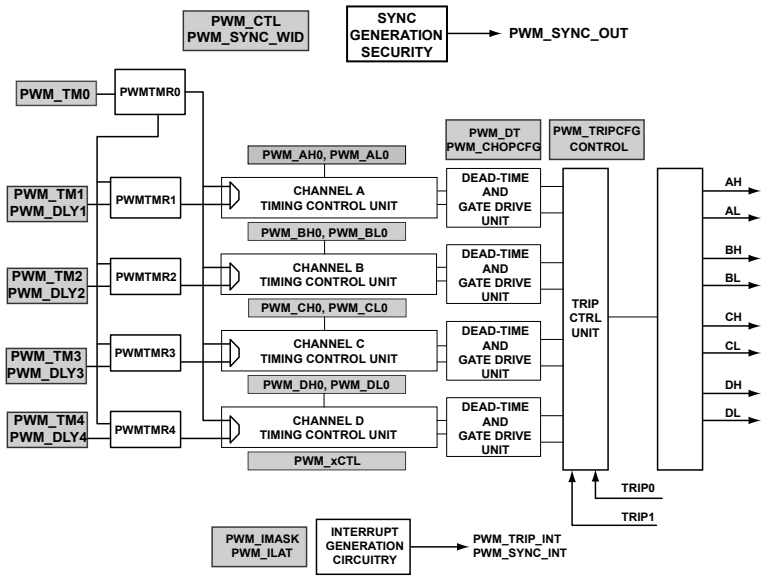

Figure 18-1: PWM Block Diagram

The following list describes the primary blocks.

- Each pair of PWM signals references either the main timer or the independent timer.
- PWMTMR0 is the main timer and can trigger the delayed start of the other timers.
- Timing control units, one for each channel, together form the core of the PWM. The unit generates the required complex waveforms on the high-side and low-side outputs for the respective channel.
- Dead-time insertion occurs after the ideal PWM output pair is generated.
- The gate drive unit generates the high-frequency chopping signal and then mixes it with the requisite PWM output signals.
- The PWM shutdown and interrupt controller manage the various PWM shutdown modes for the timing unit and generate the requisite interrupt signals.
- The PWM sync pulse control unit generates the internal PWM_SYNC pulse and also controls whether the external PWM_SYNC input pulse is used.

Timer Units

Five timers make up the time base for the PWM module. The main timer, PWMTMR0 operates at a switching frequency determined by the period register `PWM_TM0`. The four remaining timers (PWMTMR1 through PWMTMR4) can operate at independent switching frequencies determined by their respective registers.

The respective `PWM_TM[n]` register of these four timers can be programmed to work at a multiple of the main timer frequency. In this case, the `PWM_DLY[n]` registers control the lead-lag phase of a given timer based on the main timer PWMTMR0.

NOTE: When delayed operation of a timer enables:

- The register value of the timer must be equal to the PWM_TM0 register value, or
- The PWM_TM0 value must be an integer multiple of each register of the timer

Non-integer multiples are not allowed.

PWM Timer Period (PWM_TM) Registers

The 16-bit read/write PWM period registers (PWM_TM0 through PWM_TM4) control the PWM switching frequency. The fundamental timing unit of the PWM controller is t_{SYSCLK}. Therefore, the time increment (t_{SYSCLK}) is 10 ns for a 100-MHz system clock (SYSCLK) frequency, f_{SYSCLK}. The value written to the register of a timer is effectively the number of t_{SYSCLK} clock increments in one half of a PWM period. The following equation describes the required timer register value as a function of the desired PWM switching frequency (f_{PWM}):

$$PWM\_TM = f_{SYSCLK}/2 \times f_{PWM}$$

Therefore, the PWM switching period (T_s) is:

$$T_s = 2 \times PWM\_TM \times t_{SYSCLK}$$

For example, for an f_{SYSCLK} of 100 MHz and a desired PWM switching frequency (f_{PWM}) of 10 kHz (T_s = 100 ms), the correct value to load into the timer register is:

$$PWM\_TM = 100 \times 10^6 \div 2 \times 10 \times 10^3 = 5000$$

The largest value that can be written to the 16-bit timer register is 0xFFFF = 65,535. For an f_{SYSCLK} of 100 MHz, this value corresponds to a minimum PWM switching frequency of:

$$f_{PWM(min)} = 100 \times 10^6 \div 2 \text{ x } 65535 = 762 \text{ Hz}$$

NOTE: Timer register values of 0 and 1 are not defined. Do not use these values when the PWM outputs or PWM is enabled.

Timer Unit Operation

The PWM timers are up-down counters, and they operate on the peripheral clock with a period of t_{CK}. The period of the PWM timer is divided into two halves. In the first half, the timer roughly counts down from PWM_TMx/2 to –PWM_TMx/2. During this half, the PWM_STAT.TMR0PHASE through PWM_STAT.TMR4PHASE bits are held at 0. In the second half of the period, the timer roughly counts up from –PWM_TMx/2 to PWM_TMx/2. The PWM_STAT.TMR0PHASE through PWM_STAT.TMR4PHASE bits indicates a 1 during this half.

The actual partition of the periods varies slightly between odd and even values of the half-period, in the PWM_TM[n] registers.

If a timer register value is odd, for example 11, then that timer loads +5 at the beginning of the period. The timer counts down from +5 to –5 in the first half, reloads –5 at the midpoint and counts up from –5 to +5 in the second half. The reload values at the period and mid-period boundaries are the same as the previous count. The timer

counts 2 × 11 half-periods = 22 total counts in the entire period as shown in the ***Operation of Timer for Odd Value of PWM_TM*** figure.

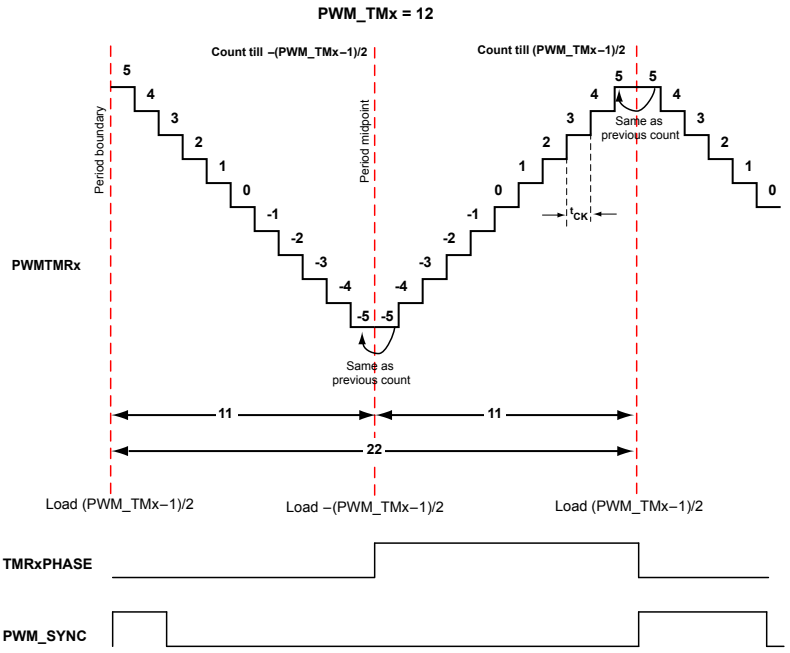

Figure 18-2: Operation of Timer for Odd Value of PWM_TM

When the timer register value is even, for example 12, then that timer loads +5 at the beginning of the period. The timer counts from +5 to –6 in the first half, reloads –5 at the midpoint and counts up from –5 to +6 in the second half. The reload values at the period and mid-period boundaries are different from the previous count. It counts 2 × 12 half-periods = 24 total counts in the entire period as shown in the ***Operation of Timer for Even Value of PWM_TM*** figure.

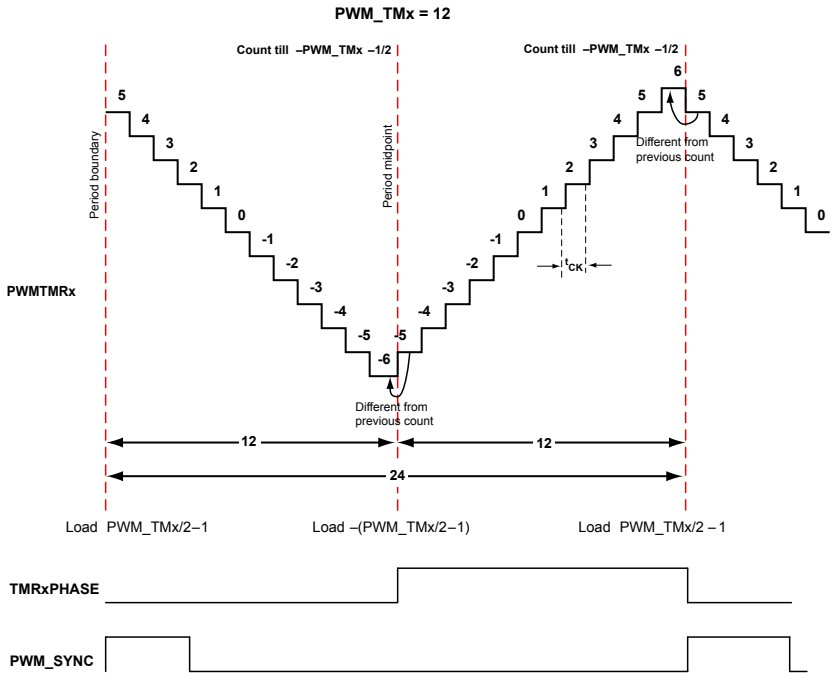

Figure 18-3: Operation of Timer for Even Value of PWM_TM

NOTE: In the operation discussed in this section, double-buffering of all channel registers and the timer registers takes place at the period boundary of the respective timers.

Phase Offset Control

The PWM timers (PWMTMR1 through PWMTMR4) can operate with a programmable phase lag relative to the main timer, PWMTMR0. To implement phase offset for a channel, use the counter-registers for channel delay (`PWM_DLY[n]`) with the PWMTMR0 and set the `PWM_CTL.DLYAEN` bit to l.

Phase offset works as follows.

1. If phase lag is used for channel A (and channel A uses PWMTMR1 to generate a duty cycle), when PWMTMR0 reaches its period boundary, it triggers the `PWM_DLYA` register. The register counts out the number SYSCLK cycles that are equal to the value programmed in the `PWM_DLYA` register.
2. At the end of this count, the `PWM_DLYA` register sends out a trigger to PWMTMR1. It receives a synchronization pulse in every period of PWMTMR0 at a point delayed from its period boundary by the value in the `PWM_DLYA` register.

For more information on how channels can reference different timers for their outputs, see Channel Timing Control Unit.

NOTE: Satisfy the following conditions when using this feature on timer y for channel Y relative to PWMTMRx.

- Program the `PWM_DLY[n]` register to a value less than 2 × `PWM_TM[n]`.
- `PWM_TM0` = N × `PWM_TM[n]`, where N is an integer.

The function of PWM_TM[n] (PWMTMR1 in the example) differs in cases where PWM_TM0 = PWM_TM1 (Case 1) to cases where PWM_TM0 = N × PWM_TM1 (Case 2). The following examples describe both cases.

- **Case 1: PWM_TM0 = PWM_TMy**

When PWM_TM0 = PWM_TMy, PWMTMRy restarts its period after receiving the synchronization pulse from the channel delay register (PWM_DLY[n]). If the trigger from the PWM_DLY[n] register is late, PWMTMRy holds its count until the trigger occurs. If the trigger is a bit early, PWMTMRy reloads without regard to whether it has completed its current period. As a result, PWMTMRy resyncs with PWMTMR0 with the phase lag programmed in the PWM_DLYA register in every one of its periods.

In this case, the expiration of the delay registers (PWM_DLY[n]) is the period boundary of PWMTMRy. Now, all the double buffered registers related to the given channel update (except the delay registers which are double buffered at the period boundary of PWMTMR0).

The ***Phase Offset Control Using DELAY*** figure shows an example where:

- PWM_TM0, PWM_TM1, and PWM_TM2 are programmed with the same value.
- PWM_DLYA and PWM_DLYB are programmed with values DELAY1 and DELAY2 respectively, such that DELAY2 > DELAY1.
- The outputs of Channel A are referenced to PWMTMR1. The outputs of channel B are referenced to PWMTMR2.

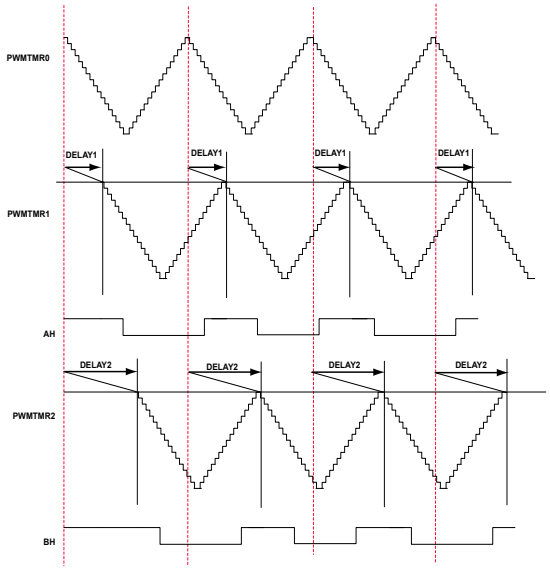

Figure 18-4: Phase Offset Control Using DELAY

The delay registers are double buffered and the new value of DELAY reloads at the period boundary of PWMTMR0. The two options exist when the new value is different from the older one. The behavior of PWMTMRy in both these cases is discussed. The ***Impact of New DELAY Value on Timer Count for Equal Timer Periods*** figure shows the behavior in the two cases. It is assumed that channel B references its outputs to PWMTMR0 and channel A references its outputs to PWMTMR1.

1. The new delay value is higher than the previous value. Here the corresponding PWMTMRy allows more than one time period between consecutive triggers from the channel delay (`PWM_DLY[n]`) register. In this case, after reaching its period boundary, PWMTMRy holds its count at the period boundary and waits for the trigger from the channel delay register. ***Impact of New DELAY Value on Timer Count for Equal Timer Periods*** figure shows case A functionality.
2. The next delay value programmed is smaller than the previous value. Here, the corresponding PWMTMRy allows only less than one time period between consecutive triggers from the channel delay register. Though the trigger comes earlier in this case, before PWMTMRy has counted out one full period, it reloads and starts its period again. The ***Impact of New DELAY Value on Timer Count for Equal Timer Periods*** figure shows case B functionality.

Therefore, PWMTMR1 waits and obeys a synchronization pulse from the `PWM_DLYA` register in every one of its periods.

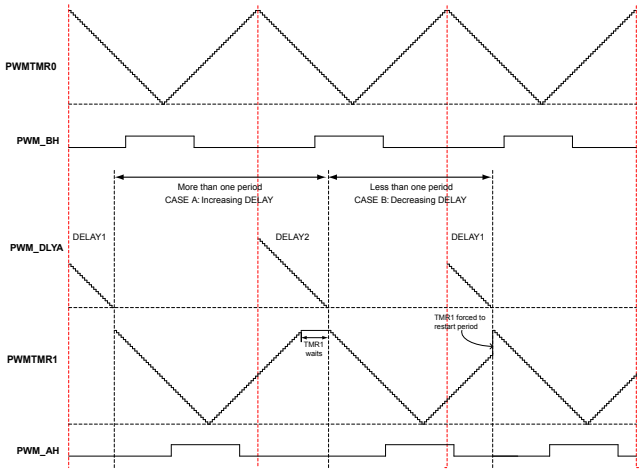

Figure 18-5: Impact of New DELAY Value on Timer Count for Equal Timer Periods

- **Case 2: PWM_TM0 = N x PWM_TMy**

In this case, within a single period of PWMTMR0 a program can fit multiple periods (N) of PWMTMRy. Additionally, the channel delay register (`PWM_DLYx`) is triggered only once every N periods of PWMTMRy.

The operation is as follows: Every Nth period of PWMTMRy, PWMTMRy expects a synchronization pulse from the `PWM_DLY[n]` register. When this register counts out that period and the trigger has not yet arrived, PWMTMRy waits at the end of the period for the trigger. PWMTMRy starts counting down once the trigger arrives. If the trigger comes earlier, PWMTMRy restarts immediately without waiting to complete the period count.

In the intervening periods, PWMTMRy operates independently. As the period ends, PWMTMRy reloads and starts the next period without intervention from the channel delay register.

The ***Impact of DELAY Value Change for the Multiple Timer Periods*** shows an example with N = 2. PWMTMRy syncs up with PWMTMR0 every second period, and is free running across every odd period boundary.

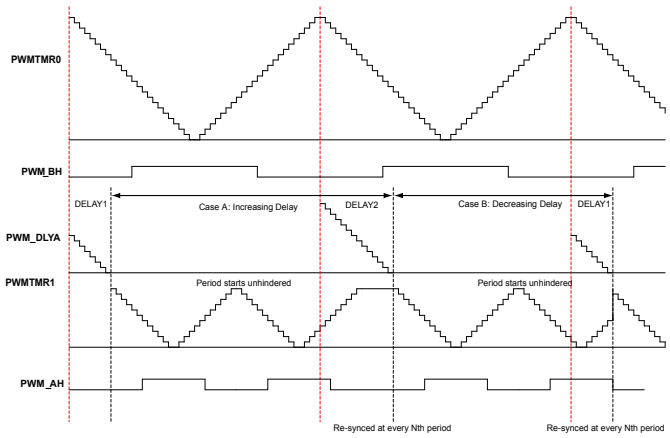

Figure 18-6: Impact of DELAY Value Change for Multiple Timer Periods

Channel Timing Control Unit

The channel timing control unit is the core of the PWM. There are four separate channels, each channel controlling a pair of output signals – the high-side output and the low-side output.

Channel Control

The PWM_CHANCFG register controls the static configuration of all the channels and is initialized once before the beginning of a PWM operation.

NOTE: The PWM_CHANCFG register is not double buffered. Do not change the contents of the register once the PWM is enabled.

Each channel works with a reference timer base. The time base can be either the main timer PWMTMR0 or the appropriate PWMTMRx. Configure the time base with the PWM_CHANCFG.REFTMRA bit field as follows.

- Channel A works with PWMTMR0 or PWMTMR1.
- Channel B works with PWMTMR0 or PWMTMR2.
- Channel C works with PWMTMR0 or PWMTMR3.
- Channel D works with PWMTMR0 or PWMTMR4.

The double-buffered channel control registers (PWM_ACTL through PWM_DCTL) contain bits that control the dynamic pulse behavior of the channel outputs. These registers have bits that enable or disable outputs and select the pulse position of outputs (explained in the following section).

Pulse Positioning and Duty Cycle Registers

The PWM uses the PULSEMODEHI and PULSEMODELO bit fields of the PWM_[n]CTL registers to define the region within the timer period where the output pulses are positioned.

- When the PWM_CHANCFG.MODELSC bit is 0, the PWM uses the PULSEMODEHI field to specify the pulse positioning for both the high-side and low-side outputs of the channel.
- When the bit is 1, the PWM uses PWM_ACTL.PULSEMODELO to define the pulse positioning for the low-side output of the channel. It uses the PWM_ACTL.PULSEMODEHI to define the pulse positioning for the high-side output of the channel.

Each channel output has two duty-cycle registers: PWM_AH0 and PWM_AH1 for the high-side output, and PWM_AL0 and PWM_AL1 for the low-side output. These registers determine the width of the output pulses. When the PWM_CHANCFG.MODELSC bit is 0, the high-side duty-cycle registers are used to determine the width of the output pulse for the low side. The duty cycle range that can be programmed into these registers is between –PWMTMx/2 and +PWMTMx/2 , when ignoring dead time.

When including dead time for channel A, for PULSEMODEs 00 and 01, the programmed duty cycle is modified. The range is limited between the values [–PWMTMx/2 + PWM_CHA_DT] and [+PWMTMx/2 + PWM_CHA_DT] considering the high-side output. For PULSEMODEs 10 and 11, the high-side duty cycle registers range is limited between values [PWM_TMx/2 + PWM_CHA_DT] and [–PWM_TMx/2 – PWM_CHA_DT].

The section Switching Dead Time (PWM_DT) Register explains dead time in detail.

NOTE: Values programmed into these registers that fall outside these limits result in over or under modulation.

Duty Cycle and Pulse Positioning Control

The PWM_ACTL.PULSEMODEHI and PWM_ACTL.PULSEMODELO fields control how the duty cycle registers modify the waveform of the high and low-side outputs. (The PWM_ACTL.PULSEMODEHI and PWM_ACTL.PULSEMODELO fields are referred to as *pulse mode* in the subsequent discussion.)

- Pulse mode = 00 – Produce a symmetrical pulse waveform around the center of the PWM period. In this mode, PWM uses only one of the duty cycle registers for an output. For example, for the AH output, PWM uses only the PWM_AH0 register. In this mode, the values in the duty cycle registers are scaled such that a value of 0 produces a 50% duty cycle.
- Pulse mode = 01 – Produce an asymmetrical pulse waveform around the center of the PWM period. In this mode, PWM uses both duty cycle registers. For example, for the PWM_AH output, PWM uses the PWM_AH0 and PWM_AH1 registers. In this mode, if the PWM_AH1 register is programmed with the same value as the PWM_AH0 register, the output is identical to the output when pulse mode =00.
- Pulse mode = 10 or 11 – Produce pulse waveforms either on the first half or the second half of the PWM period respectively. PWM uses both PWM_AH0 and PWM_AH1 registers.

  Pulse mode = 10. If the low side works from the low-side duty-cycle registers, strictly adhere to the condition PWM_AL0> PWM_AL1.

  In pulse mode = 11. If the low side works from the low-side duty-cycle registers, strictly adhere to the condition PWM_AL0< PWM_AL1.

The ***Pulse Positioning Modes*** figure shows the pulse positioning modes as previously described for PWM_AH. In the figure, DUTY0 is the value in the PWM_AL0 register and DUTY1 is the value in the PWM_AH1 register. The step signal, count, indicates the output of the timer for channel A. In the example, the signal is configured as active high and dead time is zero.

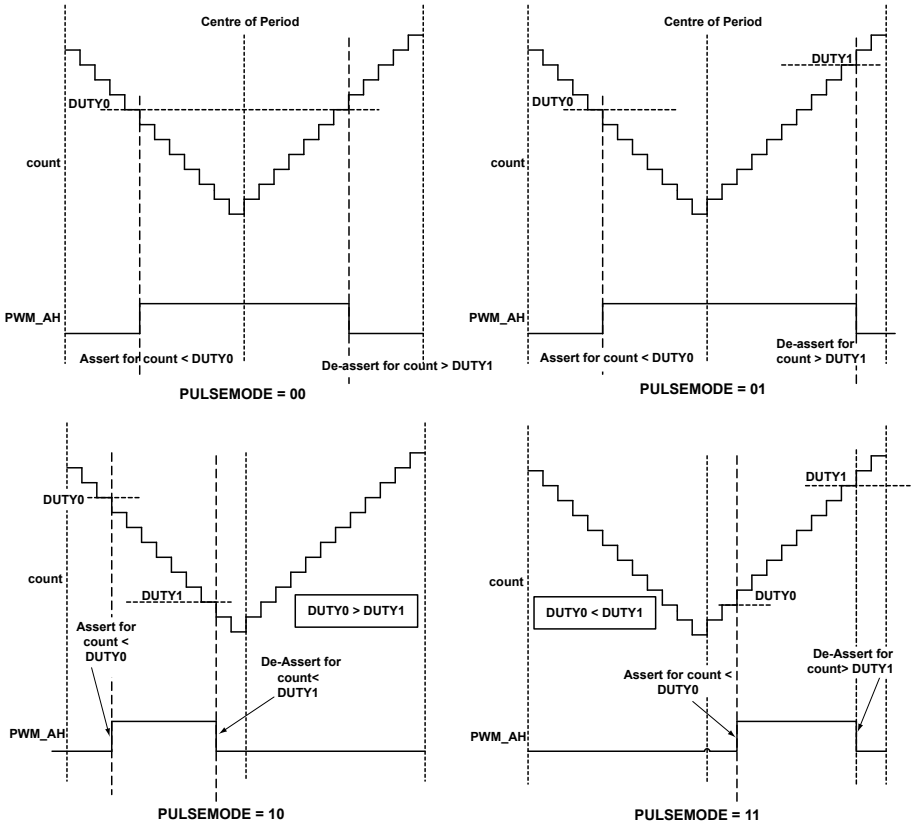

Figure 18-7: Pulse Positioning Modes

Channel Low Side Output Dependent Operation Mode and Dead Time

The low-side output waveform can be programmed to depend on the waveform of the high-side output or to be independent. The PWM uses the PWM_CHANCFG.MODELSC bit to control this functionality.

For example, channel A produces the high-side output PWM_AH and the low-side output PWM_AL. When the PWM_CHANCFG.MODELSC bit =0, the low-side output is also generated using the high-side duty-cycle registers for pulse width, the PWM_ACTL.PULSEMODEHI bits for pulse positioning and the PWM_CHANCFG.POLAH bit for polarity. If the PWM_DT register is 0, the low-side output is an inverted version of the high-side output.

When the PWM_DT register is programmed with a non-zero value, both the high-side and low-side outputs are scaled symmetrically about the points of transition in the zero dead time case. The PWM scales the output by the value programmed in the PWM_DT register.

The ***Channel Outputs in Dependent Mode for Pulse Mode*** figures show the high and low-side outputs for the case with zero and non-zero dead time for PWM_ACTL.PULSEMODEHI =00 and 01. DUTY0 is the value programmed

into the PWM_AH0 register. DUTY1 is the value programmed into the PWM_AH1 register. The PWM_CHANCFG.POLAH bit =1, indicates that both signals are active high. The PWM_DT register holds the value DT.

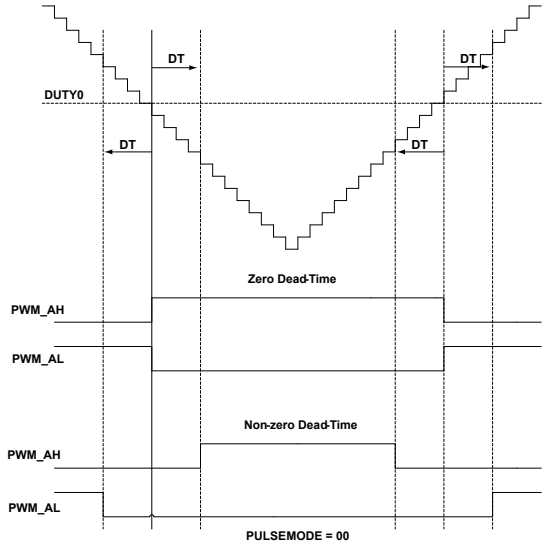

Figure 18-8: Channel Outputs in Dependent Mode for Pulse Mode = 00

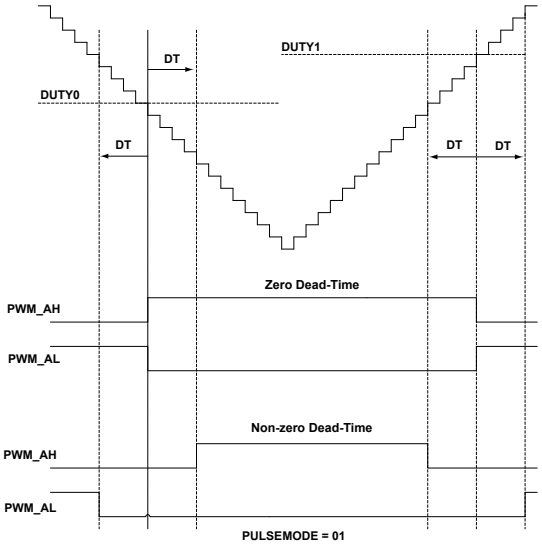

Figure 18-9: Channel Outputs in Dependent Mode for Pulse Mode = 01

The following pair of figures shows the high and low-side outputs for the case with zero and non-zero dead-time for PWM_ACTL.PULSEMODEHI =10 and 11. In the figures, DUTY0 is the value programmed into PWM_AH0 register and DUTY1 is the value programmed into the PWM_AH1 register. PWM_CHANCFG.POLAH is 1 indicating that both signals are active high. The PWM_DT register holds the value DT.

NOTE: Using dead time, the guidelines for programming the duty-cycle registers in pulse modes 10 and 11 given in Duty Cycle and Pulse Positioning Control are modified as follows: Pulse mode 10: PWM_xH0 – DT > PWM_xH1 + DT Pulse mode 11: PWM_xH0 + DT < PWM_xH1 – DT

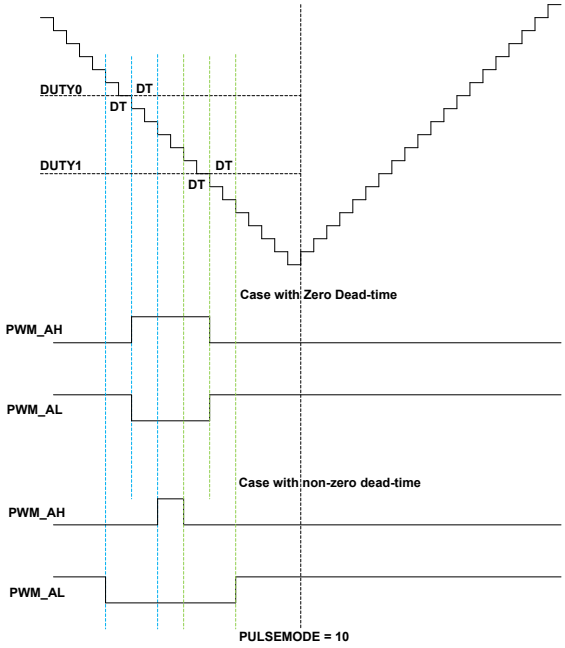

Figure 18-10: Channel Outputs in Dependent Mode for Pulse Mode = 10

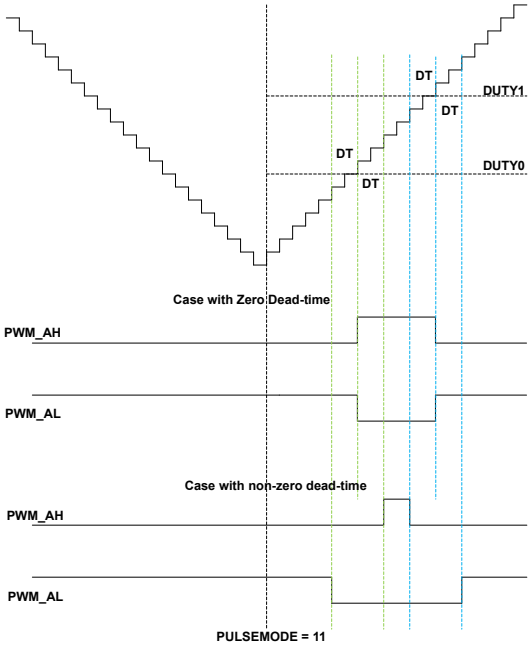

Figure 18-11: Channel Outputs in Dependent Mode for Pulse Mode = 11

Channel High Side and Low Side Outputs, Independent Operation Mode

Independent control of the `PWM_AH0` and `PWM_AL0` channel outputs is possible by setting the `PWM_CHANCFG.MODELSA` bit to 1. In this case, the PWM module:

- Generates PWM_AH using the `PWM_AH0` register.

- Uses thePWM_AH1 register to configure pulse width.
- Uses the PWM_ACTL.PULSEMODEHI bit to configure pulse position.
- Uses the PWM_CHANCFG.POLAH bit to configure polarity.
- Generates PWM_AL using PWM_AL0.
- Uses the PWM_AL1 register to configure pulse width.
- Uses the PWM_ACTL.PULSEMODELO bit to configure pulse position.
- Uses the PWM_CHANCFG.POLAL bit to configure polarity.

NOTE: In independent mode, the dead-time insertion is not applicable. The hardware forces dead time to zero.

The ***PWM_AH and PWM_AL in Independent Operation Mode*** figure shows an example of the independent mode of operation where PWM_AH and PWM_AL work from different register bits.

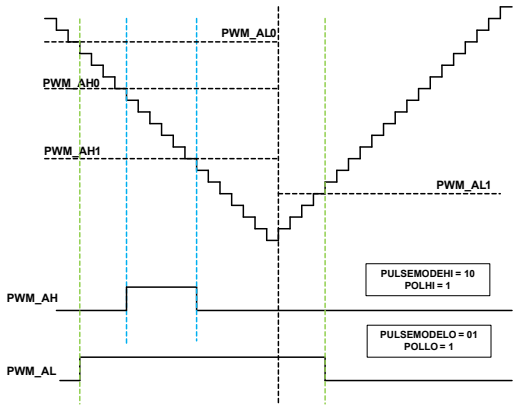

Figure 18-12: PWM_AH and PWM_AL in Independent Operation Mode

PWM_AH and PWM_AL can be positioned in the timer period with a given phase difference between them. Program the PWM_ACTL.PULSEMODEHI and PWM_ACTL.PULSEMODELO bits to different values to achieve this positioning as shown in the ***Channel Outputs Controlled Independently*** figure.

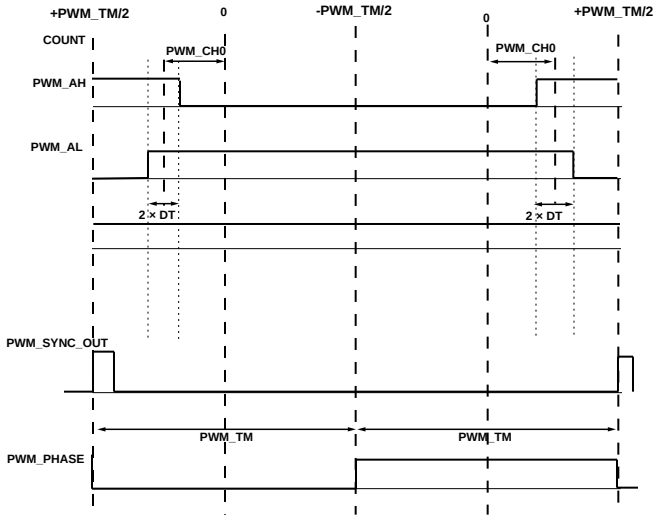

Figure 18-13: Channel Outputs Controlled Independently

Switched Reluctance Motors Application

In typical power converter configurations for switched or variable reluctance motors, motor winding is connected between the two power switches of a given inverter leg. To allow for a complete circuit in the motor winding, turn on both switches at the same time.

PWM uses switched reluctance motors in the following configurations: hard chop, alternate chop, soft chop—bottom on, and soft chop—top on.

The ***Four SR Mode Types, Active High PWM Output Signals*** figure shows the four SR mode types as active high PWM output signals.

Hard chop mode contains independently programmed rising edges of the high and low signals of a channel in the same PWM half cycle. Both signals contain independently programmed falling edges in the next PWM half cycle. The `PWM_CHANCFG.POLAH` and `PWM_CHANCFG.POLAL` bits are programmed to same values.

Alternate chop mode is similar to normal PWM operation except that the PWM channel high and low signal edges are always opposite and are independently programmed. The `PWM_CHANCFG.POLAH` and `PWM_CHANCFG.POLAL` bits are programmed to opposite values. The low-side invert is the only difference between hard chop mode and alternate chop mode.

Soft chop—bottom on uses a 100% duty on the low side of the channel. Soft chop—top on uses a 100% duty on the high side of the channel. Similar to hard chop mode, PWM uses the `PWM_AH0` duty register for the high channel and the `PWM_AL0` duty register for the low channel.

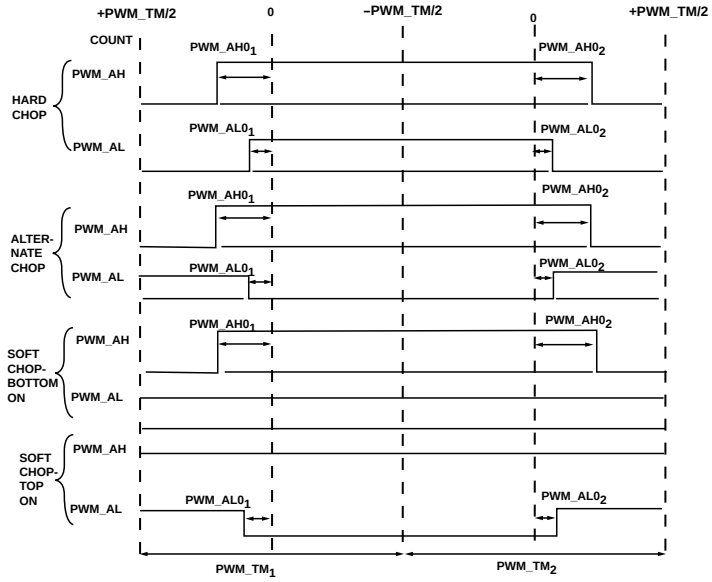

Figure 18-14: Four SR Mode Types, Active High PWM Output Signals

Switching Dead Time (PWM_DT) Register

The second important parameter that must be set up in the initial configuration of the PWM controller is the switching dead time. Dead time is a short delay introduced between turning off one PWM signal (for example, AH) and turning on the complementary signal (for example, AL). This short time delay permits turning off a power switch (AH in this case) to completely recover its blocking capability before the complementary switch is turned on. This time delay prevents a potentially destructive short-circuit condition from developing across the DC link capacitor of a typical voltage source inverter.

The 10-bit, read/write `PWM_DT` Channel A through Channel D registers control the dead time for channel-x. If the value carried by any dead-time register is PWMDT, the dead time, Td, for that channel is:

T_d = PWM_CHx_DT × 2 × t_{SYSCLK}

Therefore, a dead-time value of 0x00A introduces a 200-ns delay (for an SYSCLK of 100 MHz). The delay occurs between turning off any PWM signal (for example, AH) and then turning on its complementary signal (for example, AL). The length of dead time can be programmed in increments of 2 × t_{SYSCLK} (or 20 ns for an SYSCLK of 100 MHz). The `PWM_DT` Channel A through Channel D registers have a maximum value of 0x3FF (1023 decimal) and corresponds to a maximum programmed dead time of:

$T_{d(max)}$ = 1023 × 2 × t_{SYSCLK} = 1023 × 2 × 10 × 10^{-9} = 20.5 µs for an f_{SYSCLK} rate of 100 MHz.

Write 0 to the `PWM_DT` Channel A through Channel D registers to program the dead time.

Duty Cycle with Dead-Time Control: Calculations for PULSEMODE 00

The duty cycle registers are scaled so that a value of 0 represents a 50% PWM duty cycle. The switching signals produced are also adjusted to incorporate the programmed dead-time value using the `PWM_DT` registers. The unit in

this case produces active low signals so that a low level corresponds to a command to turn-on the associated power device.

The ***Dead Time Between Outputs in Dependent Mode*** figure shows a typical pair of PWM outputs, PWM_AH and PWM_AL. The time values in the figure indicate the integer value in the associated register and can be converted to time by multiplying by the fundamental time increment, t_{CK}. In the example, channel A is working from PWMTMR0.

In the example, the pulse mode is set to 00 so that the switching patterns are perfectly symmetrical about the mid-point of the switching period. The dead time is incorporated by moving the switching instants of both PWM signals away from the instant set by the `PWM_AH0` register. Both switching edges are moved by an equal amount (DT × t_{CK}) to preserve the symmetrical output patterns. Also shown is the `PWM_SYNC` output pulse whose rising edge denotes the beginning of the switching period, and the `PWM_STAT.TMR0PHASE` bit.

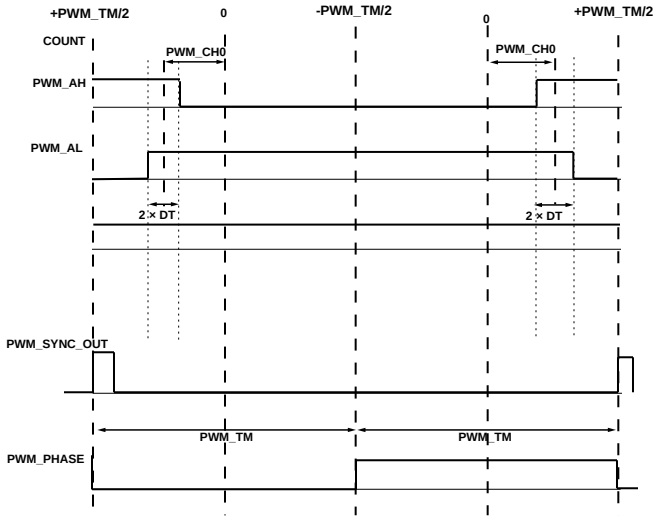

Figure 18-15: Dead Time Between Outputs in Dependent Mode

The PWM timing unit produces the resulting on-times (active low) of the PWM signals over the full PWM period (two half-periods). The figure illustrates this timing. The timing can be written per the following equation.

T_{AH} = (PWM_TM0 + 2 × (PWM_AH0 - PWM_CHx_DT)) × $t_{CK;}$

Range of T_{AH} is [0:2 × PWM_TM0 × t_{CK}]

T_{AL} = (PWM_TM0 - 2 × (PWM_AH0 + PWM_CHx_DT)) × $t_{CK;}$

Range of T_{AL} is [0:2 × PWM_TM0 × t_{CK}]

$$d_{AH} = \frac{T_{AH}}{Ts} = \frac{1}{2} + \frac{PWM\_AH0 - DT}{PWM\_TM}$$

$$d_{AL} = \frac{T_{AL}}{Ts} = \frac{1}{2} - \frac{PWM\_AH0 + DT}{PWM\_TM0}$$

The negative values of T_{AH} and T_{AL} are not permitted and the minimum permissible value is zero, corresponding to a 0% duty cycle. In a similar fashion, the maximum value is T_s, the PWM switching period, corresponding to a 100% duty cycle. Calculation of duty for other pulse modes can be similarly executed.

Special Consideration for PWM Operation in Over-Modulation

The PWM timing unit can produce PWM signals with variable duty cycle values at the PWM output pins. In pulse modes 00 and 01, at the extremities of the modulation process, duty cycles of 0% and 100% occur. In pulse modes 01 and 10, at the extremities of the modulation process, duty cycles of 0% and 50% occur. The modulation is called *full off* when the lower extremity of modulation is set for any PWM timer period for the corresponding channel. The modulation is called *full on* when the higher extremity of modulation is set for any PWM timer period for the corresponding channel. In between, for other duty cycle values, the operation is termed *normal modulation*.

Full On Modulation

In pulse modes 00 and 01, a PWM channel is in full on modulation if the high-side output of that channel is asserted. The output is asserted for the whole duration of the period of the PWM timer that channel is referencing. The conditions for full on modulation are:

- PWM_xH0 - DT > PWM_TMy/2 for pulse mode 00
- PWM_xH1 - DT > PWM_TMy/2 for pulse modes 00 and 01

In pulse mode 10, a PWM channel is in full on modulation if the high-side output of that channel is asserted. The output is asserted for the whole duration of the first half period of the PWM timer that the channel is referencing. The conditions for full on modulation are:

- PWM_xH0 - DT > PWM_TMy/2 for pulse mode 10
- PWM_xH1 + DT < -PWM_TMy/2 for pulse mode 10

In pulse mode 11, a PWM channel is in full on modulation if the high-side output of that channel is asserted. The output is asserted for the whole duration of the second half period of the PWM timer that the channel is referencing. The conditions for full on modulation are:

- PWM_xH0 + DT < -PWM_TMy/2 for pulse mode 11
- PWM_xH1 - DT > PWM_TMy/2 for pulse mode 11

Full Off Modulation

In pulse modes 00 and 01, a PWM channel is in full off modulation if the high-side output of that channel is deasserted. The output is deasserted for the whole duration of the period of the PWM timer that channel is referencing. The conditions for full off modulation are:

- PWM_xH0 - DT < -PWM_TMy/2 for pulse mode 00
- PWM_xH1 - DT < -PWM_TMy/2 for pulse modes 00 and 01

In pulse mode 10, a PWM channel is in full off modulation if the high-side output of that channel is deasserted. The output is deasserted for the whole duration of the first half period of the PWM timer that the channel is referencing. In the second half-period, it is deasserted anyway. The conditions for full off modulation are:

- PWM_xH0 - DT < -PWM_TMy/2 for pulse mode 10
- PWM_xH1 + DT < PWM_xH0 - DT for pulse mode 10

In pulse mode 11, a PWM channel is in full off modulation if the high-side output of that channel is deasserted. The output is deasserted for the whole duration of the second half period of the PWM timer that the channel is referencing. In the first half of the period, it is deasserted anyway. The conditions for full off modulation are:

- PWM_xH0 + DT > PWM_TMy/2 for pulse mode 11
- PWM_xH1 - DT > PWM_xH0 + DT for pulse mode 11

Normal Modulation

All other cases of modulation fall under this category.

Emergency Dead-Time Delays

Sometimes, during modulation transition, it is necessary to insert more emergency dead-time delays to prevent potential shoot through conditions in the inverter. (For example, when the PWM transitions into or out of full on or full off modulation). Disabling and enabling usage (related to the `PWM_ACTL.DISHI` and `PWM_ACTL.DISLO` bits) also can potentially cause outputs to violate shoot-through condition criteria. Another case is when large values vary the phase delay of a PWM timer. These transitions are detected automatically. If appropriate for safety, an emergency dead-time is inserted to prevent shoot through conditions.

There is another atypical case for the insertion of the additional emergency dead time. It occurs when both PWM signals do not toggle within a dead time of each other. In this case, insert more emergency dead time into one of the PWM signals of a given pair during these transitions. The dead-time delay is inserted into the PWM signal that is toggling into the on-state. In effect, an amount ($2 \times DT \times t_{CK}$) from the rising edge of the opposite output delays the turn-on of this signal. After this delay, the PWM signal is allowed to turn-on provided the desired output is still scheduled to be in the on-state after the emergency dead-time delay.

The ***Over Modulation Transition Example*** figure illustrates two examples of such a transition. In the figure, `PWM_ACTL.PULSEMODEHI` is kept at 1. The `PWM_AH` signal has been in full on modulation for some time and, during the current period, its pulse mode is changed to 10, keeping the full on condition. At the half-period boundary, `PWM_AH` is forced to transition to a deasserted state because pulse mode is 10. An emergency dead-time is inserted on the low-side output.

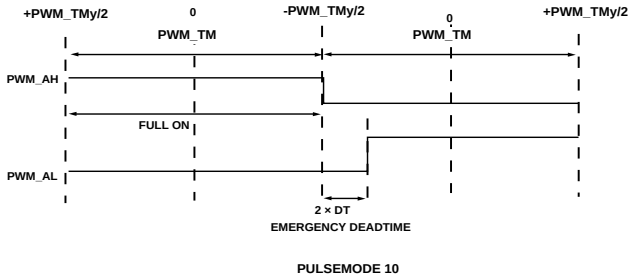

Figure 18-16: Over Modulation Transition Example

Gate Drive Unit

The gate drive unit of the PWM adds features that simplify the design of isolated gate drive circuits for PWM inverters. If using a transformer coupled power device gate drive amplifier, then the active PWM signal must be chopped at a high frequency. The `PWM_CHOPCFG` register allows the programming of this chopping mode for high frequency. The chopped active PWM signals can be required for the high-side drivers only, for the low-side drivers only, or for both the high-side and low-side switches. Therefore, independent control of this mode for both high and low-side switches is included with two separate control bits in the `PWM_CHANCFG` register.

The ***High-Side and Low-Side Outputs With Gate Chop Enabled*** figure shows the typical PWM output signals with high-frequency chopping enabled on both high-side and low-side signals. Chopping of the PWM outputs is enabled by setting bits in `PWM_CHANCFG` register. The 8-bit `PWM_CHOPCFG.VALUE` value controls the high frequency chopping. The following equation gives the period of this high frequency carrier.

$T_{chop} = [4 \times (CHOPDIV + 1)] \times t_{CK}$

and the chopping frequency is therefore an integral subdivision of the peripheral clock frequency:

$f_{chop} = f_{CK}/[4 \times (CHOPDIV + 1)]$

The `PWM_CHOPCFG.VALUE` value can range from 0 to 255, corresponding to a programmable chopping frequency rate from 122 kHz to 31.25 MHz for a 125 MHz, f_{CK} rate. Program the gate drive features before enabling the PWM controller. Do not change the gate drive features during normal operation of the controller. Following a reset, clear all bits of the `PWM_CHANCFG` register so that high frequency chopping is disabled, by default.

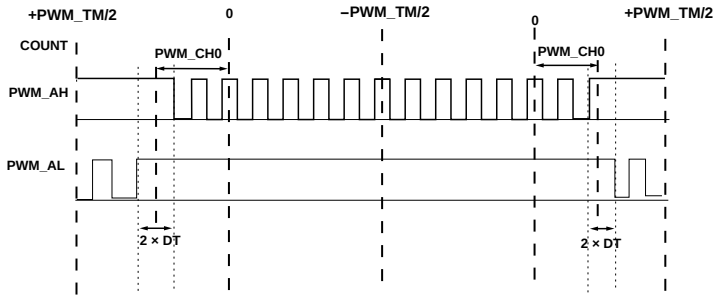

Figure 18-17: High-Side and Low-Side Outputs With Gate Chop Enabled

Output Control Feature Precedence

The order of applying output control features to the PWM signal is important and significant. Use the following order for applying the signal features to the PWM output signal.

1. Duty generation
2. Cross-over
3. High-side or low-side disable
4. Emergency dead-time insertion
5. HPPWM correction for wrong programming
6. Gate-drive chopping
7. Polarity
8. Heightened-Precision PWM (HPPWM) edge placement

NOTE: When HPPWM operation is enabled, the ***cross-over*** feature and the ***gate-drive chopping*** feature must be disabled.

Operating Modes

The PWM generator is capable of operating in the following modes:

- Emulation Mode
- High Precision 1 (medium)
- High precision 2
- Internal or External SYNC Mode
- Trigger Slave Mode
- Output Disable and Cross-Over Modes

Sync Operation Modes

The `PWM_SYNC` signal is internally generated as a function of `PWM_TM0.VALUE` and `PWM_SYNC_WID.VALUE` or is input externally. Multiple PWM configurations can be established with each PWM operating with its own independent `PWM_SYNC` signal or from its own or shared external `PWM_SYNC` signal. The external `PWM_SYNC` can be synchronous to the internal clock as in the case of a primary PWM generating an internal `PWM_SYNC` signal which drives the secondary `PWM_SYNC_IN` pin. The external `PWM_SYNC` can also be asynchronous to the internal clock as is typically the case of an off-chip `PWM_SYNC` signal used to drive each `PWM_SYNC_IN` pin for each PWM.

Internal PWM SYNC Generation

The PWM controller produces an output PWM synchronization pulse at a rate equal to period of a selected PWM timer. Programming the PWM_CTL.INTSYNCREF field controls this selection.

The PWM_SYNC pulses are generated at their respective period boundaries if:

- the other timers are running with a non-zero DELAY offset in relation to PWMTMR0, and
- the PWM_SYNC pulse is referenced to any of these timers.

The boundaries have a lag-lead offset compared to PWMTMR0.

This pulse is available for external use at the PWM_SYNC_OUT pin. The width of the PWM_SYNC pulse is programmable by the 10-bit read/write PWM_SYNC_WID register. The following equation gives the width of the PWM_SYNC pulse.

$t_{PWMSYNC} = t_{SYSCLK} \times (PWMSYNCWT + 1)$

The width of the pulse is programmable from t_{CK} to 1024 t_{CK} (corresponding to 8 ns to 8.19 µs for a f_{CK} rate of 125 MHz). Following a reset, the PWM_SYNC_WID register contains 0x3FF (1023 decimal) so that the default PWM_SYNC width is 8.19 µs, for a 125 MHz f_{CK}.

External PWM SYNC Generation

By setting the PWM_CTL.EXTSYNC bit, the PWM is set up in a mode to expect an external PWM_SYNC signal on the PWM_SYNC_IN pin. The external PWM_SYNC signal only determines the operation of the main timer PWMTMR0.

Synchronize the external sync by setting the PWM_CTL.EXTSYNCSEL bit to 0 (assumes the external PWM_SYNC selected is asynchronous).

The external PWM_SYNC period is expected to be an integer multiple of the value of the PWM_TM0 period register. When the rising edge of the external PWM_SYNC is detected, the PWMTMR0 timer is restarted at the beginning of its period. If the external PWM_SYNC period is not exactly an integer multiple of the internal PWM_SYNC, the behavior of the PWM channel outputs which are referenced to PWMTMR0 are clipped.

The effect latency from PWM_SYNC_IN to the PWM outputs is about three clock cycles in synchronous mode, and five clock cycles in asynchronous mode.

CAUTION: Do not change the value of the PWM_CTL.EXTSYNC bit while the PWM is enabled (PWM_CTL.GLOBEN =1).

Output Disable and Cross-Over Modes

Each PWM_ACTL channel control register contains separate enable bits for the high and low-side signals. The PWM module uses the PWM_ACTL.DISHI and PWM_ACTL.DISLO bits in the channel A control register to enable or disable the PWM_AH and PWM_AL outputs respectively. If the disable bit is set (=1), then the corresponding PWM output is disabled, irrespective of the value of the corresponding duty cycle register. This PWM output signal remains in the OFF state as long as the corresponding enable or disable bit is set.

The cross-over bit (PWM_ACTL.XOVR) allows programs to send the low-side output through the high-side output pin and the high-side output through the low-side output pin.

One example uses the following configuration.

- The PWM_AH0 register =0
- The PWM_CHANCFG.MODELSC bit =0
- The PWM_ACTL.DISLO bit =1
- The PWM_ACTL.XOVR bit =1.

The low-side output remains off, as in the case without crossover. The difference in cross-over is that the high-side output changes character and becomes like the low-side. What actually occurs is that the low-side duty cycle is sent to the high-side output pins, and the high-side duty cycle is sent to the low side pins. Because the PWM_ACTL.DISLO bit =1, the low-side pin remains off (see Output Control Feature Precedence).

The ***XOVR and DISHI/DISLO Functionality*** figure shows this example. In case 1, PWM_ACTL.XOVR =0; and in case 2, PWM_ACTL.XOVR =1.

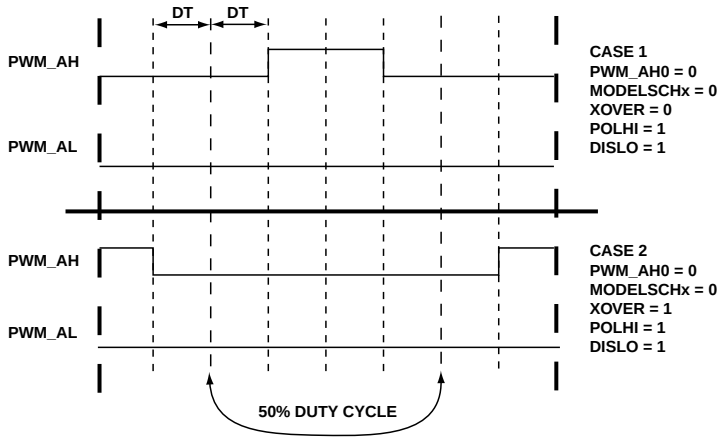

Figure 18-18: XOVR and DISHI/DISLO Functionality

Brushless DC Motor (Electronically Commutated Motor) Control

In the control of an electronically commutated motor (ECM), only two inverter legs are switched at any time. Often, the high-side device in one leg must be switched on at the same time as the low-side driver in a second leg. It is possible to turn on the high-side switch of phase A and the low-side switch of phase B at the same time by:

- Programming identical values for the duty cycles for two PWM channels (for example, PWM_CH0 = PWM_CH1), and
- Setting the PWM_BCTL.XOVR bit to crossover the BH and BL pair of PWM signals

To control ECM, normally the third inverter leg (phase C in this example) is disabled for a number of PWM cycles. To implement this function, both the PWM_CH and PWM_CL outputs are disabled by setting the PWM_CCTL.DISHI and PWM_CCTL.DISLO bits.

In normal ECM operation, each inverter leg is disabled for certain time periods so that the PWM channel registers change based on the position of the rotor shaft (motor commutation).

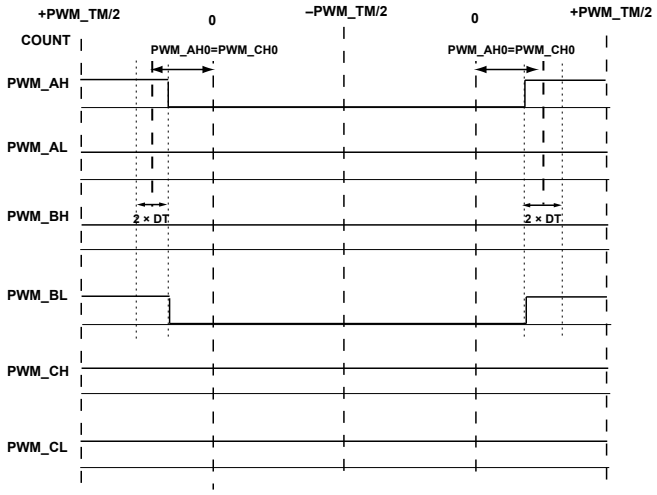

Figure 18-19: ECM Control

Trigger Slave Mode

Trigger slave mode uses the trigger slaves as either $\overline{\text{PWM\_TRIP[n]}}$ (default) or as PWM_SYNC. To configure this mode use the PADS_PCFG0 register to enable either of the paths of the TRU Slave outputs to choose either of the two functions PWM_SYNC or PWM_TRIP. Configure the PADS_PCFG0.PWM0_SYNC_TRU PADS_PCFG0.PWM1_SYNC_TRU and PADS_PCFG0.PWM2_SYNC_TRU bit fields to determine PWM slave functionality of PWM0, PWM1 and PWM2 respectively.

The ***PWM Trigger Slave Feature*** figure shows the PWM trigger slaves.

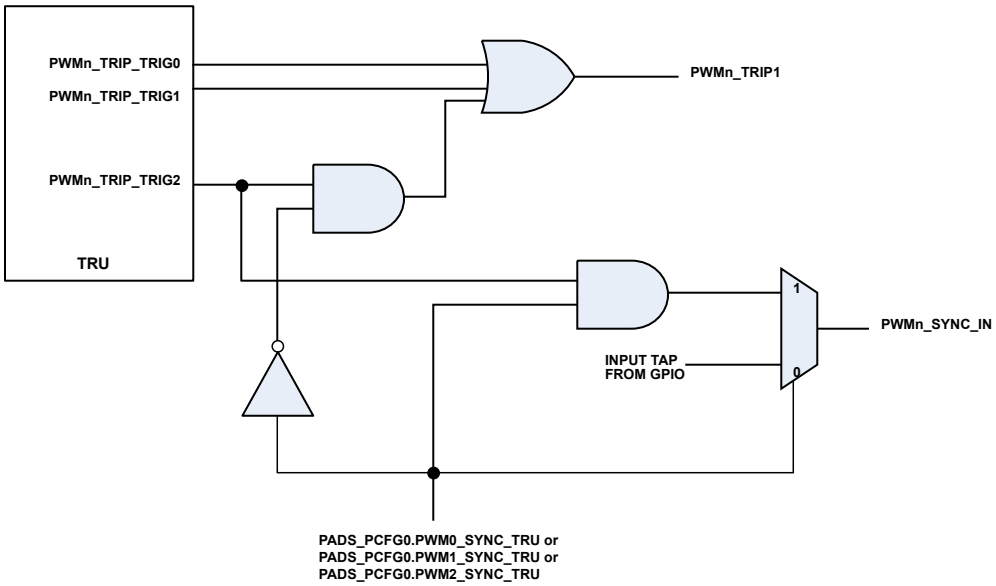

Figure 18-20: PWM Trigger Slave Feature

If the PADS_PCFG0.PWM0_SYNC_TRU bit =1, the PWM0_TRIP_TRIG2 slave trigger is routed to the PWM_SYNC pin as external sync pulse. The external sync pulse on PWM_SYNC pin is routed from PWM0_TRIP_TRIG2 slave trigger pulse (not from the GPIO pins). Also the PWM0_TRIP_TRIG2 slave trigger does not trip the PWM outputs.

Similarly, set the PADS_PCFG0.PWM1_SYNC_TRU bit =1 to route the PWM1_TRIP_TRIG2 slave trigger pulse to the PWM1_SYNC_IN pin as an external sync. Set the PADS_PCFG0.PWM2_SYNC_TRU bit =1 to route the PWM2_TRIP_TRIG2 slave trigger pulse to the PWM2_SYNC_IN pin as an external sync. By default, because the PADS_PCFG0 bit fields =0, the PWMn_TRIP_TRIG2 slave trigger is used to trip the PWM outputs and the external PWM_SYNC signal comes from the GPIO pins.

This feature can be used to synchronize PWM outputs of different PWM instances by configuring a single TRU master for all the instances. If the Trigger Routing Unit (TRU) is configured so that the PWM0_TRIP_TRIG2, PWM1_TRIP_TRIG2 and PWM2_TRIP_TRIG2 slave triggers have the same trigger master, then PWM0, PWM1 and PWM2 outputs can be synchronized.

With the PADS_PCFG0 bit fields =1, the PWMn_TRIP_TRIG2 slave trigger pulses are routed to the PWM_SYNC pins. All PWMs have the same trigger master and the TRU generates triggers for all PWMs at the same time when it receives trigger input from the master trigger. The external PWM_SYNC pulses are received at the same time for all the PWM instances when the TRU master generates the trigger. The PWM outputs are synchronized as the same PWM_SYNC signals are provided to all the PWM instance s.

Heightened-Precision Edge Placement

Heightened-precision edge placement allows a fine-grained edge placement within the system clock period. The Heightened-Precision Steps in a Single SYSCLK Period figure shows how the SYSCLK aligned edge is moved to finer resolution.

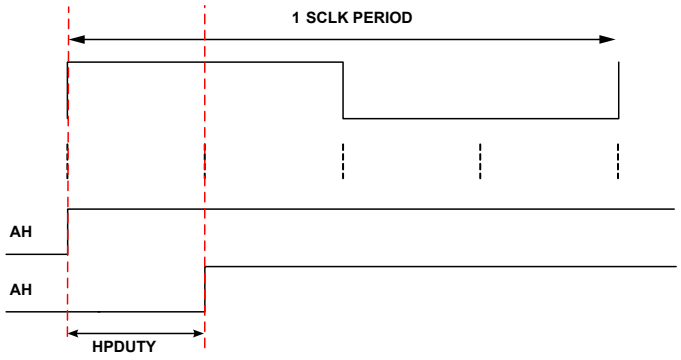

Figure 18-21: Heightened-Precision Steps in a Single SYSCLK Period

The ***Defining Fractional Duty Cycle*** figure shows an application with a duty cycle that corresponds to a fraction. The figure shows how the duty cycle is defined for the PWM based on the timer-base.

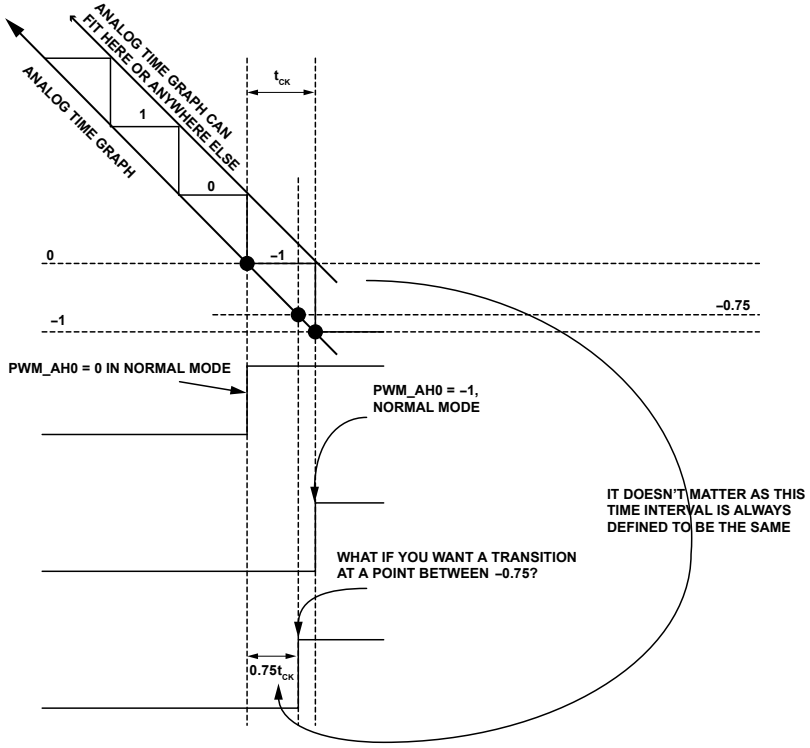

Figure 18-22: Defining Fractional Duty Cycle

Refer to the ***Defining Fractional Duty Cycle*** figure. An analog time graph is juxtaposed on the digital time graph which is represented by a cut-out of the PWM timer that channel A uses. The analog time graph is also shown to allow for better visualization. However, another graph can be used to get the same results.

In the example, in normal mode, the program can place an edge only at 0 and -1. Both points are separated by a time interval of one period of the peripheral clock, t_{CK}. This restriction limits the number of bits of resolution in the duty cycle that the program can control. In reality, the program may need to place an edge that is delayed from the zero-duty mark by a time interval of 0.75 t_{CK}.

This point is defined as -0.75. (By projecting the point back onto the analog time graph, the point is 3/4th of a unit separated from 0, towards -1 on this linear graph). The heightened-precision edge placement feature allows the program to specify such fractional duty cycles as -0.75, by providing more register bits to the channel duty registers (PWM_AH0). These bits are contained in the heightened-precision channel duty registers (PWM_AH0_HP.) The maximum time step precision that can be achieved is 2.5 ns.

A simple calculation provides the maximum fractional precision achievable for a particular clock frequency. An operational frequency of 100 MHz implies t_{CK} = 10 ns. Therefore, t_{CK}/2.5 ns = 4.

Since 4 is represented using 2 bits, the program can represent fractions up to 2 bits wide (required for certain applications) without a loss of precision.

The heightened-precision mode is enabled by setting the PADS_PCFG0_PWMGPSEL bit. The PWM_AH0_HP and PWM_AH1_HP registers work alongside the PWM_AH0 and PWM_AH1 registers to provide the overall resolution. The following example explains how signed decimal programming is implied for the heightened-precision duty values.

For the PWM_AH output, the duty-cycle register-pair PWM_AH0 and PWM_AH0_HP work together in a Q15.8 signed two's complement fixed-point format as shown in the ***Duty Cycle Notation for Heightened-Precision Edge Placement*** figure. The weight of bit position at k is 2k.

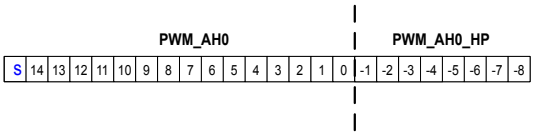

Figure 18-23: Duty Cycle Notation for Heightened-Precision Edge Placement

In the normal modes of operation (not involving heightened-precision edge placement), only the PWM_AH0 register value is programmed. The duty value programmed is a two's complement integer value. If a value of -1 is desired, the PWM_AH0 register is programmed with the number 0xFFFF (which is the two's complement of 1 in 16 bits).

In the heightened-precision mode, if a duty value corresponding to -0.75 is required, the equivalent two's complement value of -0.75 in the Q15.8 format is computed: 1111_1111_1111_1111.0100_0000 = 0xFFFF.40. In this case, the PWM_AH0 register is programmed to 0xFFFF and the PWM_AH0_HP register is programmed to 0x40.

• Heightened-Precision Edge Placement Example

The following is an example of heightened precision edge placement.

On the *positive side* of the fractional of the duty cycle, at 2 t_{CK} + 0.25 t_{CK}, the values for the PWM_AH0 and PWM_AH0_HP registers are calculated as follows.

The PWM_AH0 =0x0002 and PWM_AH0_HP =0x40 (bits 7:6).

The PWM_AH_DUTY0 register contains the bit fields from both the PWM_AH0 and PWM_AH0_HP registers. Bits [15:14] represent the decimal part or heightened-precision value and bits [31:16] represent the coarse duty cycle. The value for the combined registers is PWM_AH_DUTY0 =0x00024000.

On the *negative side* of the fractional of the duty cycle, at –2 t_{CK} – 0.25 t_{CK}, the values for the PWM_AH0 and PWM_AH0_HP registers are calculated as follows:

The coarse register represents the next count of the coarse value for negative values so that –2 becomes –3. These values are the two's complement of the positive offset (value = 3) PWM_AH0 = 0xFFFD and PWM_AH0_HP =0xC0 (bits 7:6).

To derive the correct format for a negative duty-cycle value, for example, –2.25, use: coarse value + 1 =3 for the coarse value and 1 for 0.25. Write out the absolute value as a 32-bit number first:

0000 0000 0000 0011 (.) 0100 0000 0000 0000

Then take the two's complement of the entire 32-bit number:

1111 1111 1111 1101 (.) 1100 0000 0000 0000

NOTE: This value is also written into the full duty register (PWM_AH_DUTY0). The correct value for the combined registers written in the PWM_AH_DUTY0 register is 0xFFFDC000.

Sample Waveforms for High- and Low-Side with Precision Placement

When the PWM module uses heightened-precision in the dependent mode of operation, both high and low-side outputs shift in the same direction. This operation can result in pulse-expansion of the high-side and pulse-contraction of the low-side or conversely.

The ***Output Shift in The Same Direction*** figure shows an example of a case with DT =1, and pulse expansion occurs on the high-side and pulse-contraction on the low-side. It juxtaposes a case where the PWM module does not use heightened-precision and a case where PWM does use it.

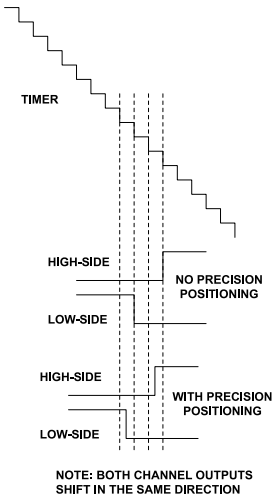

Figure 18-24: Output Shift in The Same Direction

The following ***Precision Placement –– PULSEMODE*** figures illustrate the cases for pulse modes 1, 2, and 3 (pulse mode 00 is a trivial case of pulse mode 01). The pulse modes are configured in the PWM_ACTL and PWM_BCTL

channel control registers. The figures show what happens to the edges as a decimal part is added to a programmed positive duty. In each case, assume that the original `PWM_AH0.DUTY` register value, which is the coarse duty value, changes to the `PWM_AH1.DUTY` value after programming the enhanced-resolution (`PWM_AH0_HP`, `PWM_AH1_HP`) registers. For example, changing 14 to 14.25 and 10 to 10.75. The figures are not drawn to these numbers. For negative duty values, the shifts are in the opposite direction.

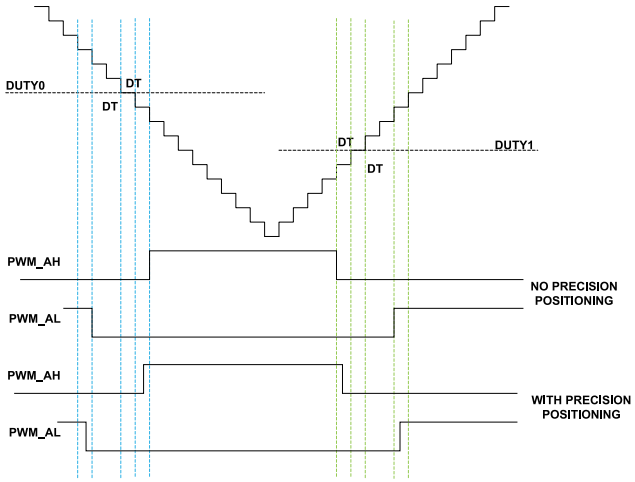

Figure 18-25: Precision Placement -– PULSEMODE=01

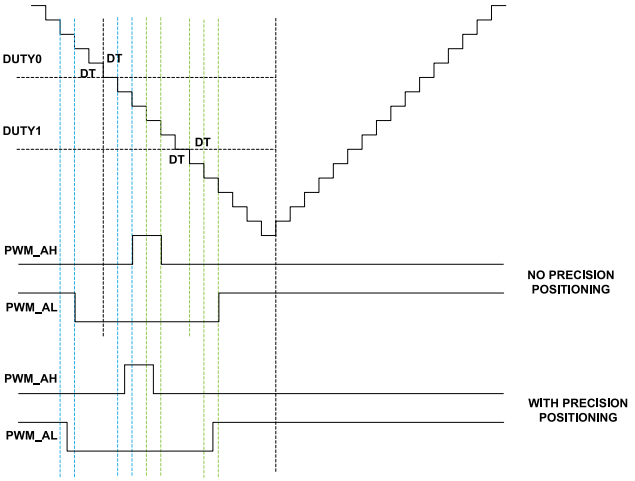

Figure 18-26: Precision Placement -– PULSEMODE=10

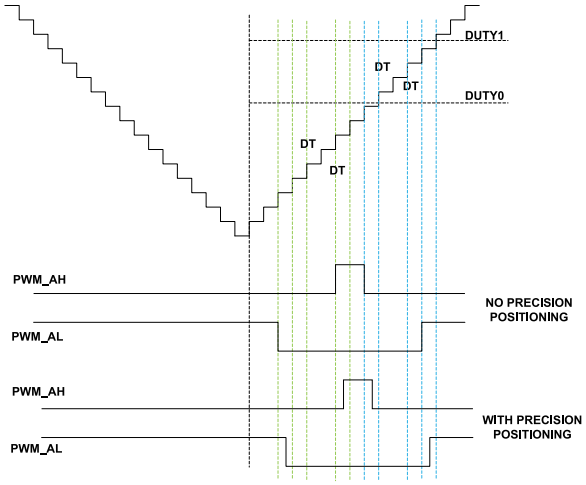

Figure 18-27: Precision Placement –– PULSEMODE=11

Emulation Mode

The PWM module can continue to operate or stop when entering halt from the emulator, based on the `PWM_CTL.EMURUN` bit setting.

- `PWM_CTL.EMURUN`=1. When the processor is halted in emulation mode, the outputs continue to toggle and be driven out of the PWM block. The counters and status register bits are set or reset according to the PWM TIMER-count or period settings.
- `PWM_CTL.EMURUN`=0. When the processor is halted in emulation mode, the outputs are shut down (enter their inactive state based on polarity), and all counters that affect status register bits are paused.

  The `PWM_STAT.EMU` bit is set.
- At restart, the PWM counters resume from their paused value. The outputs are still held in their inactive state.

  To reactivate the outputs, clear the `PWM_STAT.EMU` bit with a W1C operation.

NOTE: The `PWM_STAT.EMU` bit is not cleared by disabling the PWM or writing 0 to the `PWM_CTL.EMURUN` bit. Emergency dead time is not ensured on re-enabling the outputs by doing a W1C to the `PWM_STAT.EMU` bit. Sub SYSCLK heightened-precision edge placements can be off on the first output edge for every channel on clearing the `PWM_STAT.EMU` bit.

Event Control

The PWM uses bits in the `PWM_IMSK` and `PWM_ILAT` registers for event control. These registers allow masking and show masked interrupt status bits, respectively. The interrupt bits are latched and held on the interrupt event. The software must write a 1 to clear the interrupt bit, usually during the interrupt service routine.

The timer period (TMRxPER) interrupts are configured using the `PWM_ILAT.TMR0PER` - `PWM_ILAT.TMR4PER` bits. The PWM uses the interrupts to execute an interrupt service routine (ISR) periodically. The ISR updates the

PWM channel control and duty registers (according to a control algorithm based on expected operation and sampled existing operation). The TMRxPER interrupts also can trigger an ADC to sample data for use during the ISR.

The PWM uses the PWM_CTL.INTSYNCREF bit field to control the PWM_SYNC interrupt. The PWM uses the bit field to assign the interrupt to a user interrupt of the core. The PWM_SYNC can be configured to be either internal or externally driven using the PWM_CTL.EXTSYNC bit. When configured as an external sync, the signal can be further configured as synchronous or asynchronous using the PWM_CTL.EXTSYNCSEL bit.

As an example, when the PWM_SYNC interrupt occurs:

- The ADC samples data
- The data is algorithmically interpreted, and
- New PWM channel duties are calculated and written to the PWM

More sophisticated implementations include different startup, run time, and shutdown algorithms to determine PWM channel duties based on expected behavior and further features.

During the PWM_SYNC interrupt driven control loop, only the channel delay registers, the duty registers, and the channel C high pulse duty register values are typically updated. To see programming limitations on the PWM registers, see the *Register Descriptions* section.

Status information about the PWM is available in the PWM_STAT register, which stores all status bits, including raw interrupt status bits. In particular, the period boundary of each timer is available, as well as status bits. The PWM uses the status bits to indicate whether the operation is in the first half or the second half of the timer. Additionally, the TRIP status is also available. For more information on TRIP interrupts, see Trip Control Unit .

Trip Control Unit

The PWM output signals can be shut-off in a number of different ways. The trip inputs $\overline{\text{PWM\_TRIP[n]}}$ can be mapped to provide either a temporary or permanent shutdown on any pair of channel outputs. This shutdown mechanism is asynchronous so that the associated PWM output disable circuitry does not go through any clocked logic. This functionality ensures correct PWM shutdown even in the event of a loss of the processor clock. In addition to the hardware shutdown features, the PWM system can be shut down in software with the PWM_CTL.SWTRIP bit.

The external trip signals PWM0_TRIP0, PWM1_TRIP0 and PWM2_TRIP0 coming from GPIO will generate TRIP0 input whereas the trigger slaves PWM0_TRIP_TRIGn, PWM1_TRIP_TRIGn and PWM2_TRIP_TRIGn will generate TRIP1 input.

During any external trip event (if not disabled), the PWM outputs are turned off. When a PWM output is turned off, it means that the output level is held at a polarity opposite that given in the PWM_CHANCFG.POLxH bits. The PWM sync pulse continues to operate, when it is already enabled. A PWMTRIP interrupt occurs if unmasked, to notify the software of this event. In dependent mode of operation, both high and low-side outputs refer to the PWM_CHANCFG.POLxH bit.

Even if the clock to the PWM is damaged, an external trip event turns off the PWM outputs. But, the PWMTRIP interrupt may not occur.

The PWM trip unit processes hardware or software fault conditions and shuts down the PWM channel outputs immediately on the occurrence of these conditions. The PWM can enable shutdown mechanism separately for each channel. The design also allows for a self-restart mechanism to be enabled on a channel. Self-restart re-enables the channel outputs following the fault condition (allowed only on hardware trips) when the PWMTMRy that the channel is using reaches its period boundary.

There are two external hardware sources that can indicate a hardware fault condition:

1. $\overline{\texttt{PWM\_TRIP0}}$ input pin
2. $\overline{\texttt{PWM\_TRIP1}}$ input pin

These sources are active low inputs where a falling edge on either of these pins indicates a fault condition.

The trip unit can shut down an output of a particular channel in response to the fault event on either of these $\overline{\texttt{PWM\_TRIP[n]}}$ pins. To enable this functionality, program the PWM_TRIPCFG.EN0A bit corresponding to the channel.

Program the PWM_TRIPCFG.MODE0A bits to specify the restart mechanism for a channel that has been tripped.

1. If the PWM_TRIPCFG.MODE0A bit =0, once tripped, a trip condition is registered on this channel in the PWM_STAT.FLTTRIPA bit and the outputs of that channel are immediately shut down. This condition is called a *fault trip* condition. To resume channel output when a fault trip occurs, write a 1 to clear the PWM_STAT.FLTTRIPA bit. A processor write cannot clear the bit when the trip condition is still active. The raw trip status is available for both pins in the PWM_STAT.RAWTRIP0 register bits.
2. If the PWM_TRIPCFG.MODE0A bit =1, once tripped, a trip condition is registered on this channel in the PWM_STAT.SRTRIPA bit and the outputs of that channel are immediately shut down. This condition is called a *self-restart trip* condition. If the trip condition is not active at the next period boundary of the PWMTMRy that the channel is using, the status register bit is cleared. The outputs are restored.

The trip input pins have an external pull-down resistor on the chip pin. If the pin becomes disconnected, the processor disables PWM.

In addition to the hardware trip conditions, a global software trip bit in the PWM_CTL register allows for a software-forced fault trip condition. When the global software trip bit is set to 1, irrespective of the values in the PWM_TRIPCFG register, it sets all the PWM_STAT.FLTTRIPA bits and also gates the channel outputs. To remove the trip condition from the channel, perform a W1C on the PWM_STAT.FLTTRIPA bit of the particular channel.

If the PWM_TRIPCFG.EN0A bit is set to 1 to, for any channel, then the occurrence of a fault condition on the PWMTRIPy bit is logged in the PWM_STAT.FLTTRIPA register bit. If the corresponding PWM_IMSK.TRIP0 bit = 1, then an interrupt is generated. Tripping a channel output does not interfere with PWM_SYNC generation.

The ***Operation Under Hardware Fault Conditions*** figure shows an example where PWMTRIP0 is enabled on channel A as self-restart trip. Channel A works with the PWM_CHANCFG.POLAH bit =1. In period 2, the PWM_AH signal is full on modulated, and tries to rise at the period boundary where the self-restart occurs for the channel. However, since the low-side output of the channel was only recently removed due to a trip, the rise edge on PWM_AH is delayed until the emergency dead-time period is over. PWMTRIP1 is enabled on channel B as a fault trip. Channel B works

with the PWM_CHANCFG.POLAH bit =0. PWMTRIP1 stays low for an extended time period. The first processor write to reenable the channel output fails. The second processor write passes since the fault condition has gone away.

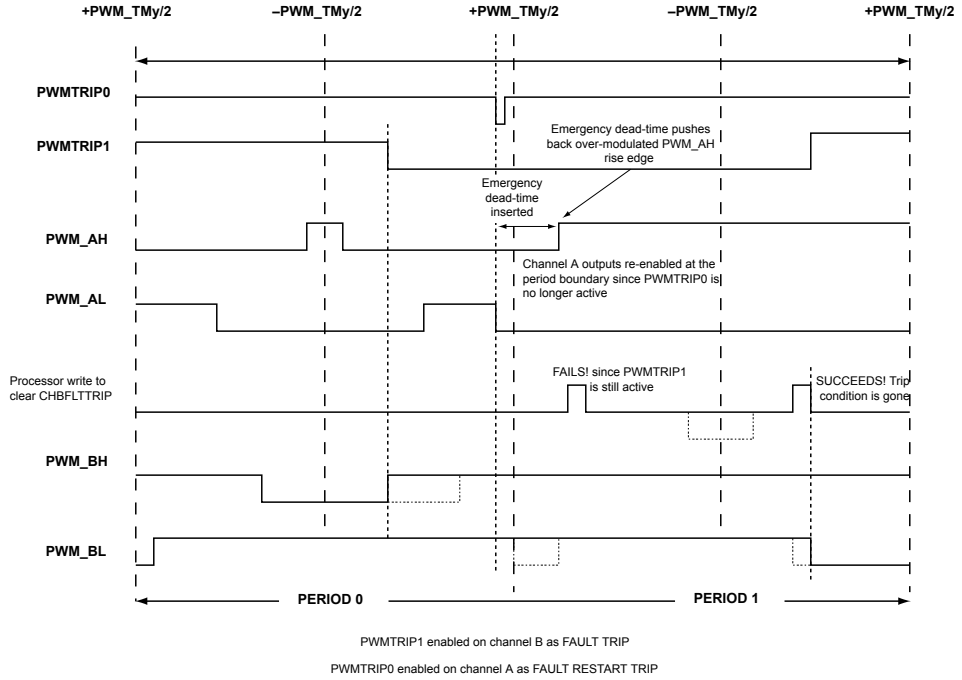

Figure 18-28: Operation Under Hardware Fault Conditions

NOTE: Dead time is ensured on re-enabling the channel outputs after trip.

NOTE: Programs must not allow changes in the configuration or enable bits of PWM_TRIPCFG register within ± 10 clock cycles of when the external trip pulse toggles.(The configuration or enable bits of PWM_TRIPCFG register select between trip enable and disable). If this time frame is not followed, then unexpected behavior occurs.

Programming Model

The following sections provide general (and some application-specific) programming steps for configuring and using the PWM module.

- Programming Model for 3-Phase AC Motor Control

Programming Model for 3-Phase AC Motor Control

The ***PWM Module and Interaction with System*** figure shows how the PWM unit (green) interfaces to both software (blue) and hardware (yellow). The software configures the unit, calculates duty cycles (Duty A, Duty B, Duty C), and services the interrupts generated by the module (PWM Sync IRQ, TRIP IRQ). The hardware applies the gate signals (AH, AL, BH, BL, CH, CL) to the inverter and provides an over-current trip signal back to the unit (TRIP0).

The typical 3-phase AC motor configuration shown in the ***PWM Module and Interaction with System*** figure applies for both permanent magnet and induction motor types.

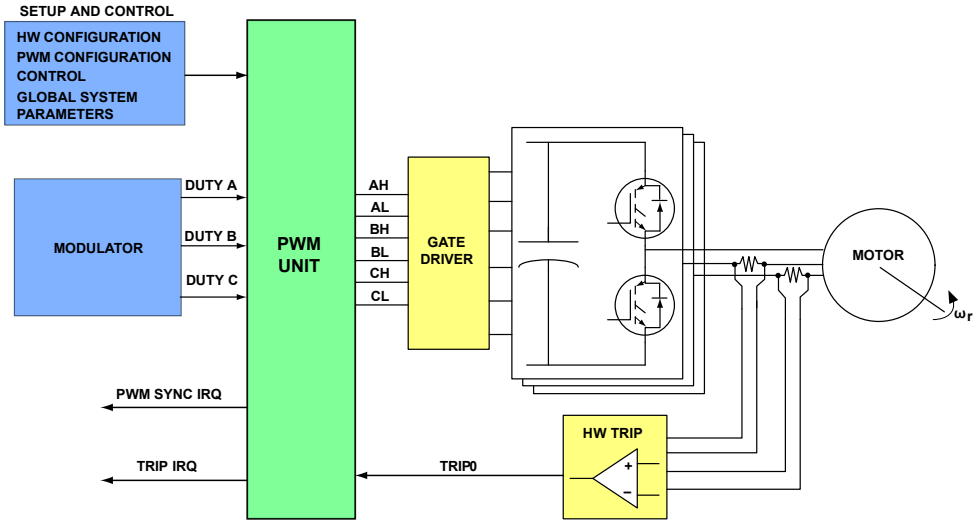

Figure 18-29: PWM Module and Interaction with System

System Parameters

The following system parameters (characteristics) influence the module configuration for this application. This example system features:

- One 3-phase AC machine
- B6 inverter
- SVPWM, including both linear- and over-modulation
- Switching frequency of 20 kHz
- Dead time of 1us
- Trip signal generated by hardware
- Active high-level gate drive
- Core frequency of 200 MHz
- Peripheral clock of 100 MHz

System State Sequencing

Managing the system state and sequence of states is critically important when programming the PWM module. The ***PWM System States*** figure provides an overview of these states.

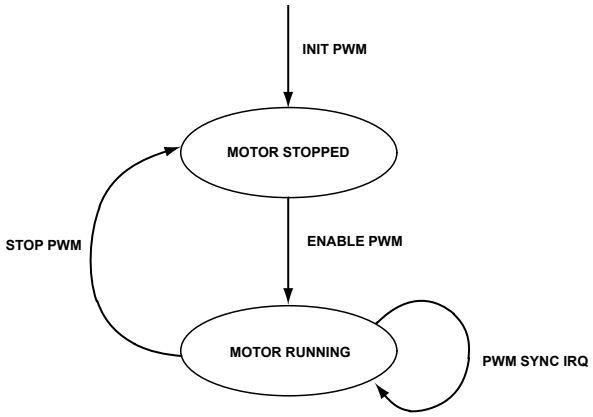

Figure 18-30: PWM System States

As shown in the state diagram, the module configuration is updated on state transitions (indicated by the arrows). The transitions are initialization, motor start, PWM sync interrupt (on each), and motor stop. The following sections discuss the transitions in detail.

- PWM Initialization for Motor Control
- PWM Enable for Motor Control
- PWM Response to Sync Interrupt for Motor Control
- PWM Disable (and Stop the Motor) for Motor Control

PWM Initialization for Motor Control

The processor must program the PWM at power-up as follows and repeat this programming to bring the PWM and the system into a known (safe) state.

1. Place the PWM module in a safe state and set up synchronization of the module using the following bitwise operations on the `PWM_CTL` and `PWM_CHANCFG` registers:

   ```
   PWM_CTL &= 0xFFE0FF08
   PWM_CTL |= 0x20000
   PWM_CHANCFG &= 0x80808080
   PWM_CHANCFG |= 0x24242424
   ```

   These operations result in the following bit settings:

   - Disable PWM (`PWM_CTL.GLOBEN` =0)
   - Disable delay for channels A, B, C, D (`PWM_CTL.DLYAEN` through `PWM_CTL.DLYDEN` =0). All phases must run with same phase.
   - Use internal synchronization by timer TMR0 (`PWM_CTL.EXTSYNC` =0, `PWM_CTL.EXTSYNCSEL` =1)
   - The same timer, TMR0 (`PWM_CTL.INTSYNCREF` = b#000) synchronizes all phases.
   - Low-side is always the inverse of high-side (`PWM_CHANCFG.POLAL` through `PWM_CHANCFG.POLDL` = 1)

- System uses active high gate driver (PWM_CHANCFG.ENCHOPAH through PWM_CHANCFG.ENCHOPDH =1)
- Disable gate chopping (PWM_CHANCFG.ENCHOPAL through PWM_CHANCFG.ENCHOPDL =0). PWM does not use the pulse transformer.

2. Set up the trip and associated interrupts using the following bitwise operations on the PWM_TRIPCFG and PWM_ILAT registers:

```
PWM_TRIPCFG &= 0xF0F0F0F0
PWM_TRIPCFG |= 0x1010101
PWM_ILAT &= 0xFFE0FFFC
PWM_ILAT |= 0x1
```

These operations result in the following bit settings:

- All phases must shut down simultaneously in case of fault: (PWM_TRIPCFG.EN0A through PWM_TRIPCFG.EN0D =0, PWM_TRIPCFG.MODE0A through PWM_TRIPCFG.MODE0D =0, PWM_TRIPCFG.EN1A through PWM_TRIPCFG.EN1D =0, PWM_TRIPCFG.MODE1A through PWM_TRIPCFG.MODE1D =0)
- Enable TRIP0 as fault trigger for all channels (PWM_TRIPCFG.EN0A through PWM_TRIPCFG.MODE1D =1).
- For thermal control and synchronization, SW intervention is needed at trip. Do not use automatic restart of any channels.
- Generate an interrupt at trip on TRIP0 (PWM_ILAT.TMR0PER = 1).

3. Configure the PWM channels using the following bitwise operations on the PWM_TRIPCFG and PWM_ILAT registers:

```
PWM_CHA_DT=0x32 PWM_CHB_DT=0x32 PWM_CHC_DT=0x32
PWM_TM0 = 0x9C4
PWM_ACTL = 0xFFFFF0000
PWM_BCTL = 0xFFFFF0000
PWM_CCTL = 0xFFFFF0000
PWM_AH0 = 0x0
PWM_BH0 = 0x0
PWM_CH0 = 0x0
```

These operations result in the following bit settings:

- Configure a dead time of 1 µs (DT =0x32 =0x32).
- Configure a PWM frequency of 20 kHz (PWM_TM0 =0x9C4).
- Disable all outputs (PWM_ACTL.DISHI through PWM_CCTL.DISHI =0, PWM_ACTL.DISLO through PWM_CCTL.DISLO =0)
- Use conventional PWM, disable crossover (PWM_ACTL.XOVR through PWM_CCTL.XOVR =0)

- Use symmetrical pulse position on all outputs (PWM_ACTL.PULSEMODEHI through PWM_CCTL.PULSEMODEHI =0, PWM_ACTL.PULSEMODELO through PWM_CCTL.PULSEMODELO =0)
- Set an initial duty-cycle of 50% (PWM_AH0 throughPWM_CH0 =0x0)

PWM Enable for Motor Control

The processor must do the following programming to enable the PWM before starting the motor.

1. Start the PWM timer TMR0 using the PWM_CTL |=0x1 bitwise operation on the PWM_CTL register.

   ADDITIONAL INFORMATION: This operation has the same effect as setting the PWM_CTL.GLOBEN bit =1.

2. Enable six PWM outputs using the following bitwise operations on the PWM_ACTL through PWM_CCTL registers.

   ```
   PWM_ACTL|=0x3
   PWM_BCTL|=0x3
   PWM_CCTL|=0x3
   ```

   ADDITIONAL INFORMATION: These operations have the same effect as enabling high and low-side channel outputs by setting the PWM_ACTL.DISHI through PWM_CCTL.DISHI bits =1 and the PWM_ACTL.DISLO through PWM_CCTL.DISLO bits =1.

3. Enable the PWM TRIP0 interrupt using the PWM_ILAT |= 0x1 bitwise operation on the PWM_ILAT register.

   ADDITIONAL INFORMATION: This operation has the same effect as setting the PWM_ILAT.TRIP0 bit =1

PWM Response to Sync Interrupt for Motor Control

When the PWM sync interrupt occurs, the processor could need to update to the PWM duty cycle with a value calculated by the motor control algorithm. This application uses symmetric pulses position and uses dependent high and low-side output. So, the PWM updates only one register for each phase.

1. Write the new duty cycle value (calculated by motor control algorithm) to the timer when the sync interrupt occurs.

   The following bitwise operations on the PWM_AH0 through PWM_CH0 registers accomplish this task:

   ```
   PWM_AH0 = Duty_A_mc_algorithm_current_value
   PWM_BH0 = Duty_B_mc_algorithm_current_value
   PWM_CH0 = Duty_C_mc_algorithm_current_value
   ```

PWM Disable (and Stop the Motor) for Motor Control

The processor must program the PWM as follows to stop the motor, disable the PWM, and disable PWM interrupts. These actions place the PWM and system in a safe, passive state.

1. Disable the PWM timer using the PWM_CTL &= 0xFFFFFFFE bitwise operation on the PWM_CTL register.

ADDITIONAL INFORMATION: This operation has the same effect as clearing the PWM_CTL.GLOBEN bit =0.

2. Disable all PWM outputs using the following bitwise operations on the PWM_ACTL through PWM_CCTL registers.

```
PWM_ACTL &= 0xFFFFFFFC
PWM_BCTL &= 0xFFFFFFFC
PWM_CCTL &= 0xFFFFFFFC
```

   These operations disable PWM outputs where the PWM_ACTL.DISHI through PWM_CCTL.DISHI bits =1 and the PWM_ACTL.DISLO through PWM_CCTL.DISLO bits =0)

3. Set the PWM duty-cycle to 50% using the following bitwise operations on the PWM_AH0 through PWM_CH0 registers.

```
PWM_AH0 =0x0
PWM_BH0 =0x0
PWM_CH0 =0x0
```

   These operations have the same effect as clearing the PWM_AH0.DUTY through PWM_CH0.DUTY bit =0.

4. Disable the PWM TRIP0 interrupt using the PWM_ILAT &= 0xFFFFFFFE bitwise operation on the PWM_ILAT register.

   ADDITIONAL INFORMATION: This operation has the same effect as clearing the PWM TRIP0 interrupt PWM_ILAT.TRIP0 =0.

ADSP-CM40x PWM Register Descriptions

Pulse-Width Modulator (PWM) contains the following registers.

Table 18-5: ADSP-CM40x PWM Register List

Name	Description
PWM_ACTL	Channel A Control Register
PWM_AH0	Channel A-High Duty-0 Register
PWM_AH0_HP	Channel A-High Heightened-Precision Duty-0 Register
PWM_AH1	Channel A-High Duty-1 Register
PWM_AH1_HP	Channel A-High Heightened-Precision Duty-1 Register
PWM_AH_DUTY0	Channel A-High Full Duty0 Register
PWM_AH_DUTY1	Channel A-High Full Duty1 Register
PWM_AL0	Channel A-Low Duty-0 Register
PWM_AL0_HP	Channel A-Low Heightened-Precision Duty-0 Register
PWM_AL1	Channel A-Low Duty-1 Register
PWM_AL1_HP	Channel A-Low Heightened-Precision Duty-1 Register

Table 18-5: ADSP-CM40x PWM Register List (Continued)

Name	Description
PWM_AL_DUTY0	Channel A-Low Full Duty0 Register
PWM_AL_DUTY1	Channel A-Low Full Duty1 Register
PWM_BCTL	Channel B Control Register
PWM_BH0	Channel B-High Duty-0 Register
PWM_BH0_HP	Channel B-High Heightened-Precision Duty-0 Register
PWM_BH1	Channel B-High Duty-1 Register
PWM_BH1_HP	Channel B-High Heightened-Precision Duty-1 Register
PWM_BH_DUTY0	Channel B-High Full Duty0 Register
PWM_BH_DUTY1	Channel B-High Full Duty1 Register
PWM_BL0	Channel B-Low Duty-0 Register
PWM_BL0_HP	Channel B-Low Heightened-Precision Duty-0 Register
PWM_BL1	Channel B-Low Duty-1 Register
PWM_BL1_HP	Channel B-Low Heightened-Precision Duty-1 Register
PWM_BL_DUTY0	Channel B-Low Full Duty0 Register
PWM_BL_DUTY1	Channel B-Low Full Duty1 Register
PWM_CCTL	Channel C Control Register
PWM_CH0	Channel C-High Pulse Duty Register 0
PWM_CH0_HP	Channel C-High Pulse Heightened-Precision Duty Register 0
PWM_CH1	Channel C-High Pulse Duty Register 1
PWM_CH1_HP	Channel C-High Pulse Heightened-Precision Duty Register 1
PWM_CHANCFG	Channel Configuration Register
PWM_CHOPCFG	Chop Configuration Register
PWM_CH_DUTY0	Channel C-High Full Duty0 Register
PWM_CH_DUTY1	Channel C-High Full Duty1 Register
PWM_CL0	Channel C-Low Pulse Duty Register 0
PWM_CL0_HP	Channel C-Low Pulse Duty Register 1
PWM_CL1	Channel C-Low Duty-1 Register
PWM_CL1_HP	Channel C-Low Heightened-Precision Duty-1 Register
PWM_CL_DUTY0	Channel C-Low Full Duty0 Register
PWM_CL_DUTY1	Channel C-Low Full Duty1 Register
PWM_CTL	Control Register

Table 18-5: ADSP-CM40x PWM Register List (Continued)

Name	Description
PWM_DCTL	Channel D Control Register
PWM_DH0	Channel D-High Duty-0 Register
PWM_DH0_HP	Channel D-High Pulse Heightened-Precision Duty Register 0
PWM_DH1	Channel D-High Pulse Duty Register 1
PWM_DH1_HP	Channel D High Pulse Heightened-Precision Duty Register 1
PWM_DH_DUTY0	Channel D-High Full Duty0 Register
PWM_DH_DUTY1	Channel D-High Full Duty1 Register
PWM_DL0	Channel D-Low Pulse Duty Register 0
PWM_DL0_HP	Channel D-Low Heightened-Precision Duty-0 Register
PWM_DL1	Channel D-Low Pulse Duty Register 1
PWM_DL1_HP	Channel D-Low Heightened-Precision Duty-1 Register
PWM_DLYA	Channel A Delay Register
PWM_DLYB	Channel B Delay Register
PWM_DLYC	Channel C Delay Register
PWM_DLYD	Channel D Delay Register
PWM_DL_DUTY0	Channel D-Low Full Duty0 Register
PWM_DL_DUTY1	Channel D-Low Full Duty1 Register
PWM_DT	Dead Time Register
PWM_ILAT	Interrupt Latch Register
PWM_IMSK	Interrupt Mask Register
PWM_STAT	Status Register
PWM_SYNC_WID	Sync Pulse Width Register
PWM_TM0	Timer 0 Period Register
PWM_TM1	Timer 1 Period Register
PWM_TM2	Timer 2 Period Register
PWM_TM3	Timer 3 Period Register
PWM_TM4	Timer 4 Period Register
PWM_TRIPCFG	Trip Configuration Register

Channel A Control Register

The PWM_ACTL register selects the low and high side output pulse mode, enables low and high side output, and enables low/high side output crossover.

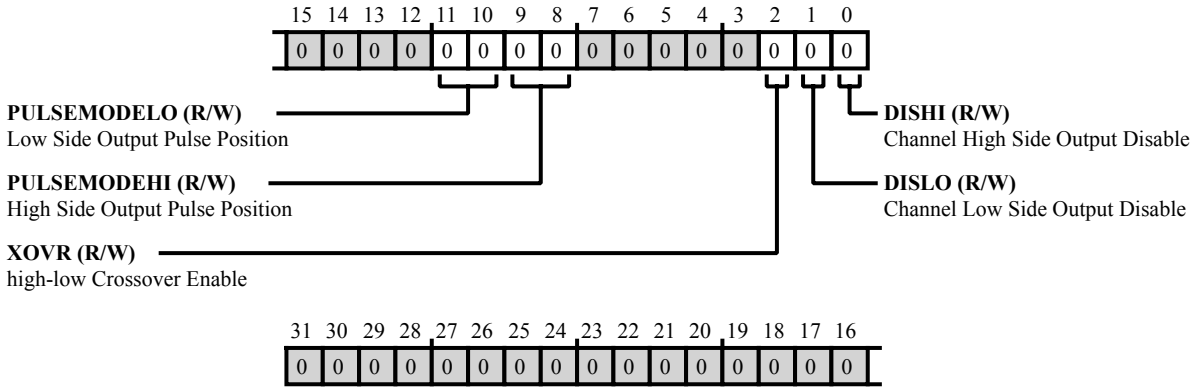

Figure 18-31: PWM_ACTL Register Diagram

Table 18-6: PWM_ACTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/W)	PULSEMODELO	Low Side Output Pulse Position. The PWM_ACTL.PULSEMODELO bits select the pulse position for Channel A low side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_AL0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (PWM_AL0 and PWM_AL1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (PWM_AL0 and PWM_AL1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half

Table 18-6: PWM_ACTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/W)	PULSEMODEHI	High Side Output Pulse Position. The PWM_ACTL.PULSEMODEHI bits select the pulse position for Channel A high side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_AH0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (PWM_AH0 and PWM_AH1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (PWM_AH0 and PWM_AH1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half
2 (R/W)	XOVR	high-low Crossover Enable. The PWM_ACTL.XOVR bit enables crossover between the channels high and low side outputs. When enabled, this bit directs the PWM to send the low-side output through the high-side output pin and the high-side output through the low side output pin.	
		0	Disable Crossover
		1	Enable Crossover
1 (R/W)	DISLO	Channel Low Side Output Disable. The PWM_ACTL.DISLO bit enables the channels low side output.	
		0	Enable Low Side Output
		1	Disable Low Side Output
0 (R/W)	DISHI	Channel High Side Output Disable. The PWM_ACTL.DISHI bit enables the channels high side output.	
		0	Enable High Side Output
		1	Disable High Side Output

Channel A-High Duty-0 Register

The PWM_AH0 and PWM_AH1 registers determine the width for the high side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the high side output pulses for the Channel A duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_ACTL.PULSEMODEHI bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_AH0 register to determine the assertion and de-assertion count for the high side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts Channel A high pulse output for count less than PWM_AH0 and de-asserts this output for count greater than PWM_AH1.

The value range for the PWM_AH0 and PWM_AH1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_AH0 and PWM_AH1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_DT (two's complement) to +PWM_TM0/2 + PWM_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_DT (two's complement) to -PWM_TM0/2 - PWM_DT.

Note that using values in the PWM_AH0 or PWM_AH1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

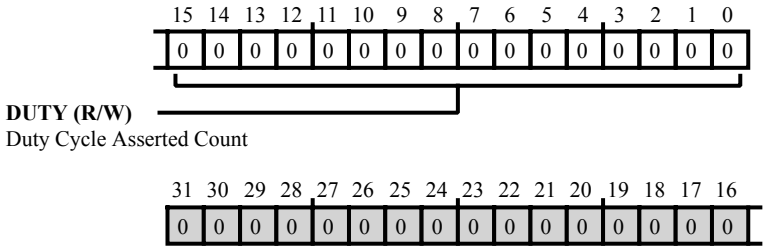

Figure 18-32: PWM_AH0 Register Diagram

Table 18-7: PWM_AH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_AH0.DUTY bits select the duty cycle asserted count for Channel A high side output.

Channel A-High Heightened-Precision Duty-0 Register

The PWM_AH0_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_AH0 register, allows programs to specify fractional duty cycles.The PWM_AH0_HP register and the PWM_AH0 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_AH0_HP and the PWM_AH0 registers are also present in the single full duty register (if available).

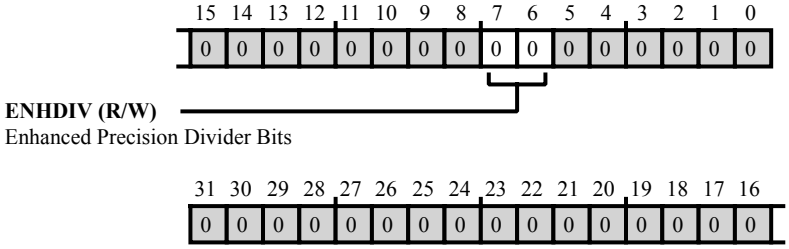

Figure 18-33: PWM_AH0_HP Register Diagram

Table 18-8: PWM_AH0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits.

Channel A-High Duty-1 Register

The PWM_AH0 and PWM_AH1 registers determine the width for the high side output pulses. For more information, see the PWM_AH0 register description.

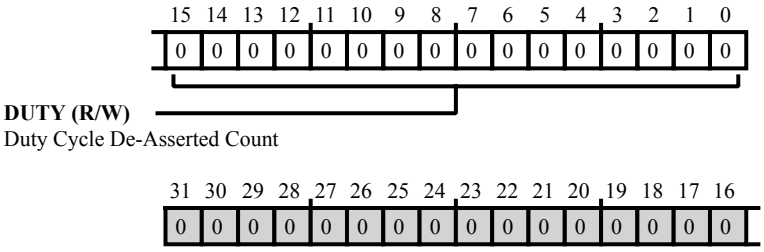

Figure 18-34: PWM_AH1 Register Diagram

Table 18-9: PWM_AH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The PWM_AH1.DUTY bits select the duty cycle de-asserted count for Channel A high side output.

Channel A-High Heightened-Precision Duty-1 Register

The PWM_AH1_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_AH0 register, allows programs to specify fractional duty cycles.The PWM_AH1_HP register and the PWM_AH1 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_AH1_HP and the PWM_AH1 registers are also present in the single full duty register (if available).

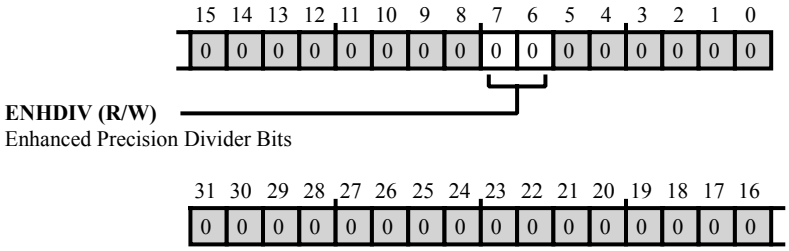

Figure 18-35: PWM_AH1_HP Register Diagram

Table 18-10: PWM_AH1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_AH1_HP.ENHDIV bits provide fractional duty cycles for Channel A high side output.

Channel A-High Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_AH_DUTY0 register contains the PWM_AH_DUTY0.DUTY bit field from the PWM_AH0 register and the PWM_AH_DUTY0.ENHDIV bit field from the PWM_AH0_HP register.

Note that the PWM_AH_DUTY0 register reads the PWM_AH0 and the PWM_AH0_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

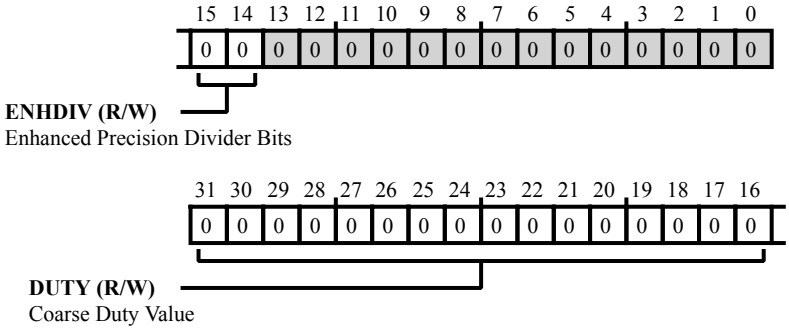

Figure 18-36: PWM_AH_DUTY0 Register Diagram

Table 18-11: PWM_AH_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_AH_DUTY0.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_AH_DUTY0.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_AH_DUTY0.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel A-High Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_AH_DUTY1 register contains the PWM_AH_DUTY1.DUTY bit field from the PWM_AH1 register and the PWM_AH_DUTY1.ENHDIV bit field from the PWM_AH1_HP register.

Note that the PWM_AH_DUTY1 register reads the PWM_AH1 and the PWM_AH1_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

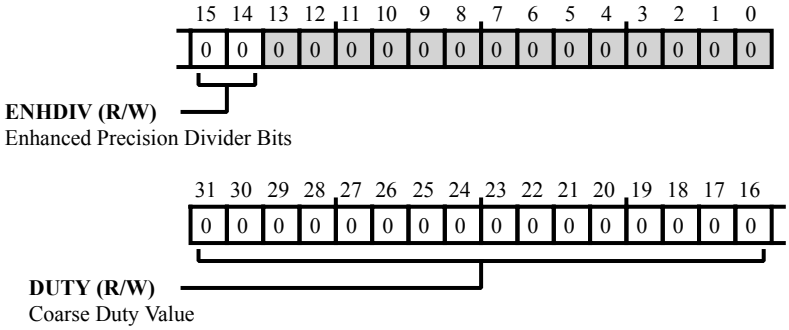

Figure 18-37: PWM_AH_DUTY1 Register Diagram

Table 18-12: PWM_AH_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_AH_DUTY1.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_AH_DUTY1.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_AH_DUTY1.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel A-Low Duty-0 Register

The PWM_AL0 and PWM_AL1 registers determine the width for the low side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the low side output pulses for the Channel A duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_ACTL.PULSEMODELO bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_AL0 register to determine the assertion and de-assertion count for the low side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts Channel A low pulse output for count less than PWM_AL0 and de-asserts this output for count greater than PWM_AL1.

The value range for the PWM_AL0 and PWM_AL1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_AL0 and PWM_AL1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_DT (two's complement) to +PWM_TM0/2 + PWM_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_DT (two's complement) to -PWM_TM0/2 - PWM_DT.

Note that using values in the PWM_AL0 or PWM_AL1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

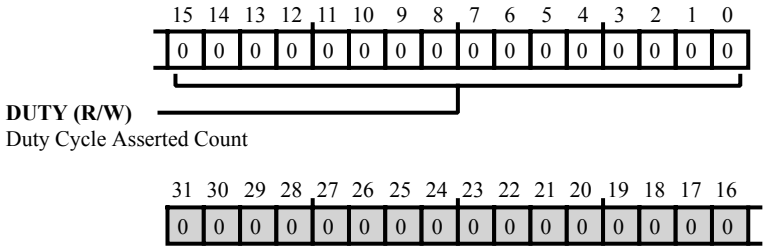

Figure 18-38: PWM_AL0 Register Diagram

Table 18-13: PWM_AL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_AL0.DUTY bits select the duty cycle asserted count for Channel A low side output.

Channel A-Low Heightened-Precision Duty-0 Register

The PWM_AL0_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_AL0 register, allows programs to specify fractional duty cycles.The PWM_AL0_HP register and the PWM_AL0 register work together in a Q15.8 signed two's complement fixed-point format. Note that the bit fields in the PWM_AL0_HP and the PWM_AL0 registers are also present in the single full duty register (if available).

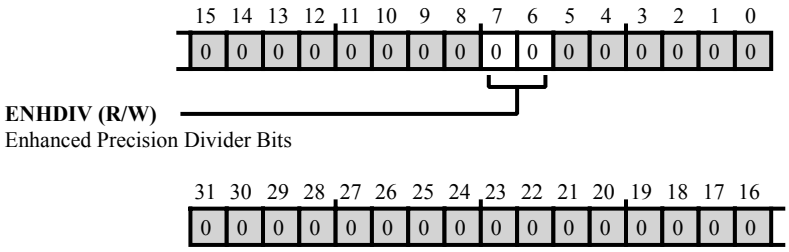

Figure 18-39: PWM_AL0_HP Register Diagram

Table 18-14: PWM_AL0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_AL0_HP.ENHDIV bits provide fractional duty cycles for Channel A low side output.

Channel A-Low Duty-1 Register

The PWM_AL0 and PWM_AL1 registers determine the width for the low side output pulses. For more information, see the PWM_AL0 register description.

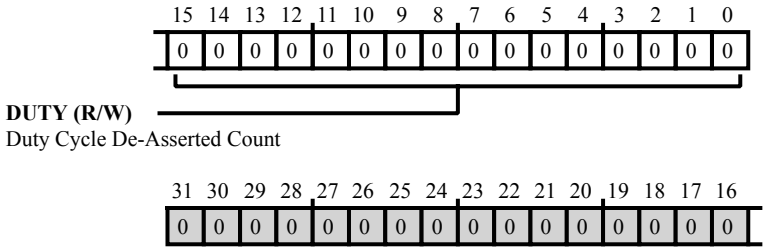

Figure 18-40: PWM_AL1 Register Diagram

Table 18-15: PWM_AL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The PWM_AL1.DUTY bits select the duty cycle de-asserted count for Channel A low side output.

Channel A-Low Heightened-Precision Duty-1 Register

The PWM_AL1_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_AL1 register, allows programs to specify fractional duty cycles.The PWM_AL1_HP register and the PWM_AL1 register work together in a Q15.8 signed two's complement fixed-point format. Note that the bit fields in the PWM_AL1_HP and the PWM_AL1 registers are also present in the single full duty register (if available).

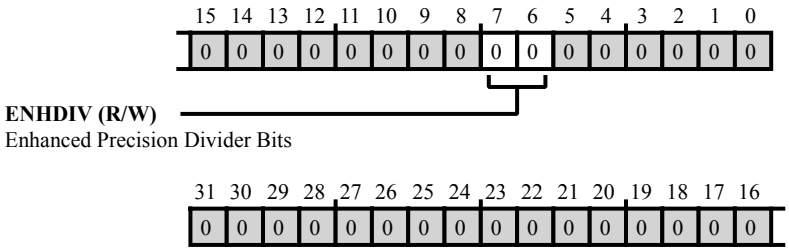

Figure 18-41: PWM_AL1_HP Register Diagram

Table 18-16: PWM_AL1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_AL1_HP.ENHDIV bits provide fractional duty cycles for Channel A low side output.

Channel A-Low Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_AL_DUTY0 register contains the PWM_AL_DUTY0.DUTY bit field from the PWM_AL0 register and the PWM_AL_DUTY0.ENHDIV bit field from the PWM_AL0_HP register.

Note that the PWM_AL_DUTY0 register reads the PWM_AL0 and the PWM_AL0_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

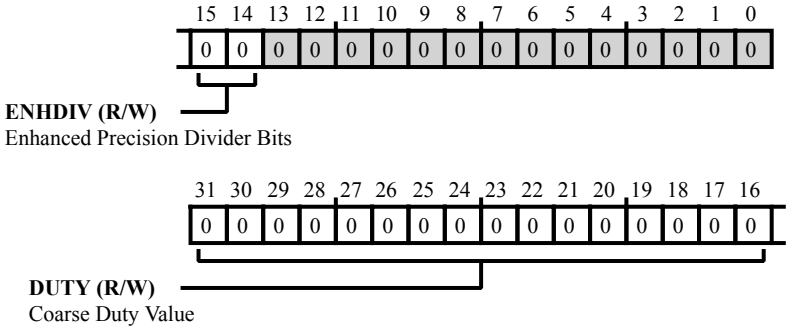

Figure 18-42: PWM_AL_DUTY0 Register Diagram

Table 18-17: PWM_AL_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_AL_DUTY0.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_AL_DUTY0.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_AL_DUTY0.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel A-Low Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_AL_DUTY1 register contains the PWM_AL_DUTY1.DUTY bit field from the PWM_AL1 register and the PWM_AL_DUTY1.ENHDIV bit field from the PWM_AH0_HP register.

Note that the PWM_AL_DUTY1 register reads the PWM_AL1 and the PWM_AL1_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

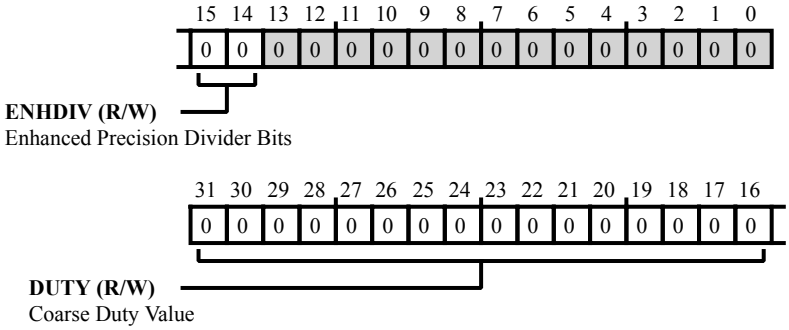

Figure 18-43: PWM_AL_DUTY1 Register Diagram

Table 18-18: PWM_AL_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_AL_DUTY1.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_AL_DUTY1.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_AL_DUTY1.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel B Control Register

The PWM_BCTL register selects the low and high side output pulse mode, enables low and high side output, and enables low/high side output crossover.

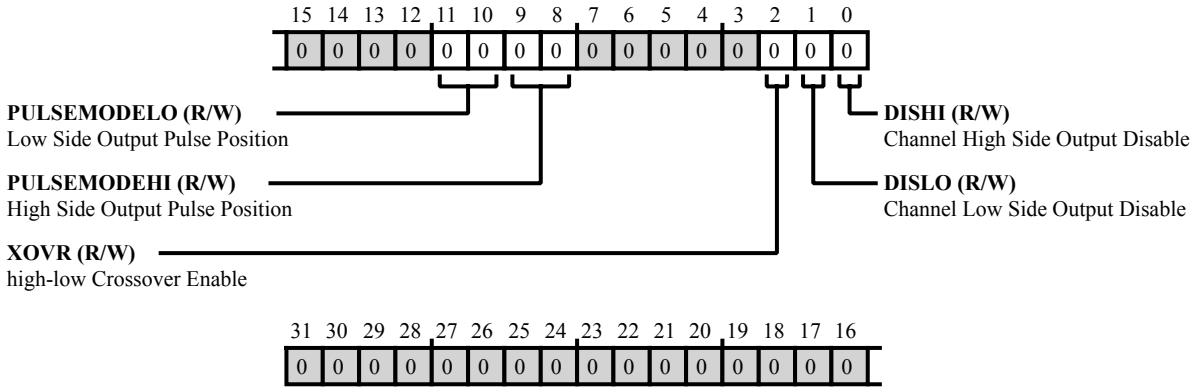

Figure 18-44: PWM_BCTL Register Diagram

Table 18-19: PWM_BCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/W)	PULSEMODELO	Low Side Output Pulse Position. The PWM_BCTL.PULSEMODELO bits select the pulse position for Channel B low side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_BL0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (PWM_BL0 and PWM_BL1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (PWM_BL0 and PWM_BL1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half

Table 18-19: PWM_BCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/W)	PULSEMODEHI	High Side Output Pulse Position. The PWM_BCTL.PULSEMODEHI bits select the pulse position for Channel B high side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_BH0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (PWM_BH0 and PWM_BH1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (PWM_BH0 and PWM_BH1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half
2 (R/W)	XOVR	high-low Crossover Enable. The PWM_BCTL.XOVR bit enables crossover between the channels high and low side outputs. When enabled, this bit directs the PWM to send the low-side output through the high-side output pin and the high-side output through the low side output pin.	
		0	Disable Crossover
		1	Enable Crossover
1 (R/W)	DISLO	Channel Low Side Output Disable. The PWM_BCTL.DISLO bit enables the channels low side output.	
		0	Enable Low Side Output
		1	Disable Low Side Output
0 (R/W)	DISHI	Channel High Side Output Disable. The PWM_BCTL.DISHI bit enables the channels high side output.	
		0	Enable High Side Output
		1	Disable High Side Output

Channel B-High Duty-0 Register

The PWM_BH0 and PWM_BH1 registers determine the width for the high side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the high side output pulses for the Channel B duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_BCTL.PULSEMODEHI bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_BH0 register to determine the assertion and de-assertion count for the high side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts Channel B high pulse output for count less than PWM_BH0 and de-asserts this output for count greater than PWM_BH1.

The value range for the PWM_BH0 and PWM_BH1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_BH0 and PWM_BH1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_DT (two's complement) to +PWM_TM0/2 + PWM_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_DT (two's complement) to -PWM_TM0/2 - PWM_DT.

Note that using values in the PWM_BH0 or PWM_BH1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

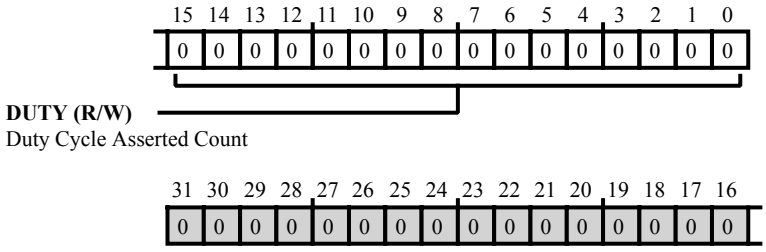

Figure 18-45: PWM_BH0 Register Diagram

Table 18-20: PWM_BH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count.

Channel B-High Heightened-Precision Duty-0 Register

The PWM_BH0_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_BH0 register, allows programs to specify fractional duty cycles. The PWM_BH0_HP register and the PWM_BH0 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_BH0_HP and the PWM_BH0 registers are also present in the single full duty register (if available).

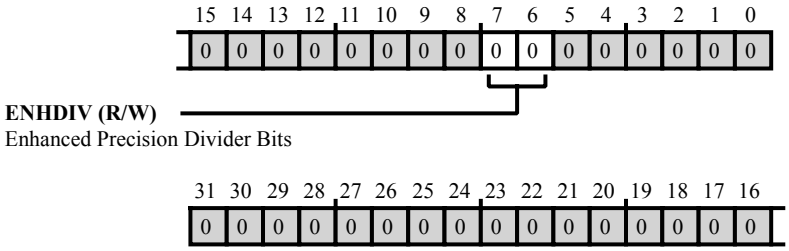

Figure 18-46: PWM_BH0_HP Register Diagram

Table 18-21: PWM_BH0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_BH0_HP.ENHDIV bits provide fractional duty cycles for Channel B high side output.

Channel B-High Duty-1 Register

The PWM_BH0 and PWM_BH1 registers determine the width for the high side output pulses. For more information, see the PWM_BH0 register description.

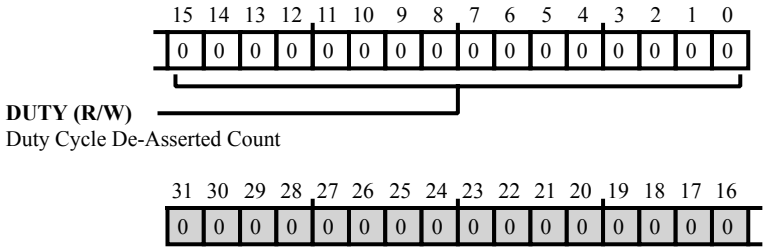

Figure 18-47: PWM_BH1 Register Diagram

Table 18-22: PWM_BH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count.

Channel B-High Heightened-Precision Duty-1 Register

The PWM_BH1_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_BH1 register, allows programs to specify fractional duty cycles. The PWM_BH1_HP register and the PWM_BH1 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_BH1_HP and the PWM_BH1 registers are also present in the single full duty register (if available).

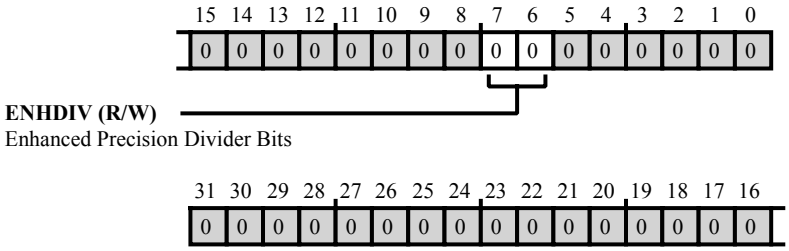

Figure 18-48: PWM_BH1_HP Register Diagram

Table 18-23: PWM_BH1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_BH1_HP.ENHDIV bits provide fractional duty cycles for Channel B high side output.

Channel B-High Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_BH_DUTY0 register contains the PWM_BH_DUTY0.DUTY bit field from the PWM_BH0 register and the PWM_BH_DUTY0.ENHDIV bit field from the PWM_BH0_HP register.

Note that the PWM_BH_DUTY0 register reads the PWM_BH0 and the PWM_BH0_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

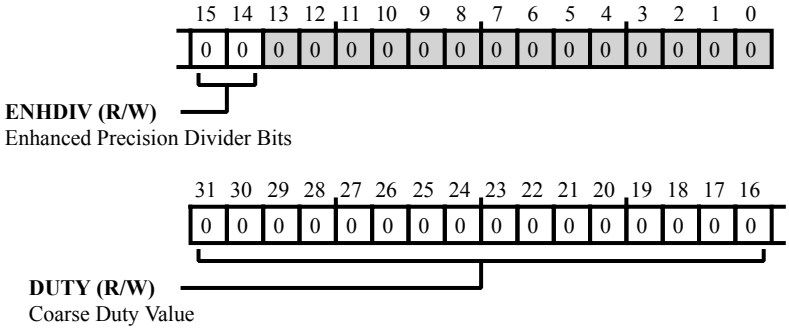

Figure 18-49: PWM_BH_DUTY0 Register Diagram

Table 18-24: PWM_BH_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_BH_DUTY0.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_BH_DUTY0.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_BH_DUTY0.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel B-High Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_BH_DUTY1 register contains the PWM_BH_DUTY1.DUTY bit field from the PWM_BH1 register and the PWM_BH_DUTY1.ENHDIV bit field from the PWM_BH1_HP register.

Note that the PWM_BH_DUTY1 register reads the PWM_BH1 and the PWM_BH1_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

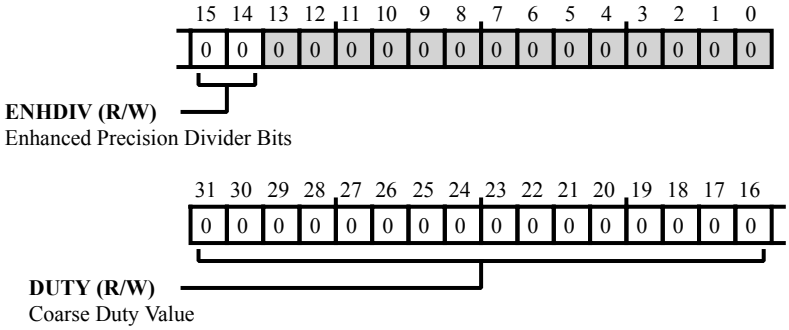

Figure 18-50: PWM_BH_DUTY1 Register Diagram

Table 18-25: PWM_BH_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_BH_DUTY1.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_BH_DUTY1.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_BH_DUTY1.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel B-Low Duty-0 Register

The PWM_BL0 and PWM_BL1 registers determine the width for the low side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the low side output pulses for the Channel B duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_BCTL.PULSEMODELO bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_BL0 register to determine the assertion and de-assertion count for the low side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts Channel B low pulse output for count less than PWM_BL0 and de-asserts this output for count greater than PWM_BL1.

The value range for the PWM_BL0 and PWM_BL1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_BL0 and PWM_BL1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_DT (two's complement) to +PWM_TM0/2 + PWM_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_DT (two's complement) to -PWM_TM0/2 - PWM_DT.

Note that using values in the PWM_BL0 or PWM_BL1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

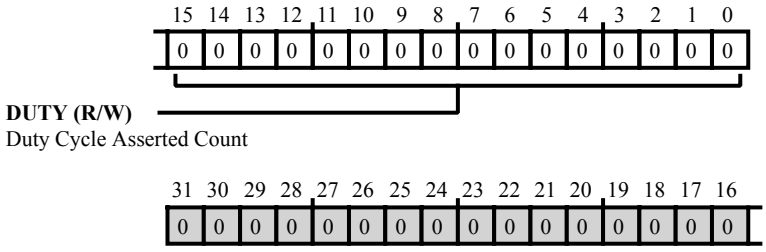

Figure 18-51: PWM_BL0 Register Diagram

Table 18-26: PWM_BL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_BL0.DUTY bits select the duty cycle asserted count for Channel B low side output.

Channel B-Low Heightened-Precision Duty-0 Register

The PWM_BL0_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_BL0 register, allows programs to specify fractional duty cycles. The PWM_BL0_HP register and the PWM_BL0 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_BL0_HP and the PWM_BL0 registers are also present in the single full duty register (PWM_BL_DUTY0).

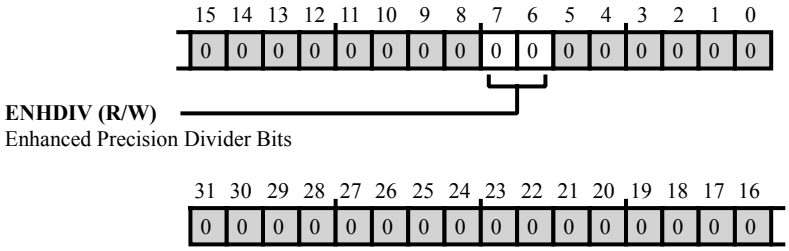

Figure 18-52: PWM_BL0_HP Register Diagram

Table 18-27: PWM_BL0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_BL0_HP.ENHDIV bits provide fractional duty cycles for Channel B low side output.

Channel B-Low Duty-1 Register

The PWM_BL0 and PWM_BL1 registers determine the width for the low side output pulses. For more information, see the PWM_BL0 register description.

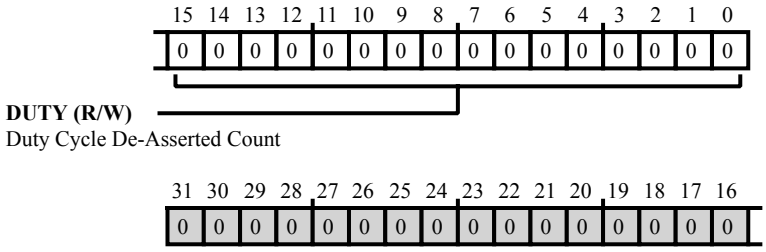

Figure 18-53: PWM_BL1 Register Diagram

Table 18-28: PWM_BL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The PWM_BL1.DUTY bits select the duty cycle de-asserted count for Channel B low side output.

Channel B-Low Heightened-Precision Duty-1 Register

The PWM_BL1_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_BL1 register, allows programs to specify fractional duty cycles. The PWM_BL1_HP register and the PWM_BL1 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_BL1_HP and the PWM_BL1 registers are also present in the single full duty register (if available).

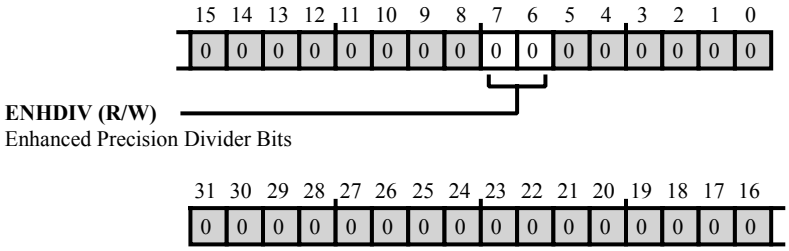

Figure 18-54: PWM_BL1_HP Register Diagram

Table 18-29: PWM_BL1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_BL1_HP.ENHDIV bits provide fractional duty cycles for Channel B low side output.

Channel B-Low Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_BL_DUTY0 register contains the PWM_BL_DUTY0.DUTY bit field from the PWM_BL0 register and the PWM_BL_DUTY0.ENHDIV bit field from the PWM_BL0_HP register.

Note that the PWM_BL_DUTY0 register reads the PWM_BL0 and the PWM_BL0_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

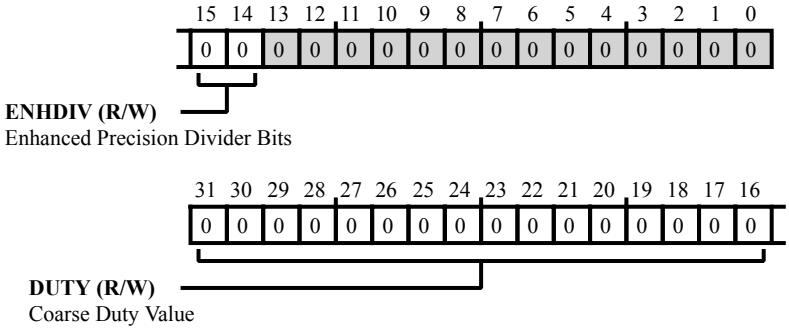

Figure 18-55: PWM_BL_DUTY0 Register Diagram

Table 18-30: PWM_BL_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_BL_DUTY0.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_BL_DUTY0.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_BL_DUTY0.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel B-Low Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_BL_DUTY1 register contains the PWM_BL_DUTY1.DUTY bit field from the PWM_BL1 register and the PWM_BL_DUTY1.ENHDIV bit field from the PWM_BL1_HP register.

Note that the PWM_BL_DUTY1 register reads the PWM_BL1 and the PWM_BL1_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

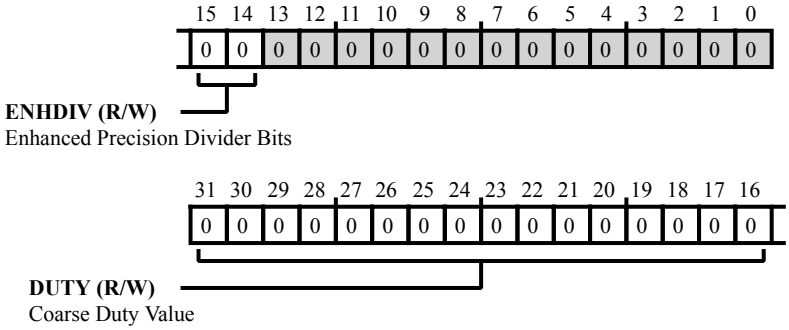

Figure 18-56: PWM_BL_DUTY1 Register Diagram

Table 18-31: PWM_BL_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_BL_DUTY1.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_BL_DUTY1.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_BL_DUTY1.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel C Control Register

The PWM_CCTL register selects the low and high side output pulse mode, enables low and high side output, and enables low/high side output crossover.

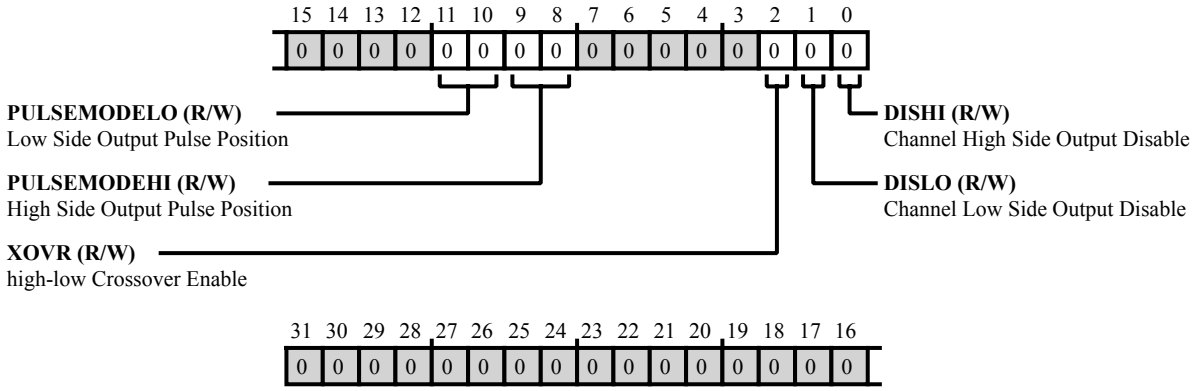

Figure 18-57: PWM_CCTL Register Diagram

Table 18-32: PWM_CCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/W)	PULSEMODELO	Low Side Output Pulse Position. The PWM_CCTL.PULSEMODELO bits select the pulse position for Channel C low side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_CL0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (PWM_CL0 and PWM_CL1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (PWM_CL0 and PWM_CL1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half

Table 18-32: PWM_CCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/W)	PULSEMODEHI	High Side Output Pulse Position. The PWM_CCTL.PULSEMODEHI bits select the pulse position for Channel C high side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_CH0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (PWM_CH0 and PWM_CH1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (PWM_CH0 and PWM_CH1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half
2 (R/W)	XOVR	high-low Crossover Enable. The PWM_CCTL.XOVR bit enables crossover between the channels high and low side outputs. When enabled, this bit directs the PWM to send the low-side output through the high-side output pin and the high-side output through the low side output pin.	
		0	Disable Crossover
		1	Enable Crossover
1 (R/W)	DISLO	Channel Low Side Output Disable. The PWM_CCTL.DISLO bit enables the channels low side output.	
		0	Enable Low Side Output
		1	Disable Low Side Output
0 (R/W)	DISHI	Channel High Side Output Disable. The PWM_CCTL.DISHI bit enables the channels high side output.	
		0	Enable High Side Output
		1	Disable High Side Output

Channel C-High Pulse Duty Register 0

The PWM_CH0 and PWM_CH1 registers determine the width for the high side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the high side output pulses for the Channel C duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_CCTL.PULSEMODEHI bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_CH0 register to determine the assertion and de-assertion count for the high side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts Channel C high pulse output for count less than PWM_CH0 and de-asserts this output for count greater than PWM_CH1.

The value range for the PWM_CH0 and PWM_CH1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_CH0 and PWM_CH1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_DT (two's complement) to +PWM_TM0/2 + PWM_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_DT (two's complement) to -PWM_TM0/2 - PWM_DT.

Note that using values in the PWM_CH0 or PWM_CH1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

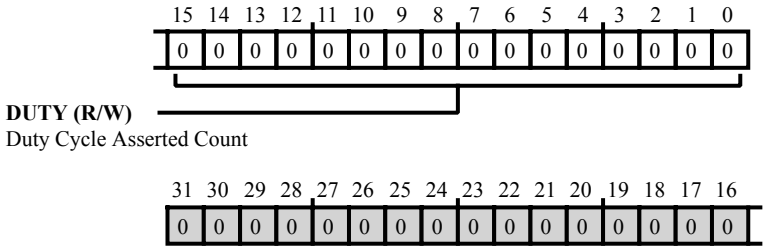

Figure 18-58: PWM_CH0 Register Diagram

Table 18-33: PWM_CH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_CH0.DUTY bits select the duty cycle asserted count for Channel C high side output.

Channel C-High Pulse Heightened-Precision Duty Register 0

The PWM_CH0_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_CH0 register, allows programs to specify fractional duty cycles. The PWM_CH0_HP register and the PWM_CH0 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_CH0_HP and the PWM_CH0 registers are also present in the single full duty register (if available).

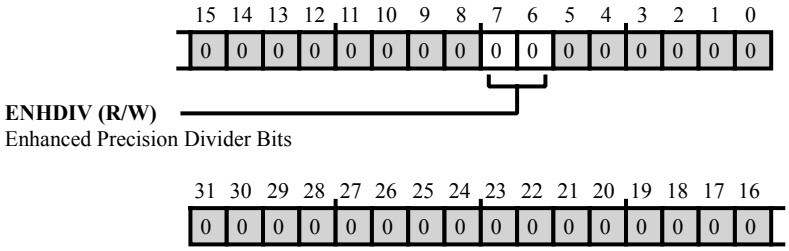

Figure 18-59: PWM_CH0_HP Register Diagram

Table 18-34: PWM_CH0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_CH0_HP.ENHDIV bits provide fractional duty cycles for Channel C high side output.

Channel C-High Pulse Duty Register 1

The PWM_CH0 and PWM_CH1 registers determine the width for the high side output pulses. For more information, see the PWM_CH0 register description.

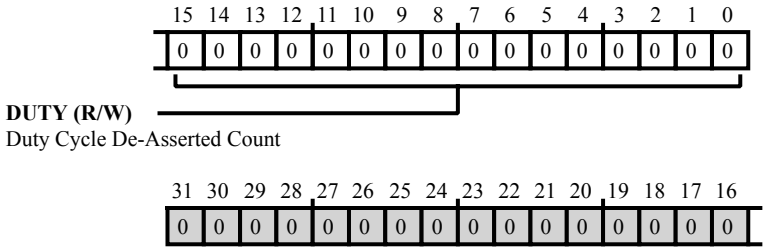

Figure 18-60: PWM_CH1 Register Diagram

Table 18-35: PWM_CH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The PWM_CH1.DUTY bits select the duty cycle de-asserted count for Channel C high side output.

Channel C-High Pulse Heightened-Precision Duty Register 1

The PWM_CH1_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_CH1 register, allows programs to specify fractional duty cycles. The PWM_CH1_HP register and the PWM_CH1 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_CH1_HP and the PWM_CH1 registers are also present in the single full duty register (if available).

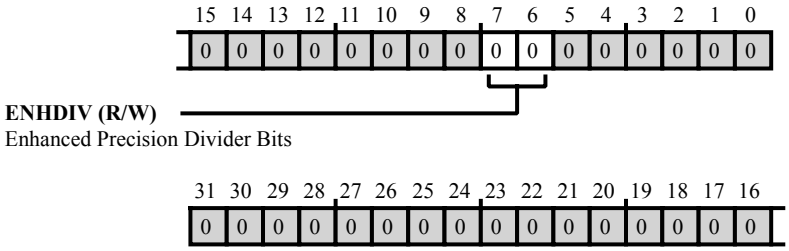

Figure 18-61: PWM_CH1_HP Register Diagram

Table 18-36: PWM_CH1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_CH1_HP.ENHDIV bits provide fractional duty cycles for Channel C high side output.

Channel Configuration Register

The PWM_CHANCFG register configures Channel A, B, C, and D reference timer selection, high and low side output features, and enables high frequency chopping operation. Do not change the value of any bits in the PWM register while the PWM is enabled (PWM_CTL.GLOBEN =1).

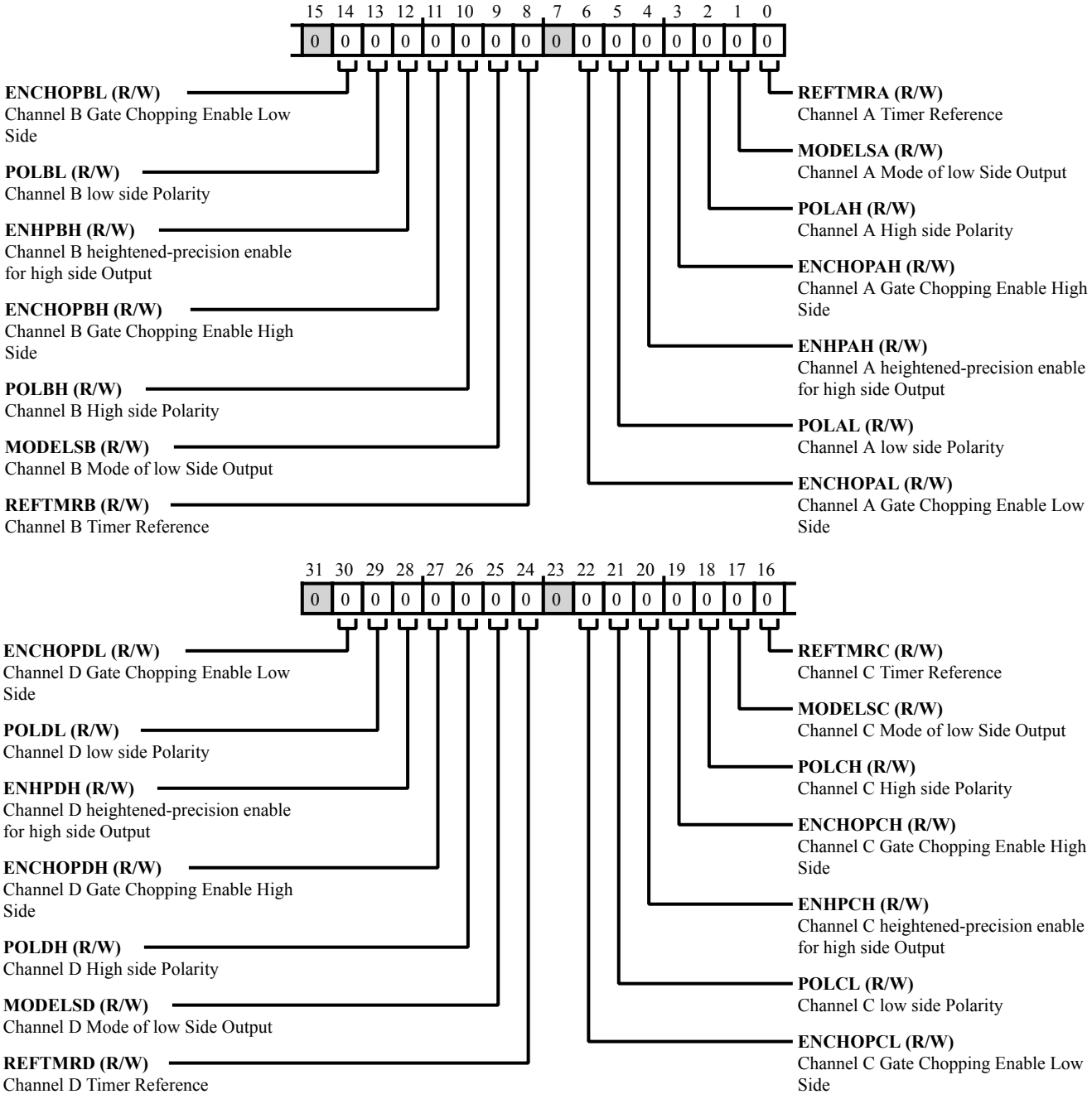

Figure 18-62: PWM_CHANCFG Register Diagram

Table 18-37: PWM_CHANCFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	ENCHOPDL	Channel D Gate Chopping Enable Low Side. The PWM_CHANCFG.ENCHOPDL bit enables mixing of the Channel D low side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel D Low Side
		1	Enable Chopping Channel D Low Side
29 (R/W)	POLDL	Channel D low side Polarity. The PWM_CHANCFG.POLDL bit selects the Channel D low side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
28 (R/W)	ENHPDH	Channel D heightened-precision enable for high side Output. The PWM_CHANCFG.ENHPDH bit enables heightened-precision Channel D high side output.	
		0	Disable HP Output Channel D High
		1	Enable HP Output Channel D High
27 (R/W)	ENCHOPDH	Channel D Gate Chopping Enable High Side. The PWM_CHANCFG.ENCHOPDH bit enables mixing of the Channel D high side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel D High Side
		1	Enable Chopping Channel D High Side
26 (R/W)	POLDH	Channel D High side Polarity. The PWM_CHANCFG.POLDH bit selects the Channel D high side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High

Table 18-37: PWM_CHANCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
25 (R/W)	MODELSD	Channel D Mode of low Side Output. The PWM_CHANCFG.MODELSD bit selects whether the low side output waveform is based on independent controls or whether the low side output depends on the high side output controls. When PWM_CHANCFG.MODELSD =0, the low side output is an inverted form of the high side output, which is generated using the PWM_BH0 and PWM_BH1 registers for pulse width, using the PWM_BCTL.PULSEMODEHI bits for pulse positioning, and PWM_CHANCFG.POLBH bits for polarity.	
		0	Invert of high output
		1	Independent control
24 (R/W)	REFTMRD	Channel D Timer Reference. The PWM_CHANCFG.REFTMRD bit selects whether the PWM uses PWMTMR1 or PWMTMR0 as the reference timer for Channel D operation.	
		0	PWMTMR0 is Channel D reference
		1	PWMTMR1 is Channel D reference
22 (R/W)	ENCHOPCL	Channel C Gate Chopping Enable Low Side. The PWM_CHANCFG.ENCHOPCL bit enables mixing of the Channel C low side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel C Low Side
		1	Enable Chopping Channel C Low Side
21 (R/W)	POLCL	Channel C low side Polarity. The PWM_CHANCFG.POLCL bit selects the Channel C low side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
20 (R/W)	ENHPCH	Channel C heightened-precision enable for high side Output. The PWM_CHANCFG.ENHPCH bit enables heightened-precision Channel C high side output.	
		0	Disable HP Output Channel C High
		1	Enable HP Output Channel C High

Table 18-37: PWM_CHANCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W)	ENCHOPCH	Channel C Gate Chopping Enable High Side. The PWM_CHANCFG.ENCHOPCH bit enables mixing of the Channel C high side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel C High Side
		1	Enable Chopping Channel C High Side
18 (R/W)	POLCH	Channel C High side Polarity. The PWM_CHANCFG.POLCH bit selects the Channel C high side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
17 (R/W)	MODELSC	Channel C Mode of low Side Output. The PWM_CHANCFG.MODELSC bit selects whether the low side output waveform is based on independent controls or whether the low side output depends on the high side output controls. When PWM_CHANCFG.MODELSC =0, the low side output is an inverted form of the high side output, which is generated using the PWM_BH0 and PWM_BH1 registers for pulse width, using the PWM_BCTL.PULSEMODEHI bits for pulse positioning, and PWM_CHANCFG.POLBH bits for polarity.	
		0	Invert of high output
		1	Independent control
16 (R/W)	REFTMRC	Channel C Timer Reference. The PWM_CHANCFG.REFTMRC bit selects whether the PWM uses PWMTMR1 or PWMTMR0 as the reference timer for Channel C operation.	
		0	PWMTMR0 is Channel C reference
		1	PWMTMR1 is Channel C reference
14 (R/W)	ENCHOPBL	Channel B Gate Chopping Enable Low Side. The PWM_CHANCFG.ENCHOPBL bit enables mixing of the Channel B low side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel B Low Side
		1	Enable Chopping Channel B Low Side

Table 18-37: PWM_CHANCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	POLBL	Channel B low side Polarity. The PWM_CHANCFG.POLBL bit selects the Channel B low side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
12 (R/W)	ENHPBH	Channel B heightened-precision enable for high side Output. The PWM_CHANCFG.ENHPBH bit enables heightened-precision Channel B high side output.	
		0	Disable HP Output Channel B High
		1	Enable HP Output Channel B High
11 (R/W)	ENCHOPBH	Channel B Gate Chopping Enable High Side. The PWM_CHANCFG.ENCHOPBH bit enables mixing of the Channel B high side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel B High Side
		1	Enable Chopping Channel B High Side
10 (R/W)	POLBH	Channel B High side Polarity. The PWM_CHANCFG.POLBH bit selects the Channel B high side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
9 (R/W)	MODELSB	Channel B Mode of low Side Output. The PWM_CHANCFG.MODELSB bit selects whether the low side output waveform is based on independent controls or whether the low side output depends on the high side output controls. When PWM_CHANCFG.MODELSB =0, the low side output is an inverted form of the high side output, which is generated using the PWM_BH0 and PWM_BH1 registers for pulse width, using the PWM_BCTL.PULSEMODEHI bits for pulse positioning, and PWM_CHANCFG.POLBH bits for polarity.	
		0	Invert of high output
		1	Independent control
8 (R/W)	REFTMRB	Channel B Timer Reference. The PWM_CHANCFG.REFTMRB bit selects whether the PWM uses PWMTMR1 or PWMTMR0 as the reference timer for Channel B operation.	
		0	PWMTMR0 is Channel B reference
		1	PWMTMR1 is Channel B reference

Table 18-37: PWM_CHANCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	ENCHOPAL	Channel A Gate Chopping Enable Low Side. The PWM_CHANCFG.ENCHOPAL bit enables mixing of the Channel A low side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel A Low Side
		1	Enable Chopping Channel A Low Side
5 (R/W)	POLAL	Channel A low side Polarity. The PWM_CHANCFG.POLAL bit selects the Channel A low side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
4 (R/W)	ENHPAH	Channel A heightened-precision enable for high side Output. The PWM_CHANCFG.ENHPAH bit enables heightened-precision Channel A high side output.	
		0	Disable HP Output Channel A High
		1	Enable HP Output Channel A High
3 (R/W)	ENCHOPAH	Channel A Gate Chopping Enable High Side. The PWM_CHANCFG.ENCHOPAH bit enables mixing of the Channel A high side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel A High Side
		1	Enable Chopping Channel A High Side
2 (R/W)	POLAH	Channel A High side Polarity. The PWM_CHANCFG.POLAH bit selects the Channel A high side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High

Table 18-37: PWM_CHANCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	MODELSA	Channel A Mode of low Side Output. The PWM_CHANCFG.MODELSA bit selects whether the low side output waveform is based on independent controls or whether the low side output depends on the high side output controls. When PWM_CHANCFG.MODELSA =0, the low side output is an inverted form of the high side output, which is generated using the PWM_AH0 and PWM_AH1 registers for pulse width, using the PWM_ACTL.PULSEMODEHI bits for pulse positioning, and PWM_CHANCFG.POLAH bits for polarity.	
		0	Invert of high output
		1	Independent control
0 (R/W)	REFTMRA	Channel A Timer Reference. The PWM_CHANCFG.REFTMRA bit selects whether the PWM uses PWMTMR1 or PWMTMR0 as the reference timer for Channel A operation.	
		0	PWMTMR0 is Channel A reference
		1	PWMTMR1 is Channel A reference

Chop Configuration Register

The PWM_CHOPCFG register holds a divisor value that controls the chopping frequency. The PWM permits a mixing of the output signals with a high-frequency chopping signal to aid with interfacing to pulse transformers. Also note that high-frequency chopping may be independently enabled for each channel's high-side and the low-side outputs using channel control bits. (For example, control chopping for Channel A with the PWM_CHANCFG.ENCHOPAH and PWM_CHANCFG.ENCHOPAH bits.)

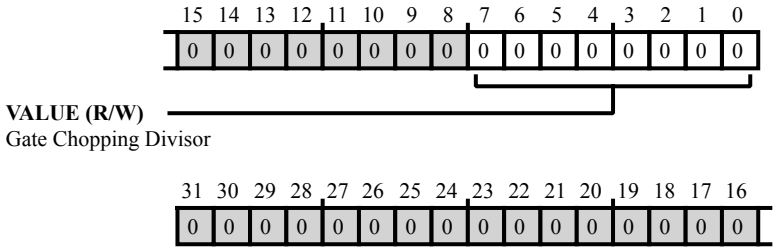

Figure 18-63: PWM_CHOPCFG Register Diagram

Table 18-38: PWM_CHOPCFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Gate Chopping Divisor. The PWM_CHOPCFG.VALUE bits provide the high frequency chopping divisor. When the divisor value is changed, the new period takes effect from the next edge of the chopping signal. The PWM_CHOPCFG.VALUE value may be calculated using either of the following formulas: CHOPDIV = $[(T_{CHOP}/T_{CK}) / 4] - 1$ CHOPDIV = $[(f_{CK} / f_{CHOP}) / 4] - 1$

Channel C-High Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_CH_DUTY0 register contains the PWM_CH_DUTY0.DUTY bit field from the PWM_CH0 register and the PWM_CH_DUTY0.ENHDIV bit field from the PWM_CH0_HP register.

Note that the PWM_CH_DUTY0 register reads the PWM_CH0 and the PWM_CH0_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

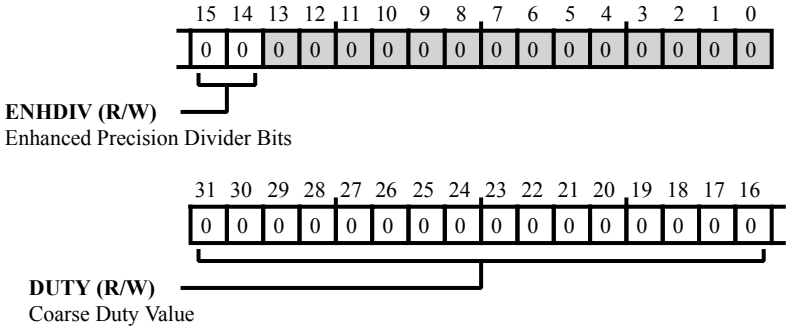

Figure 18-64: PWM_CH_DUTY0 Register Diagram

Table 18-39: PWM_CH_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_CH_DUTY0.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_CH_DUTY0.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_CH_DUTY0.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel C-High Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_CH_DUTY1 register contains the PWM_CH_DUTY1.DUTY bit field from the PWM_CH1 register and the PWM_CH_DUTY1.ENHDIV bit field from the PWM_CH1_HP register.

Note that the PWM_CH_DUTY1 register reads the PWM_CH1 and the PWM_CH1_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

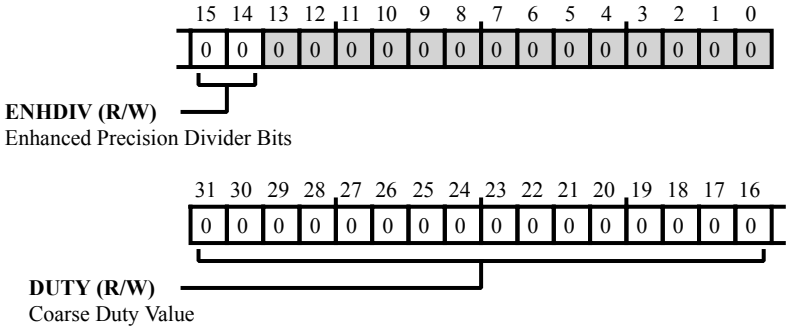

Figure 18-65: PWM_CH_DUTY1 Register Diagram

Table 18-40: PWM_CH_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_CH_DUTY1.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_CH_DUTY1.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_CH_DUTY1.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel C-Low Pulse Duty Register 0

The PWM_CL0 and PWM_CL1 registers determine the width for the low side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the low side output pulses for the Channel C duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_CCTL.PULSEMODELO bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_CL0 register to determine the assertion and de-assertion count for the low side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts Channel C low pulse output for count less than PWM_CL0 and de-asserts this output for count greater than PWM_CL1.

The value range for the PWM_CL0 and PWM_CL1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_CL0 and PWM_CL1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_DT (two's complement) to +PWM_TM0/2 + PWM_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_DT (two's complement) to -PWM_TM0/2 - PWM_DT.

Note that using values in the PWM_CL0 or PWM_CL1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

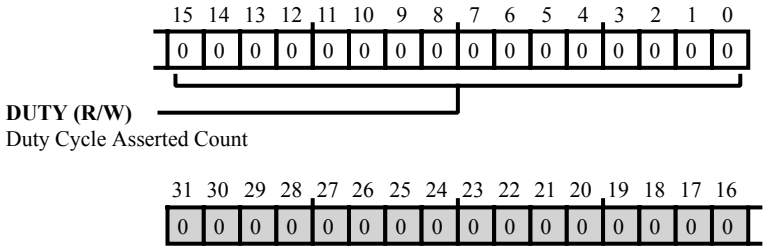

Figure 18-66: PWM_CL0 Register Diagram

Table 18-41: PWM_CL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_CL0.DUTY bits select the duty cycle asserted count for Channel C low side output.

Channel C-Low Pulse Duty Register 1

The PWM_CL0_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_CL0 register, allows programs to specify fractional duty cycles. The PWM_CL0_HP register and the PWM_CL0 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_CL0_HP and the PWM_CL0 registers are also present in the single full duty register (if available).

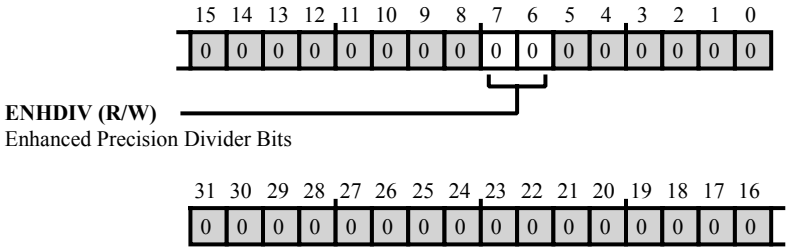

Figure 18-67: PWM_CL0_HP Register Diagram

Table 18-42: PWM_CL0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_CL0_HP.ENHDIV bits provide fractional duty cycles for Channel C low side output.

Channel C-Low Duty-1 Register

The PWM_CL0 and PWM_CL1 registers determine the width for the low side output pulses. For more information, see the PWM_CL0 register description.

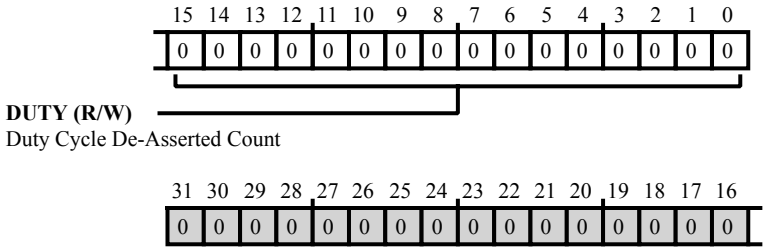

Figure 18-68: PWM_CL1 Register Diagram

Table 18-43: PWM_CL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count.

Channel C-Low Heightened-Precision Duty-1 Register

The PWM_CL1_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_CL1 register, allows programs to specify fractional duty cycles. The PWM_CL1_HP register and the PWM_CL1 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_CL1_HP and the PWM_CL1 registers are also present in the single full duty register (if available).

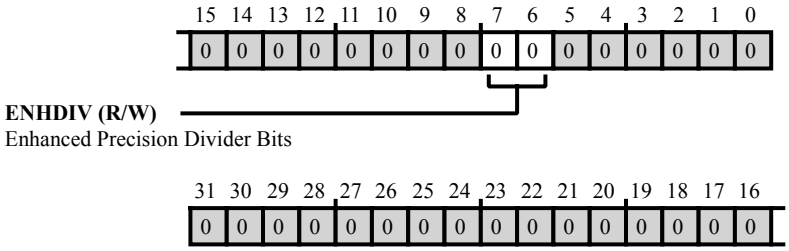

Figure 18-69: PWM_CL1_HP Register Diagram

Table 18-44: PWM_CL1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_CL1_HP.ENHDIV bits provide fractional duty cycles for Channel C low side output.

Channel C-Low Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_CL_DUTY0 register contains the PWM_CL_DUTY0.DUTY bit field from the PWM_CL0 register and the PWM_CL_DUTY0.ENHDIV bit field from the PWM_CL0_HP register.

Note that the PWM_CL_DUTY0 register reads the PWM_CL0 and the PWM_CL0_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

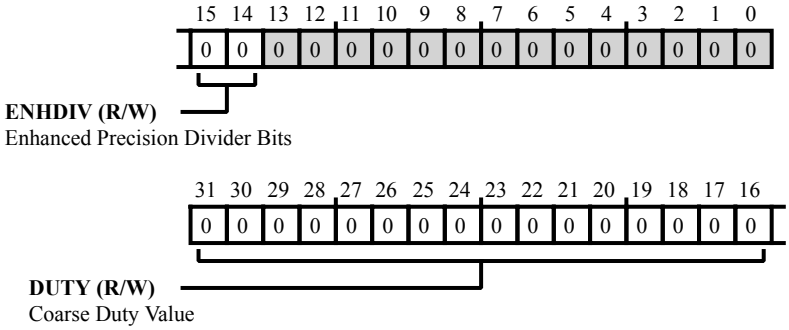

Figure 18-70: PWM_CL_DUTY0 Register Diagram

Table 18-45: PWM_CL_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_CL_DUTY0.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_CL_DUTY0.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_CL_DUTY0.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel C-Low Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_CL_DUTY1 register contains the PWM_CL_DUTY1.DUTY bit field from the PWM_CL1 register and the PWM_CL_DUTY1.ENHDIV bit field from the PWM_CL1_HP register.

Note that the PWM_CL_DUTY1 register reads the PWM_CL1 and the PWM_CL1_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

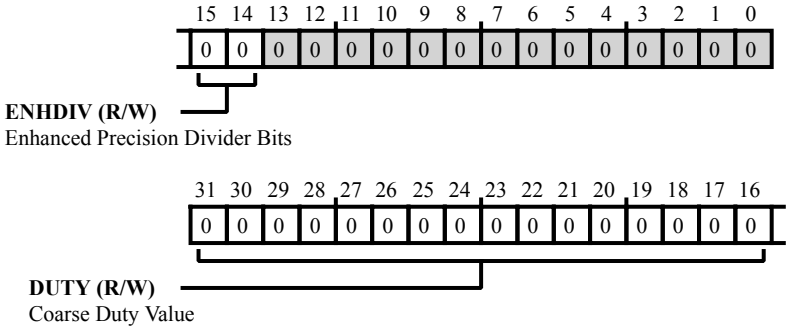

Figure 18-71: PWM_CL_DUTY1 Register Diagram

Table 18-46: PWM_CL_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_CL_DUTY1.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_CL_DUTY1.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_CL_DUTY1.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Control Register

The PWM_CTL register enables the PWM, enables delay counters for the channels, and configures sync features. This register also provides support for tripping a PWM fault condition through software.

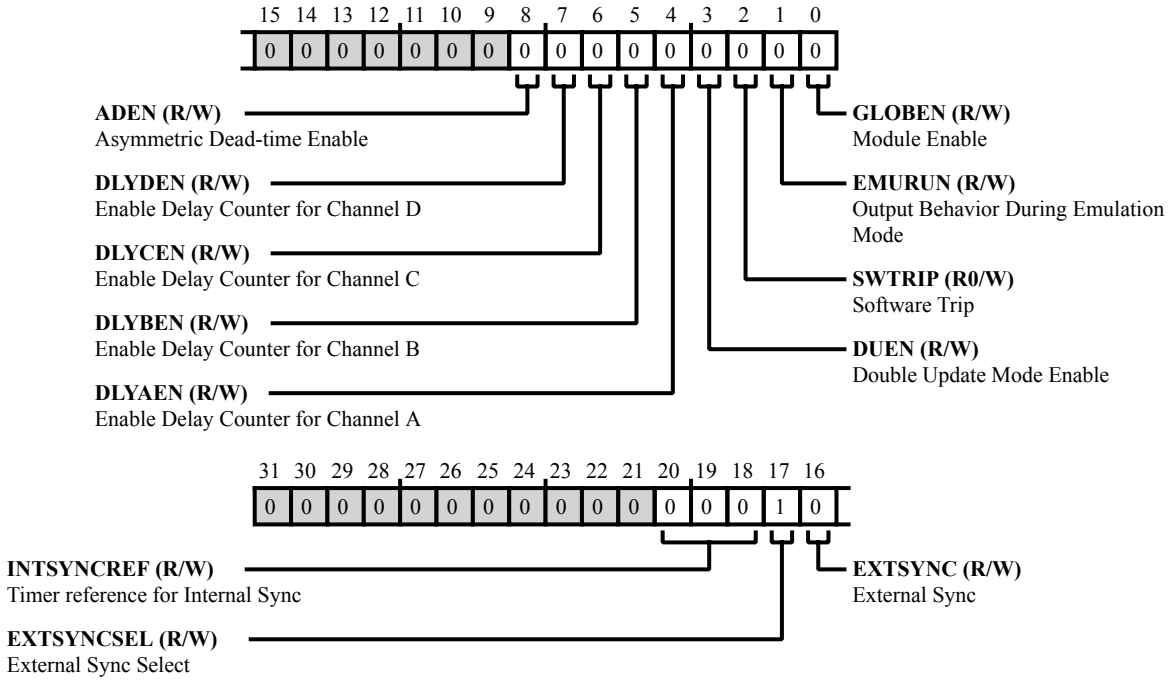

Figure 18-72: PWM_CTL Register Diagram

Table 18-47: PWM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20:18 (R/W)	INTSYNCREF	Timer reference for Internal Sync. The PWM_CTL.INTSYNCREF bits select the timer reference for the internal sync. Note that all other combinations reserved.	
		0	PWMTMR0 provides sync reference
		1	PWMTMR1 provides sync reference
		2	PWMTMR2 provides sync reference
		3	PWMTMR3 provides sync reference
		4	PWMTMR4 provides sync reference

Table 18-47: PWM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	EXTSYNCSEL	External Sync Select. The PWM_CTL.EXTSYNCSEL bit selects whether the external sync signal is synchronous or asynchronous. Note that latency in PWM sync response differs between asynchronous and synchronous external sync modes. For more information, see the PWM functional description.	
		0	Asynchronous External Sync
		1	Synchronous External Sync
16 (R/W)	EXTSYNC	External Sync. The PWM_CTL.EXTSYNC bit selects whether the PWM uses an external or internal sync signal for the main timer (PWMTMR0). Do not change the value of the PWM_CTL.EXTSYNC bit while the PWM is enabled (PWM_CTL.GLOBEN =1).	
		0	Internal sync used
		1	External sync used
8 (R/W)	ADEN	Asymmetric Dead-time Enable. When symmetric dead-time is enabled, in the dependent mode, both the high-side and low-side outputs are shrunk by DT cycles on both the assertion and de-assertion edges. When symmetric dead-time is enabled, the falling edges of both high and low-side outputs occur at the programmed duty value, but their rise-times are delayed by 2 times DT.	
		0	Dead-time is symmetric
		1	Dead-time is asymmetric
7 (R/W)	DLYDEN	Enable Delay Counter for Channel D. The PWM_CTL.DLYDEN bit enables the Channel D delay counter, supporting phase-offset control. Do not change the value of the PWM_CTL.DLYDEN bit while the PWM is enabled (PWM_CTL.GLOBEN =1).	
		0	Disable
		1	Enable
6 (R/W)	DLYCEN	Enable Delay Counter for Channel C. The PWM_CTL.DLYCEN bit enables the Channel C delay counter, supporting phase-offset control. Do not change the value of the PWM_CTL.DLYCEN bit while the PWM is enabled (PWM_CTL.GLOBEN =1).	
		0	Disable
		1	Enable

Table 18-47: PWM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	DLYBEN	Enable Delay Counter for Channel B. The PWM_CTL.DLYBEN bit enables the Channel B delay counter, supporting phase-offset control. Do not change the value of the PWM_CTL.DLYBEN bit while the PWM is enabled (PWM_CTL.GLOBEN =1).	
		0	Disable
		1	Enable
4 (R/W)	DLYAEN	Enable Delay Counter for Channel A. The PWM_CTL.DLYAEN bit enables the Channel A delay counter, supporting phase-offset control. Do not change the value of the PWM_CTL.DLYAEN bit while the PWM is enabled (PWM_CTL.GLOBEN =1).	
		0	Disable
		1	Enable
3 (R/W)	DUEN	Double Update Mode Enable. In Single Update Mode, double buffering of all registers happens at the period boundary of the PWM timer. In Double Update Mode, double buffering of all registers (except delay counter registers) happens at the middle of the period as well as the beginning of the period.	
		0	Single Update Mode
		1	Double Update Mode
2 (R0/W)	SWTRIP	Software Trip. The PWM_CTL.SWTRIP bit permits tripping a fault condition through software, shutting down PWM output. This bit always read as 0. If the PWM_CTL.SWTRIP bit and PWM_CTL.GLOBEN bit are set in the same write, the write does not trip the fault condition.	
		0	Don't force fault
		1	Force a Fault Trip Condition
1 (R/W)	EMURUN	Output Behavior During Emulation Mode. The PWM_CTL.EMURUN bit selects PWM output behavior during emulation mode.	
		0	Disable Outputs
		1	Enable Outputs

Table 18-47: PWM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	GLOBEN	Module Enable. The PWM_CTL.GLOBEN bit enables the PWM, enabling all timers and outputs. While this bit is enabled, processor code should not change the value of the PWM_CTL.DLYAEN bit, PWM_CTL.DLYBEN bit, PWM_CTL.DLYCEN bit, PWM_CTL.DLYDEN bit, PWM_CTL.EXTSYNCSEL bit, or any bits in the PWM_CHANCFG register. Note that there is a latency between PWM disable and the cessation of output waveforms. There is also a latency between PWM enable and start of output waveforms. For the latency description, see the PWM functional description.	
		0	Disable
		1	Enable

Channel D Control Register

The PWM_DCTL register selects the low and high side output pulse mode, enables low and high side output, and enables low/high side output crossover.

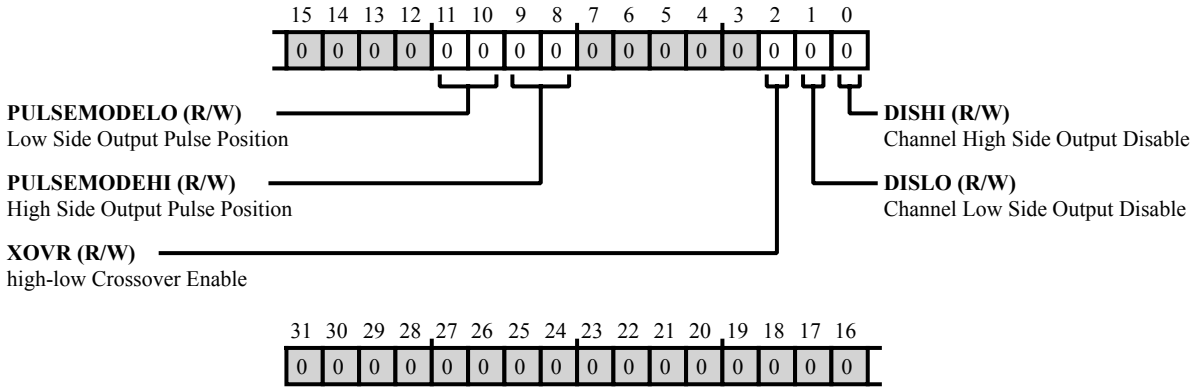

Figure 18-73: PWM_DCTL Register Diagram

Table 18-48: PWM_DCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/W)	PULSEMODELO	Low Side Output Pulse Position. The PWM_DCTL.PULSEMODELO bits select the pulse position for Channel D low side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_DL0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (PWM_DL0 and PWM_DL1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (PWM_DL0 and PWM_DL1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half

Table 18-48: PWM_DCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/W)	PULSEMODEHI	High Side Output Pulse Position. The PWM_DCTL.PULSEMODEHI bits select the pulse position for Channel D high side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_DH0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (PWM_DH0 and PWM_DH1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (PWM_DH0 and PWM_DH1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half
2 (R/W)	XOVR	high-low Crossover Enable. The PWM_DCTL.XOVR bit enables crossover between the channels high and low side outputs. When enabled, this bit directs the PWM to send the low-side output through the high-side output pin and the high-side output through the low side output pin.	
		0	Disable Crossover
		1	Enable Crossover
1 (R/W)	DISLO	Channel Low Side Output Disable. The PWM_DCTL.DISLO bit enables the channels low side output.	
		0	Enable Low Side Output
		1	Disable Low Side Output
0 (R/W)	DISHI	Channel High Side Output Disable. The PWM_DCTL.DISHI bit enables the channels high side output.	
		0	Enable High Side Output
		1	Disable High Side Output

Channel D-High Duty-0 Register

The PWM_DH0 and PWM_DH1 registers determine the width for the high side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the high side output pulses for the Channel D duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_DCTL.PULSEMODEHI bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_DH0 register to determine the assertion and de-assertion count for the high side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts Channel D high pulse output for count less than PWM_DH0 and de-asserts this output for count greater than PWM_DH1.

The value range for the PWM_DH0 and PWM_DH1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_DH0 and PWM_DH1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_DT (two's complement) to +PWM_TM0/2 + PWM_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_DT (two's complement) to -PWM_TM0/2 - PWM_DT.

Note that using values in the PWM_DH0 or PWM_DH1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

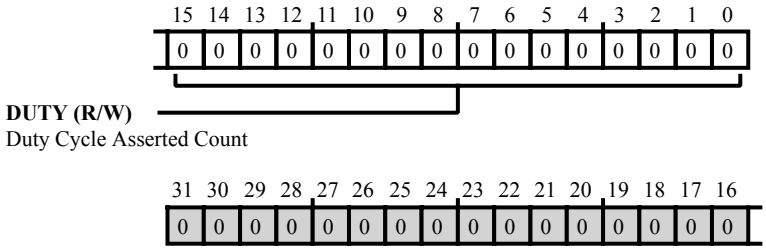

Figure 18-74: PWM_DH0 Register Diagram

Table 18-49: PWM_DH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_DH0.DUTY bits select the duty cycle asserted count for Channel D high side output.

Channel D-High Pulse Heightened-Precision Duty Register 0

The PWM_DH0_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_DH0 register, allows programs to specify fractional duty cycles. The PWM_DH0_HP register and the PWM_DH0 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_DH0_HP and the PWM_DH0 registers are also present in the single full duty register (if available).

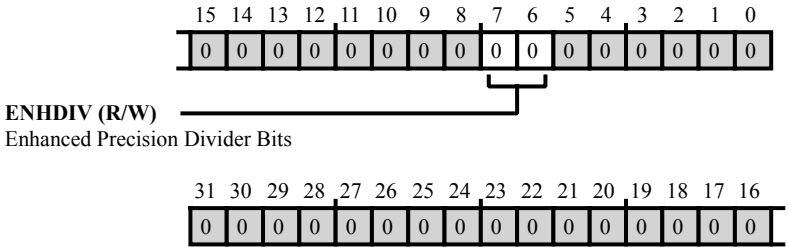

Figure 18-75: PWM_DH0_HP Register Diagram

Table 18-50: PWM_DH0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_DH0_HP.ENHDIV bits provide fractional duty cycles for Channel D high side output.

Channel D-High Pulse Duty Register 1

The PWM_DH0 and PWM_DH1 registers determine the width for the high side output pulses. For more information, see the PWM_DH0 register description.

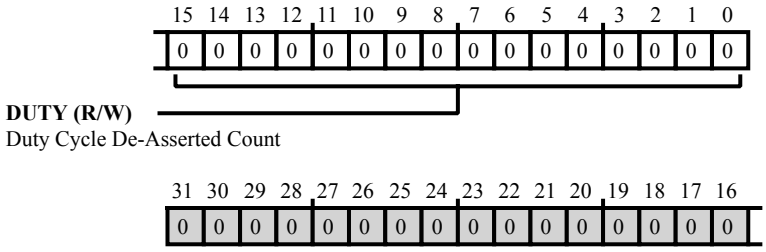

Figure 18-76: PWM_DH1 Register Diagram

Table 18-51: PWM_DH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The PWM_DH1.DUTY bits select the duty cycle de-asserted count for Channel D high side output.

Channel D High Pulse Heightened-Precision Duty Register 1

The PWM_DH1_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_DH1 register, allows programs to specify fractional duty cycles. The PWM_DH1_HP register and the PWM_DH1 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_DH1_HP and the PWM_DH1 registers are also present in the single full duty register (if available).

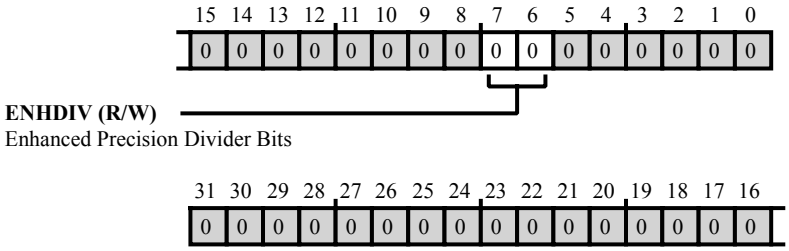

Figure 18-77: PWM_DH1_HP Register Diagram

Table 18-52: PWM_DH1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_DH1_HP.ENHDIV bits provide fractional duty cycles for Channel D high side output.

Channel D-High Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_DH_DUTY0 register contains the PWM_DH_DUTY0.DUTY bit field from the PWM_DH0 register and the PWM_DH_DUTY0.ENHDIV bit field from the PWM_DH0_HP register.

Note that the PWM_DH_DUTY0 register reads the PWM_DH0 and the PWM_DH0_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

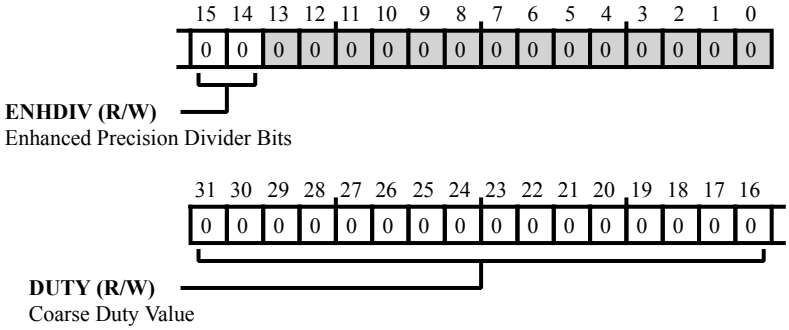

Figure 18-78: PWM_DH_DUTY0 Register Diagram

Table 18-53: PWM_DH_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_DH_DUTY0.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_DH_DUTY0.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_DH_DUTY0.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel D-High Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_DH_DUTY1 register contains the PWM_DH_DUTY1.DUTY bit field from the PWM_DH1 register and the PWM_DH_DUTY1.ENHDIV bit field from the PWM_DH1_HP register.

Note that the PWM_DH_DUTY1 register reads the PWM_DH1 and the PWM_DH1_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

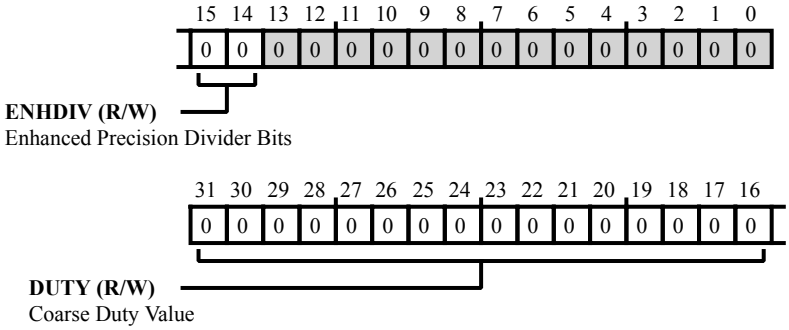

Figure 18-79: PWM_DH_DUTY1 Register Diagram

Table 18-54: PWM_DH_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_DH_DUTY1.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_DH_DUTY1.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_DH_DUTY1.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel D-Low Pulse Duty Register 0

The PWM_DL0 and PWM_DL1 registers determine the width for the low side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the low side output pulses for the Channel D duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_DCTL.PULSEMODELO bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_DL0 register to determine the assertion and de-assertion count for the low side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts Channel D low pulse output for count less than PWM_DL0 and de-asserts this output for count greater than PWM_DL1.

The value range for the PWM_DL0 and PWM_DL1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_DL0 and PWM_DL1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_DT (two's complement) to +PWM_TM0/2 + PWM_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_DT (two's complement) to -PWM_TM0/2 - PWM_DT.

Note that using values in the PWM_DL0 or PWM_DL1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

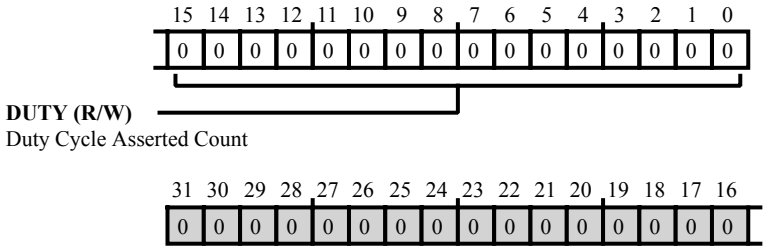

Figure 18-80: PWM_DL0 Register Diagram

Table 18-55: PWM_DL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_DL0.DUTY bits select the duty cycle asserted count for Channel D low side output.

Channel D-Low Heightened-Precision Duty-0 Register

The PWM_DL0_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_DL0 register, allows programs to specify fractional duty cycles. The PWM_DL0_HP register and the PWM_DL0 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_DL0_HP and the PWM_DL0 registers are also present in the single full duty register (if available).

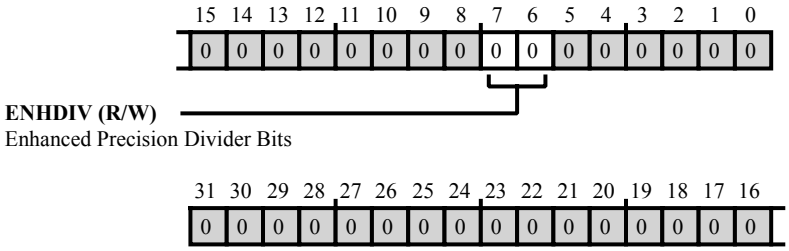

Figure 18-81: PWM_DL0_HP Register Diagram

Table 18-56: PWM_DL0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_DL0_HP.ENHDIV bits provide fractional duty cycles for Channel D low side output.

Channel D-Low Pulse Duty Register 1

The PWM_DL0 and PWM_DL1 registers determine the width for the low side output pulses. For more information, see the PWM_DL0 register description.

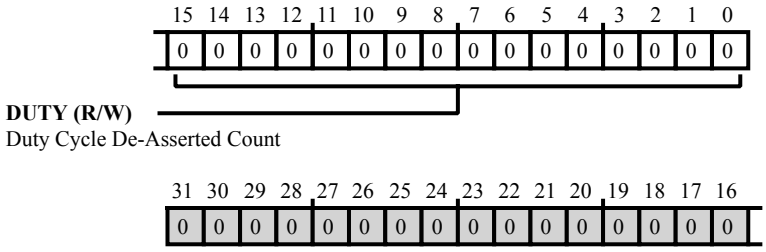

Figure 18-82: PWM_DL1 Register Diagram

Table 18-57: PWM_DL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The PWM_DL1.DUTY bits select the duty cycle de-asserted count for Channel D low side output.

Channel D-Low Heightened-Precision Duty-1 Register

The PWM_DL1_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_DL1 register, allows programs to specify fractional duty cycles. The PWM_DL1_HP register and the PWM_DL1 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_DL1_HP and the PWM_DL1 registers are also present in the single full duty register (if available).

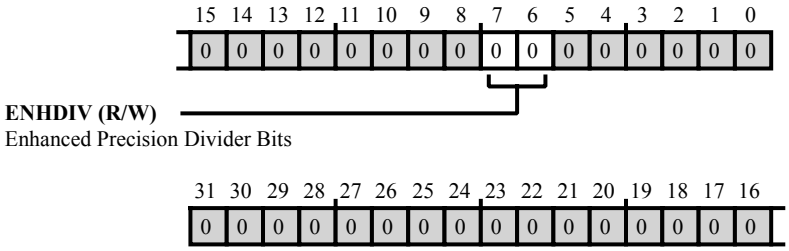

Figure 18-83: PWM_DL1_HP Register Diagram

Table 18-58: PWM_DL1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_DL1_HP.ENHDIV bits provide fractional duty cycles for Channel D low side output.

Channel A Delay Register

The PWM_DLYA register controls a delay for the Channel A timer (only PWMTMR1, PWMTMR2, PWMTMR3 or PWMTMR4) with reference to the main timer (PWMTMR0). To use apply this delay, the delay must be enabled (PWM_CTL.DLYAEN =1). For more information about applying the delay, see the PWM Functional Description section. Note that the PWM_DLYA delay value must be less that less that twice the period value of the timer being used for the channel (for example, if PWMTMR1 is used, PWM_DLYA must be less than 2xPWMTMR1). Also, note that the period of the main timer must be an integer multiple of the timer being used for the channel (for example, if PWMTMR1 is used, PWMTMR0 = NxPWMTMR1, where N is an integer).

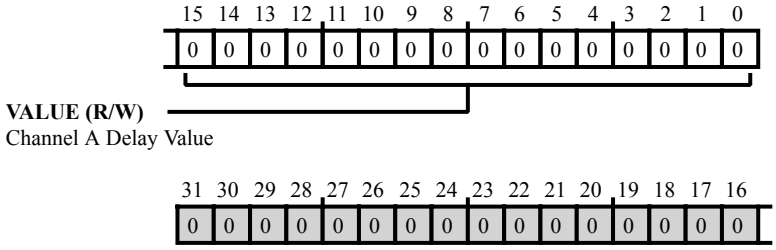

Figure 18-84: PWM_DLYA Register Diagram

Table 18-59: PWM_DLYA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Channel A Delay Value. The PWM_DLYA.VALUE bits select the phase delay between the main timer (PWMTMR0) and the timer used for Channel A.

Channel B Delay Register

The PWM_DLYB register controls a delay for the Channel B timer (only PWMTMR1, PWMTMR2, PWMTMR3 or PWMTMR4) with reference to the main timer (PWMTMR0). To use apply this delay, the delay must be enabled (PWM_CTL.DLYBEN =1). For more information about applying the delay, see the PWM Functional Description section. Note that the PWM_DLYB delay value must be less that less that twice the period value of the timer being used for the channel (for example, if PWMTMR1 is used, PWM_DLYB must be less than 2xPWMTMR1). Also, note that the period of the main timer must be an integer multiple of the timer being used for the channel (for example, if PWMTMR1 is used, PWMTMR0 = NxPWMTMR1, where N is an integer).

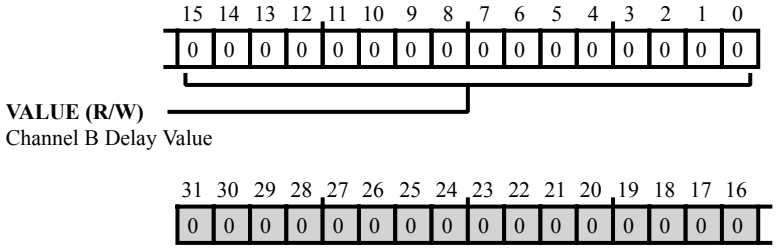

Figure 18-85: PWM_DLYB Register Diagram

Table 18-60: PWM_DLYB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Channel B Delay Value. The PWM_DLYB.VALUE bits select the phase delay between the main timer (PWMTMR0) and the timer used for Channel B.

Channel C Delay Register

The PWM_DLYC register controls a delay for the Channel C timer (only PWMTMR1, PWMTMR2, PWMTMR3 or PWMTMR4) with reference to the main timer (PWMTMR0). To use apply this delay, the delay must be enabled (PWM_CTL.DLYCEN =1}. For more information about applying the delay, see the PWM Functional Description section. Note that the PWM_DLYC delay value must be less that less that twice the period value of the timer being used for the channel (for example, if PWMTMR1 is used, PWM_DLYC must be less than 2xPWMTMR1). Also, note that the period of the main timer must be an integer multiple of the timer being used for the channel (for example, if PWMTMR1 is used, PWMTMR0 = NxPWMTMR1, where N is an integer).

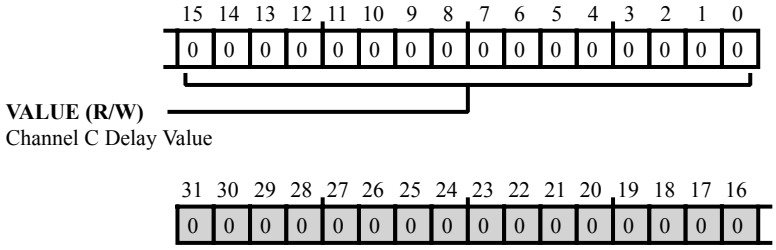

Figure 18-86: PWM_DLYC Register Diagram

Table 18-61: PWM_DLYC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Channel C Delay Value. The PWM_DLYC.VALUE bits select the phase delay between the main timer (PWMTMR0) and the timer used for Channel C.

Channel D Delay Register

The PWM_DLYD register controls a delay for the Channel D timer (only PWMTMR1, PWMTMR2, PWMTMR3 or PWMTMR4) with reference to the main timer (PWMTMR0). To use apply this delay, the delay must be enabled (PWM_CTL.DLYDEN =1). For more information about applying the delay, see the PWM Functional Description section. Note that the PWM_DLYD delay value must be less that less that twice the period value of the timer being used for the channel (for example, if PWMTMR1 is used, PWM_DLYD must be less than 2xPWMTMR1). Also, note that the period of the main timer must be an integer multiple of the timer being used for the channel (for example, if PWMTMR1 is used, PWMTMR0 = NxPWMTMR1, where N is an integer).

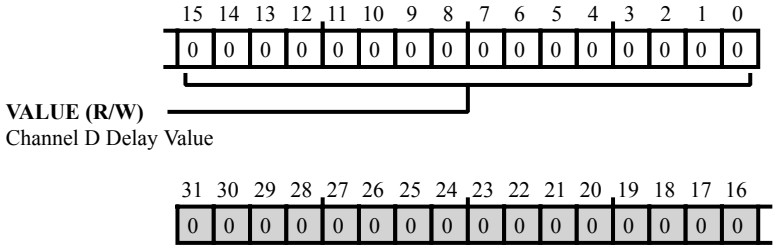

Figure 18-87: PWM_DLYD Register Diagram

Table 18-62: PWM_DLYD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Channel D Delay Value. The PWM_DLYD.VALUE bits select the phase delay between the main timer (PWMTMR0) and the timer used for Channel D.

Channel D-Low Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_DL_DUTY0 register contains the PWM_DL_DUTY0.DUTY bit field from the PWM_DL0 register and the PWM_DL_DUTY0.ENHDIV bit field from the PWM_DL0_HP register.

Note that the PWM_DL_DUTY0 register reads the PWM_DL0 and the PWM_DL0_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

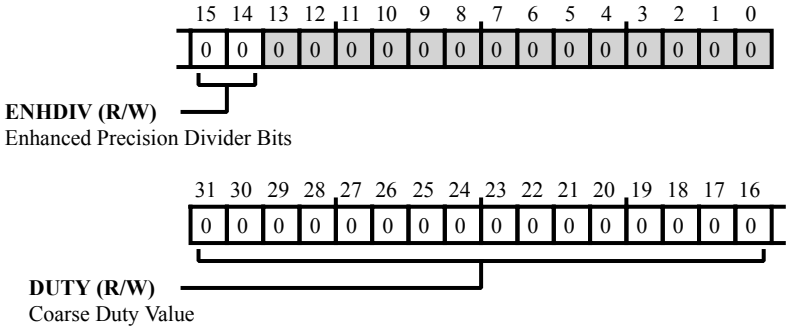

Figure 18-88: PWM_DL_DUTY0 Register Diagram

Table 18-63: PWM_DL_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_DL_DUTY0.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_DL_DUTY0.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_DL_DUTY0.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel D-Low Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_DL_DUTY1 register contains the PWM_DL_DUTY1.DUTY bit field from the PWM_DL1 register and the PWM_DL_DUTY1.ENHDIV bit field from the PWM_DL1_HP register.

Note that the PWM_DL_DUTY1 register reads the PWM_DL1 and the PWM_DL1_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

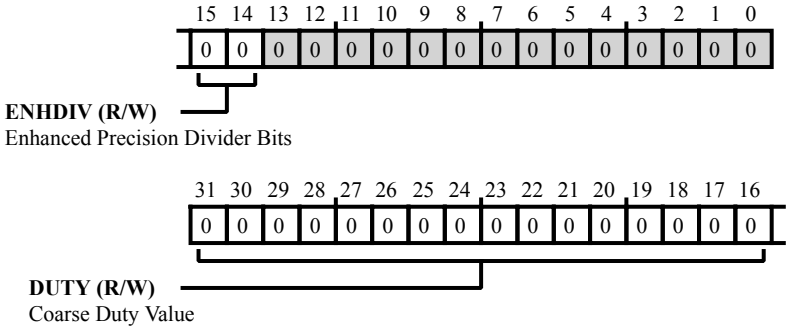

Figure 18-89: PWM_DL_DUTY1 Register Diagram

Table 18-64: PWM_DL_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_DL_DUTY1.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_DL_DUTY1.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_DL_DUTY1.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Dead Time Register

The PWM_DT register controls the dead time, which the PWM inserts into the pairs of output signals. Note that each channel has its own version of a double buffered dead time register, the double buffering of which depends on the period boundary of the PWMTMRx that the channel could be currently using. The dead time, T_d, is related to the value in the PWM_DT register by:

T_d = PWM_DT x 2 x t_{CK}

Note that the PWM holds the buffered PWM_DT value for a channel at 0 if the channel's low side mode is independent (for example, PWM_CHANCFG.MODELSA =1). Also, note that the PWM_DT value must be less than half the respective timer period (for example, PWM_TM0/2). For more information about applying dead time to PWM output pairs, see the PWM Functional Description section.

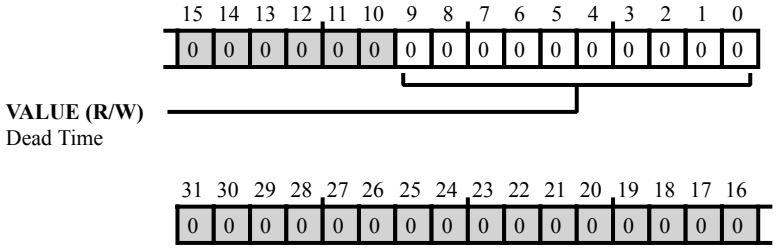

Figure 18-90: PWM_DT Register Diagram

Table 18-65: PWM_DT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	VALUE	Dead Time. The PWM_DT.VALUE bits select the dead time that the PWM adds to the timing of the output pairs.

Interrupt Latch Register

The PWM_ILAT register latches the occurrence of unmasked (enabled) PWM interrupts. These interrupts are unmasked or masked with the PWM_IMSK register.

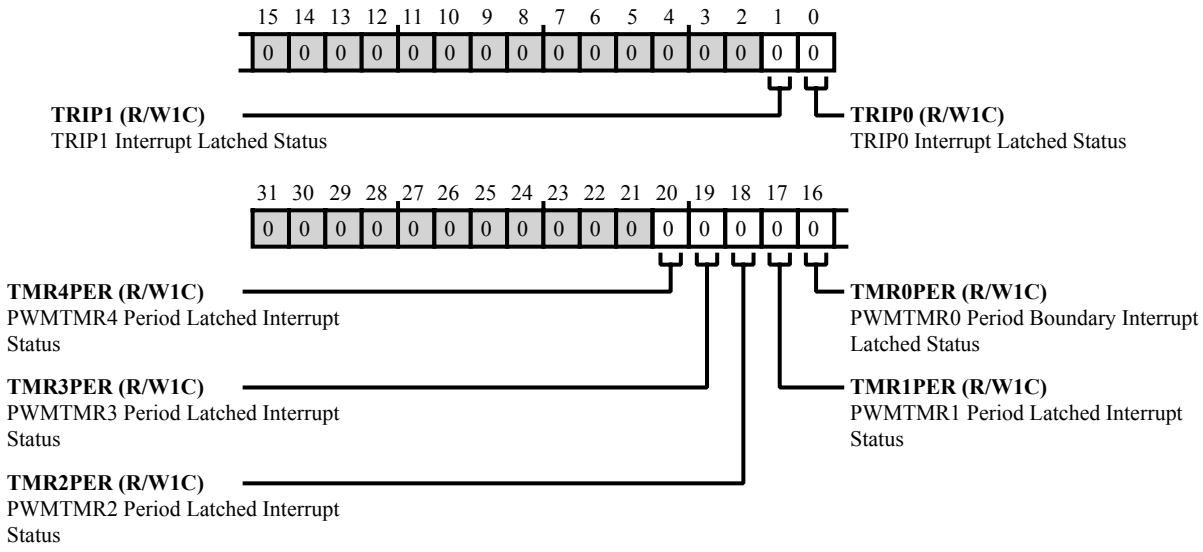

Figure 18-91: PWM_ILAT Register Diagram

Table 18-66: PWM_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/W1C)	TMR4PER	PWMTMR4 Period Latched Interrupt Status. The PWM_ILAT.TMR4PER bit indicates the latched status of the PWMTMR4 period boundary interrupt.	
		0	No Interrupt Latched
		1	Interrupt Latched
19 (R/W1C)	TMR3PER	PWMTMR3 Period Latched Interrupt Status. The PWM_ILAT.TMR3PER bit indicates the latched status of the PWMTMR3 period boundary interrupt.	
		0	No Interrupt Latched
		1	Interrupt Latched
18 (R/W1C)	TMR2PER	PWMTMR2 Period Latched Interrupt Status. The PWM_ILAT.TMR2PER bit indicates the latched status of the PWMTMR2 period boundary interrupt.	
		0	No Interrupt Latched
		1	Interrupt Latched

Table 18-66: PWM_ILAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W1C)	TMR1PER	PWMTMR1 Period Latched Interrupt Status. The `PWM_ILAT.TMR1PER` bit indicates the latched status of the PWMTMR1 period boundary interrupt.	
		0	No Interrupt Latched
		1	Interrupt Latched
16 (R/W1C)	TMR0PER	PWMTMR0 Period Boundary Interrupt Latched Status. The `PWM_ILAT.TMR0PER` bit indicates the latched status of the PWMTMR0 period boundary interrupt.	
		0	No Interrupt Latched
		1	Interrupt Latched
1 (R/W1C)	TRIP1	TRIP1 Interrupt Latched Status. The `PWM_ILAT.TRIP1` bit indicates the latched status of the TRIP1 interrupt.	
		0	No Interrupt Latched
		1	Interrupt Latched
0 (R/W1C)	TRIP0	TRIP0 Interrupt Latched Status. The `PWM_ILAT.TRIP0` bit indicates the latched status of the TRIP0 interrupt.	
		0	No Interrupt Latched
		1	Interrupt Latched

Interrupt Mask Register

The PWM_IMSK register masks (disables) or unmasks (enables) PWM interrupts. When an unmasked interrupt occurs, the PWM latches the interrupt status in the PWM_ILAT register.

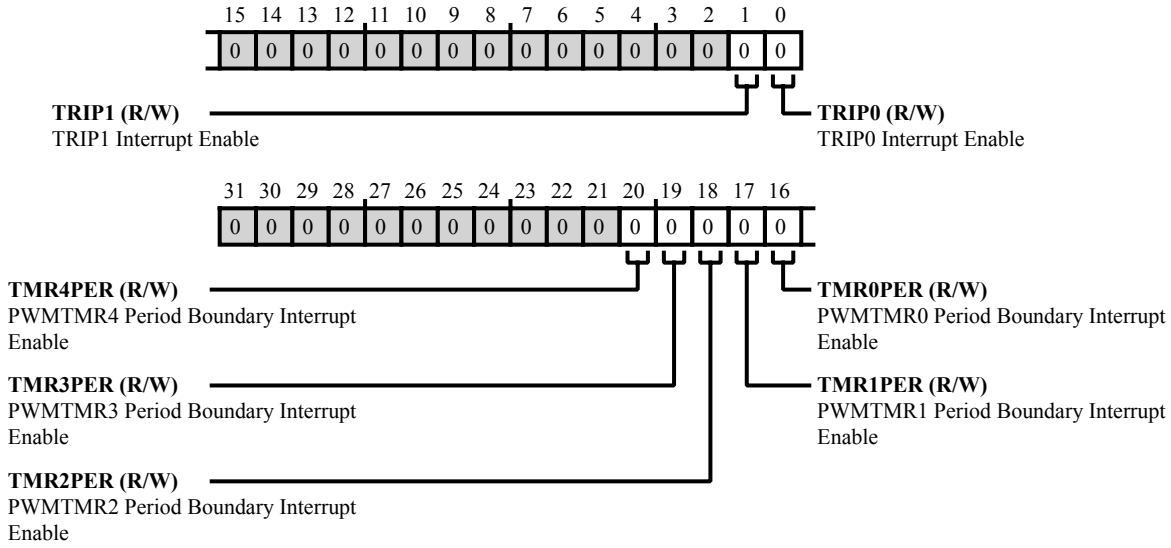

Figure 18-92: PWM_IMSK Register Diagram

Table 18-67: PWM_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/W)	TMR4PER	PWMTMR4 Period Boundary Interrupt Enable. The PWM_IMSK.TMR4PER bit enables (unmasks) the PWMTMR4 period boundary interrupt. This condition occurs when the timers period boundary is reached (PWM_STAT.TMR4PER =1).	
		0	Mask PWMTMR4 Period Interrupt
		1	Unmask PWMTMR4 Period Interrupt
19 (R/W)	TMR3PER	PWMTMR3 Period Boundary Interrupt Enable. The PWM_IMSK.TMR3PER bit enables (unmasks) the PWMTMR3 period boundary interrupt. This condition occurs when the timers period boundary is reached (PWM_STAT.TMR3PER =1).	
		0	Mask PWMTMR3 Period Interrupt
		1	Unmask PWMTMR3 Period Interrupt

Table 18-67: PWM_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	TMR2PER	PWMTMR2 Period Boundary Interrupt Enable. The PWM_IMSK.TMR2PER bit enables (unmasks) the PWMTMR2 period boundary interrupt. This condition occurs when the timers period boundary is reached (PWM_STAT.TMR2PER =1).	
		0	Mask PWMTMR2 Period Interrupt
		1	Unmask PWMTMR2 Period Interrupt
17 (R/W)	TMR1PER	PWMTMR1 Period Boundary Interrupt Enable. The PWM_IMSK.TMR1PER bit enables (unmasks) the PWMTMR1 period boundary interrupt. This condition occurs when the timers period boundary is reached (PWM_STAT.TMR1PER =1).	
		0	Mask PWMTMR1 Period Interrupt
		1	Unmask PWMTMR1 Period Interrupt
16 (R/W)	TMR0PER	PWMTMR0 Period Boundary Interrupt Enable. The PWM_IMSK.TMR0PER bit enables (unmasks) the PWMTMR0 period boundary interrupt. This condition occurs when the timers period boundary is reached (PWM_STAT.TMR0PER =1).	
		0	Mask PWMTMR0 Period Interrupt
		1	Unmask PWMTMR0 Period Interrupt
1 (R/W)	TRIP1	TRIP1 Interrupt Enable. The PWM_IMSK.TRIP1 bit enables (unmasks) the TRIP1 interrupt. This condition occurs when fault input is tripped (PWM_STAT.TRIP1 =1).	
		0	Mask TRIP1 Interrupt
		1	Unmask TRIP1 Interrupt
0 (R/W)	TRIP0	TRIP0 Interrupt Enable. The PWM_IMSK.TRIP0 bit enables (unmasks) the TRIP0 interrupt. This condition occurs when fault input is tripped (PWM_STAT.TRIP0 =1).	
		0	Mask TRIP0 Interrupt
		1	Unmask TRIP0 Interrupt

Status Register

The PWM_STAT register indicates the PWM PWMTRIP1-0 fault and input level status, indicates the Channel A-D fault and self-restart status, and indicates the PWMTMR4-0 phase.

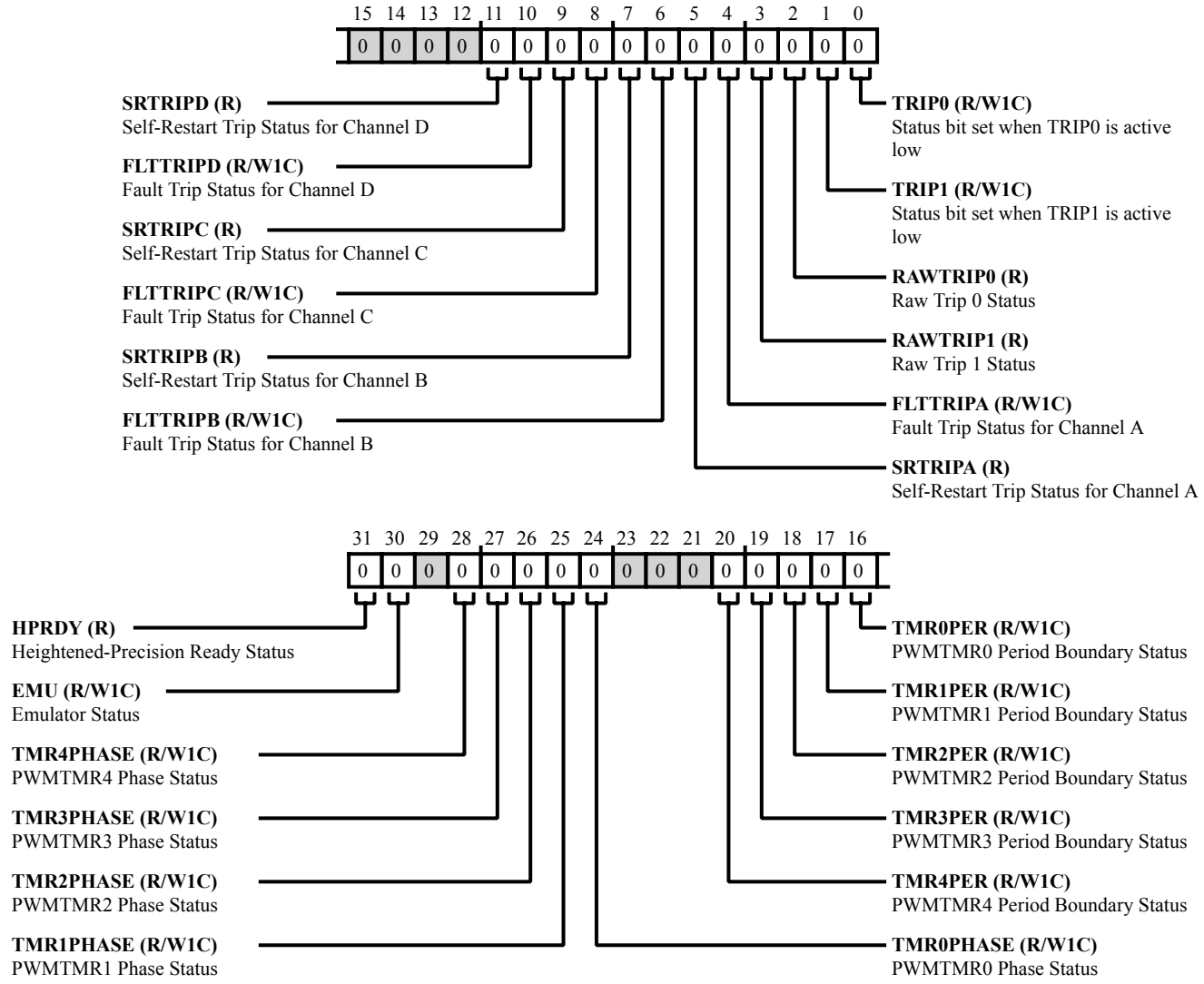

Figure 18-93: PWM_STAT Register Diagram

Table 18-68: PWM_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/NW)	HPRDY	Heightened-Precision Ready Status. The PWM_STAT.HPRDY bit indicates whether or not the PWM is ready for heightened-precision operation.	
		0	HPPWM Not Ready For Operation
		1	HPPWM Ready For Operation
30 (R/W1C)	EMU	Emulator Status.	
28 (R/W1C)	TMR4PHASE	PWMTMR4 Phase Status. The PWM_STAT.TMR4PHASE bit indicates the current phase for the PWMTMR4 waveform.	
		0	1st Half Phase
		1	2nd Half Phase
27 (R/W1C)	TMR3PHASE	PWMTMR3 Phase Status. The PWM_STAT.TMR3PHASE bit indicates the current phase for the PWMTMR3 waveform.	
		0	1st Half Phase
		1	2nd Half Phase
26 (R/W1C)	TMR2PHASE	PWMTMR2 Phase Status. The PWM_STAT.TMR2PHASE bit indicates the current phase for the PWMTMR2 waveform.	
		0	1st Half Phase
		1	2nd Half Phase
25 (R/W1C)	TMR1PHASE	PWMTMR1 Phase Status. The PWM_STAT.TMR1PHASE bit indicates the current phase for the PWMTMR1 waveform.	
		0	1st Half Phase
		1	2nd Half Phase
24 (R/W1C)	TMR0PHASE	PWMTMR0 Phase Status. The PWM_STAT.TMR0PHASE bit indicates the current phase for the PWMTMR0 waveform.	
		0	1st Half Phase
		1	2nd Half Phase

Table 18-68: PWM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/W1C)	TMR4PER	PWMTMR4 Period Boundary Status. The PWM_STAT.TMR4PER bit indicates whether or not the PWMTMR4 period boundary has been reached.	
		0	PWMTMR4 period boundary not reached
		1	PWMTMR4 period boundary reached
19 (R/W1C)	TMR3PER	PWMTMR3 Period Boundary Status. The PWM_STAT.TMR3PER bit indicates whether or not the PWMTMR3 period boundary has been reached.	
		0	PWMTMR3 period boundary not reached
		1	PWMTMR3 period boundary reached
18 (R/W1C)	TMR2PER	PWMTMR2 Period Boundary Status. The PWM_STAT.TMR2PER bit indicates whether or not the PWMTMR2 period boundary has been reached.	
		0	PWMTMR2 period boundary not reached
		1	PWMTMR2 period boundary reached
17 (R/W1C)	TMR1PER	PWMTMR1 Period Boundary Status. The PWM_STAT.TMR1PER bit indicates whether or not the PWMTMR1 period boundary has been reached.	
		0	PWMTMR1 period boundary not reached
		1	PWMTMR1 period boundary reached
16 (R/W1C)	TMR0PER	PWMTMR0 Period Boundary Status. The PWM_STAT.TMR0PER bit indicates whether or not the PWMTMR0 period boundary has been reached.	
		0	PWMTMR0 period boundary not reached
		1	PWMTMR0 period boundary reached
11 (R/NW)	SRTRIPD	Self-Restart Trip Status for Channel D. The PWM_STAT.SRTRIPD bit indicates whether the PWM Channel D self-restart has been tripped. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Channel D Self-Restart Trip Status is "not tripped"
		1	Channel D Self-Restart Trip Status is "tripped"

Table 18-68: PWM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W1C)	FLTTRIPD	Fault Trip Status for Channel D. The PWM_STAT.FLTTRIPD bit indicates whether the PWM Channel D fault has been tripped. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Channel D Fault Trip Status is "not tripped"
		1	Channel D Fault Trip Status is "tripped"
9 (R/NW)	SRTRIPC	Self-Restart Trip Status for Channel C. The PWM_STAT.SRTRIPC bit indicates whether the PWM Channel C self-restart has been tripped. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Channel C Self-Restart Trip Status is "not tripped"
		1	Channel C Self-Restart Trip Status is "tripped"
8 (R/W1C)	FLTTRIPC	Fault Trip Status for Channel C. The PWM_STAT.FLTTRIPC bit indicates whether the PWM Channel C fault has been tripped. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Channel C Fault Trip Status is "not tripped"
		1	Channel C Fault Trip Status is "tripped"
7 (R/NW)	SRTRIPB	Self-Restart Trip Status for Channel B. The PWM_STAT.SRTRIPB bit indicates whether the PWM Channel B self-restart has been tripped. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Channel B Self-Restart Trip Status is "not tripped"
		1	Channel B Self-Restart Trip Status is "tripped"
6 (R/W1C)	FLTTRIPB	Fault Trip Status for Channel B. The PWM_STAT.FLTTRIPB bit indicates whether the PWM Channel B fault has been tripped. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Channel B Fault Trip Status is "not tripped"
		1	Channel A Fault Trip Status is "tripped"
5 (R/NW)	SRTRIPA	Self-Restart Trip Status for Channel A. The PWM_STAT.SRTRIPA bit indicates whether the PWM Channel A self-restart has been tripped. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Channel A Self-Restart Trip Status is "not tripped"
		1	Channel A Self-Restart Trip Status is "tripped"

Table 18-68: PWM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W1C)	FLTTRIPA	Fault Trip Status for Channel A. The PWM_STAT.FLTTRIPA bit indicates whether the PWM Channel A fault has been tripped. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Channel A Fault Trip Status is "not tripped"
		1	Channel A Fault Trip Status is "tripped"
3 (R/NW)	RAWTRIP1	Raw Trip 1 Status. The PWM_STAT.RAWTRIP1 bit indicates the raw input level for the PWM TRIP1 input.	
		0	TRIP1 Level is Low
		1	TRIP1 Level is High
2 (R/NW)	RAWTRIP0	Raw Trip 0 Status. The PWM_STAT.RAWTRIP0 bit indicates the raw input level for the PWM TRIP0 input.	
		0	TRIP0 Level is Low
		1	TRIP0 Level is High
1 (R/W1C)	TRIP1	Status bit set when TRIP1 is active low. The PWM_STAT.TRIP1 bit indicates whether the PWM TRIP1 fault has been tripped with an active-low input.	
		0	TRIP1 status is "not tripped"
		1	TRIP1 status is "tripped" (active low)
0 (R/W1C)	TRIP0	Status bit set when TRIP0 is active low. The PWM_STAT.TRIP0 bit indicates whether the PWM TRIP0 fault has been tripped with an active-low input.	
		0	TRIP0 status is "not tripped"
		1	TRIP0 status is "tripped" (active low)

Sync Pulse Width Register

The PWM_SYNC_WID register selects the pulse width for the external sync pulse available on the PWM_SYNC pin. The relation between the PWM_SYNC_WID register value and the pulse width (T_{PWM_SYNC}) is give by the formula:

PWM_SYNC_WID = (T_{PWM_SYNC} / t_{CK}) -1

For more information about applying the sync pulse width, see the PWM Functional Description section. Note that if the pulse width is changed in between sync pulses, the PWM applies the changed width on the next internal sync pulse. If, while the sync pulse is active, the chosen timer reaches its period boundary, the changed pulse width takes effect on that period boundary.

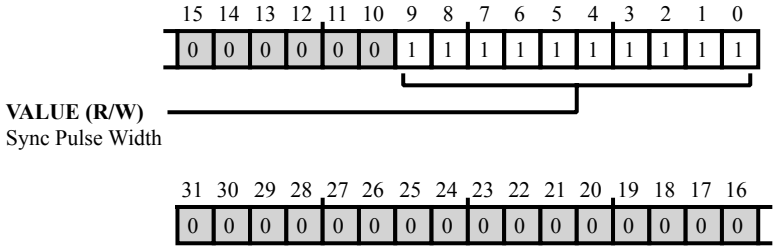

Figure 18-94: PWM_SYNC_WID Register Diagram

Table 18-69: PWM_SYNC_WID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	VALUE	Sync Pulse Width. The PWM_SYNC_WID.VALUE bits select the pulse width for the external sync pulse available on the PWM_SYNC pin.

Timer 0 Period Register

The PWM_TM0 register controls the switch period (T_{SP} of the PWMTMR0 timer. The PWM_TM0 value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

PWM_TM0= (T_{SP}) / 2 x t_{CK}

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that PWM_TM0 values of 0 and 1 are not defined and must not be used when the PWM is enabled.

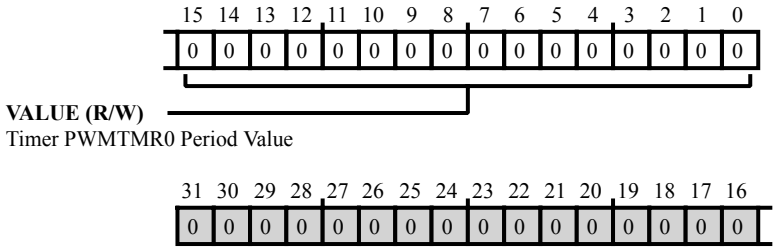

Figure 18-95: PWM_TM0 Register Diagram

Table 18-70: PWM_TM0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR0 Period Value. The PWM_TM0.VALUE bits select the period for the PWMTMR0 timer.

Timer 1 Period Register

The PWM_TM1 register controls the switch period (T_{SP} of the PWMTMR1 timer. The PWM_TM1 value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

PWM_TM1= (T_{SP}) / 2 x t_{CK}

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that PWM_TM1 values of 0 and 1 are not defined and must not be used when the PWM is enabled.

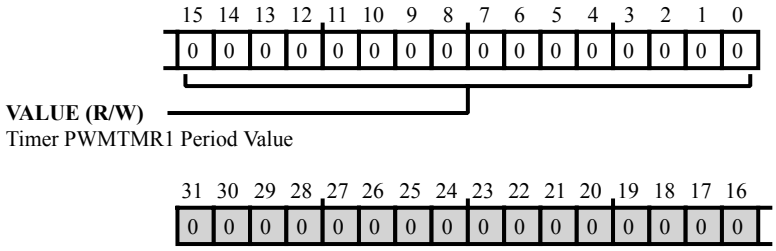

Figure 18-96: PWM_TM1 Register Diagram

Table 18-71: PWM_TM1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR1 Period Value. The PWM_TM1.VALUE bits select the period for the PWMTMR1 timer.

Timer 2 Period Register

The PWM_TM2 register controls the switch period (T_{SP} of the PWMTMR2 timer. The PWM_TM2 value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

PWM_TM1= (T_{SP}) / 2 x t_{CK}

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that PWM_TM2 values of 0 and 1 are not defined and must not be used when the PWM is enabled.

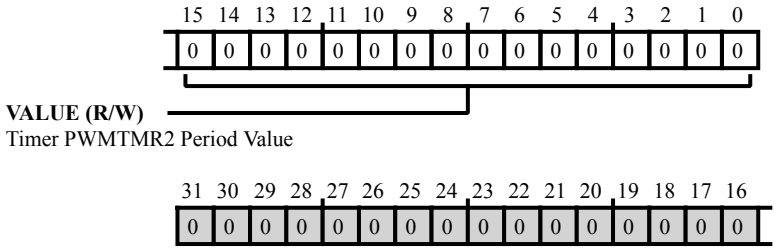

Figure 18-97: PWM_TM2 Register Diagram

Table 18-72: PWM_TM2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR2 Period Value. The PWM_TM2.VALUE bits select the period for the PWMTMR2 timer.

Timer 3 Period Register

The PWM_TM3 register controls the switch period (T_{SP} of the PWMTMR3 timer. The PWM_TM3 value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

PWM_TM3= (T_{SP}) / 2 x t_{CK}

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that PWM_TM3 values of 0 and 1 are not defined and must not be used when the PWM is enabled.

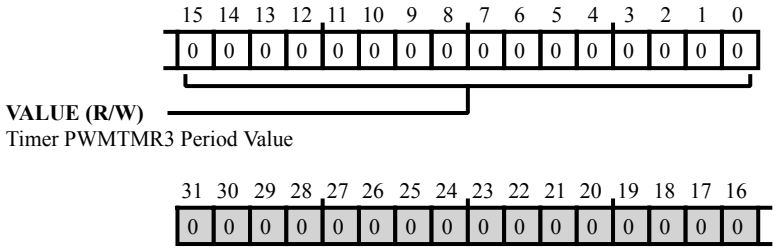

Figure 18-98: PWM_TM3 Register Diagram

Table 18-73: PWM_TM3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR3 Period Value. The PWM_TM3.VALUE bits select the period for the PWMTMR3 timer.

Timer 4 Period Register

The PWM_TM4 register controls the switch period (T_{SP} of the PWMTMR4 timer. The PWM_TM4 value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

PWM_TM4= (T_{SP}) / 2 x t_{CK}

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that PWM_TM4 values of 0 and 1 are not defined and must not be used when the PWM is enabled.

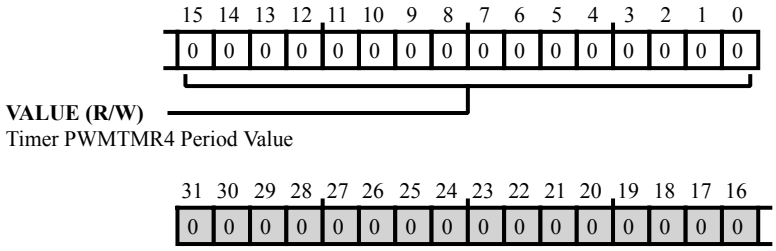

Figure 18-99: PWM_TM4 Register Diagram

Table 18-74: PWM_TM4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR4 Period Value. The PWM_TM4.VALUE bits select the period for the PWMTMR4 timer.

Trip Configuration Register

The PWM_TRIPCFG register configures Channel A, B, C, and D trip operation for trip inputs TRIP0 and TRIP1.

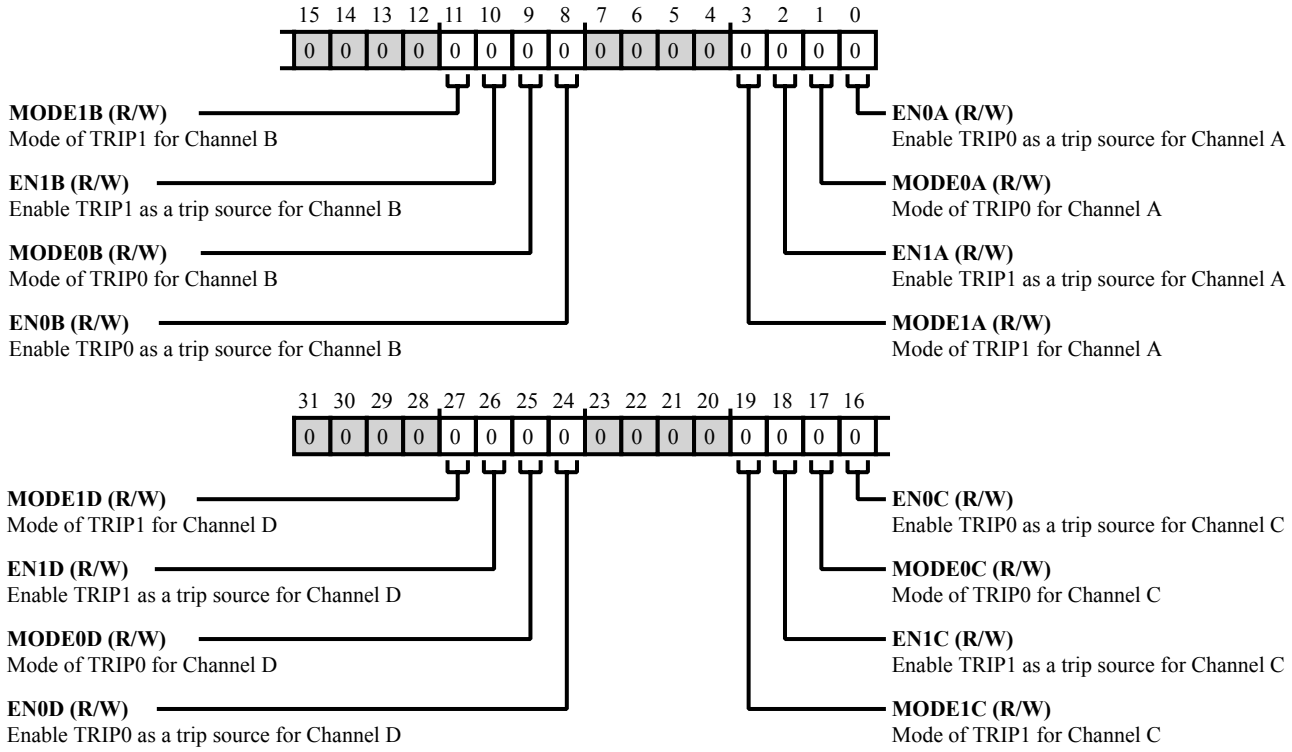

Figure 18-100: PWM_TRIPCFG Register Diagram

Table 18-75: PWM_TRIPCFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
27 (R/W)	MODE1D	Mode of TRIP1 for Channel D. The PWM_TRIPCFG.MODE1D bit selects the trip mode of TRIP1 for Channel D. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP1 Input
		1	Self Restart on TRIP1 Input
26 (R/W)	EN1D	Enable TRIP1 as a trip source for Channel D. The PWM_TRIPCFG.EN1D bit enables TRIP1 as a trip source for Channel D.	
		0	Disable TRIP1 for Channel D
		1	Enable TRIP1 for Channel D

Table 18-75: PWM_TRIPCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
25 (R/W)	MODE0D	Mode of TRIP0 for Channel D. The PWM_TRIPCFG.MODE0D bit selects the trip mode of TRIP0 for Channel D. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP0 Input
		1	Self Restart on TRIP0 Input
24 (R/W)	EN0D	Enable TRIP0 as a trip source for Channel D. The PWM_TRIPCFG.EN0D bit enables TRIP0 as a trip source for Channel D.	
		0	Disable TRIP0 for Channel D
		1	Enable TRIP0 for Channel D
19 (R/W)	MODE1C	Mode of TRIP1 for Channel C. The PWM_TRIPCFG.MODE1C bit selects the trip mode of TRIP1 for Channel C. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP1 Input
		1	Self Restart on TRIP1 Input
18 (R/W)	EN1C	Enable TRIP1 as a trip source for Channel C. The PWM_TRIPCFG.EN1C bit enables TRIP1 as a trip source for Channel C.	
		0	Disable TRIP1 for Channel C
		1	Enable TRIP1 for Channel C
17 (R/W)	MODE0C	Mode of TRIP0 for Channel C. The PWM_TRIPCFG.MODE0C bit selects the trip mode of TRIP0 for Channel C. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP0 Input
		1	Self Restart on TRIP0 Input
16 (R/W)	EN0C	Enable TRIP0 as a trip source for Channel C. The PWM_TRIPCFG.EN0C bit enables TRIP0 as a trip source for Channel C.	
		0	Disable TRIP0 for Channel C
		1	Enable TRIP0 for Channel C
11 (R/W)	MODE1B	Mode of TRIP1 for Channel B. The PWM_TRIPCFG.MODE1B bit selects the trip mode of TRIP1 for Channel B. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP1 Input
		1	Self Restart on TRIP1 Input

Table 18-75: PWM_TRIPCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W)	EN1B	Enable TRIP1 as a trip source for Channel B. The PWM_TRIPCFG.EN1B bit enables TRIP1 as a trip source for Channel B.	
		0	Disable TRIP1 for Channel B
		1	Enable TRIP1 for Channel B
9 (R/W)	MODE0B	Mode of TRIP0 for Channel B. The PWM_TRIPCFG.MODE0B bit selects the trip mode of TRIP0 for Channel B. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP0 Input
		1	Self Restart on TRIP0 Input
8 (R/W)	EN0B	Enable TRIP0 as a trip source for Channel B. The PWM_TRIPCFG.EN0B bit enables TRIP0 as a trip source for Channel B.	
		0	Disable TRIP0 for Channel B
		1	Enable TRIP0 for Channel B
3 (R/W)	MODE1A	Mode of TRIP1 for Channel A. The PWM_TRIPCFG.MODE1A bit selects the trip mode of TRIP1 for Channel A. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP1 Input
		1	Self Restart on TRIP1 Input
2 (R/W)	EN1A	Enable TRIP1 as a trip source for Channel A. The PWM_TRIPCFG.EN1A bit enables TRIP1 as a trip source for Channel A.	
		0	Disable TRIP1 for Channel A
		1	Enable TRIP1 for Channel A

Table 18-75: PWM_TRIPCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	MODE0A	Mode of TRIP0 for Channel A. The PWM_TRIPCFG.MODE0A bit selects the trip mode of TRIP0 for Channel A. In fault-trip mode (PWM_TRIPCFG.MODE0A =0), after the input is tripped, the trip status appears in the corresponding channels fault-trip status bit (for example, PWM_STAT.FLTTRIPA), and the PWM immediately shuts down outputs of that channel. After a fault trip occurs, when the trip condition is no longer active, the processor may cause channel outputs to resume by completing a write-1-to-clear the corresponding fault-trip status bit. The raw (input level) trip input state is available from the PWM_STAT.RAWTRIP0 and PWM_STAT.RAWTRIP0 bits. In self-restart mode (PWM_TRIPCFG.MODE0A =1), after the input is tripped, the trip status appears in the corresponding channels self-restart status bit (for example, PWM_STAT.SRTRIPA), and the PWM immediately shuts down outputs of that channel. On the next timer period boundary (of the PWMTMRx used by that channel), if the trip condition is not active, the PWM clears the status and restarts the channels output.	
		0	Fault Trip on TRIP0 Input
		1	Self Restart on TRIP0 Input
0 (R/W)	EN0A	Enable TRIP0 as a trip source for Channel A. The PWM_TRIPCFG.EN0A bit enables TRIP0 as a trip source for Channel A.	
		0	Disable TRIP0 for Channel A
		1	Enable TRIP0 for Channel A

19 Universal Asynchronous Receiver/Transmitter (UART)

The UART module is a full-duplex peripheral compatible with PC-style industry-standard UARTs. The UART converts data between serial and parallel formats. The serial communication follows an asynchronous protocol that supports various word lengths, stop bits, bit rates, and parity-generation options. The UART includes interrupt-handling hardware. Multiple events can generate interrupts.

The UART is logically compliant to EIA-232E, EIA-422, EIA-485 and LIN standards, but usually requires external transceiver devices to meet electrical requirements. In IrDA (Infrared Data Association) mode, the UART meets the half-duplex IrDA SIR (9.6/115.2 Kbps rate) protocol. In multi-drop bus mode, the UART meets the full-duplex MDB/ICP v2.0 protocol.

The UART module supports partial modem status and control functionality to allow for hardware flow control.

The UART is a DMA-capable peripheral with separate transmit and receive DMA master channels. The use of DMA requires minimal software intervention as the DMA engine moves the data. The UART can also use a programmed core mode of operation. The core mode requires software management of the data flow using either interrupts or polling.

The UART can use one of the peripheral timers for a hardware-assisted auto-baud detection mechanism. The timers are external to the UART.

UART Features

Each UART includes the following features.

- 5–8 data bits
- Programmable extra stop bit and programmable extra half-stop bit
- Even, odd, and sticky parity bit options
- Extra 8-stage receive FIFO with programmable threshold interrupt
- Flexible transmit and receive interrupt timing
- 3 interrupt outputs for receive, transmit, and status

- Independent DMA operation for receive and transmit
- Programmable automatic request to send (RTS)/clear to send (CTS) hardware flow control
- False start bit detection
- SIR IrDA operation mode
- MDB/ICP v2.0 operation mode
- Internal loopback
- Improved bit rate granularity
- LIN break command/Inter-frame gap transmission support

Table 19-1: UART Specifications

Feature	Availability
Protocol	
Master-Capable	Yes
Slave-Capable	Yes
Transmission Simplex	Yes
Transmission Half-Duplex	Yes
Transmission Full-Duplex	Yes
Access Type	
Data Buffer	Yes
Core Data Access	Yes
DMA Data Access	Yes
DMA Channels	2 (per UART Port)
DMA Descriptor	Yes
Boot Capable	Yes (Slave Mode)
Local Memory	No
Clock Operation	SYSCLK/16

UART Functional Description

The following sections provide details on the UARTs functionality.

ADSP-CM40x UART Register List

The Universal Asynchronous Receiver/Transmitter module (UART) is a full-duplex peripheral compatible with PC-style industry-standard UARTs. The UART converts data between serial and parallel formats. The serial communication follows an asynchronous protocol that supports various word length, stop bit, parity, and interrupt generation

options. A set of registers governs UART operations. For more information on UART functionality, see the UART register descriptions.

Table 19-2: ADSP-CM40x UART Register List

Name	Description
UART_CLK	Clock Rate Register
UART_CTL	Control Register
UART_IMSK	Interrupt Mask Register
UART_IMSK_CLR	Interrupt Mask Clear Register
UART_IMSK_SET	Interrupt Mask Set Register
UART_RBR	Receive Buffer Register
UART_RSR	Receive Shift Register
UART_RXCNT	Receive Counter Register
UART_SCR	Scratch Register
UART_STAT	Status Register
UART_TAIP	Transmit Address/Insert Pulse Register
UART_THR	Transmit Hold Register
UART_TSR	Transmit Shift Register
UART_TXCNT	Transmit Counter Register

ADSP-CM40x UART Interrupt List

Table 19-3: ADSP-CM40x UART Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
27	UART0_TXDMA_ERR	UART0 DMA Channel 4 Error	Level	
28	UART0_RXDMA_ERR	UART0 DMA Channel 5 Error	Level	
33	UART1_TXDMA_ERR	UART1 DMA Channel 10 Error	Level	
34	UART1_RXDMA_ERR	UART1 DMA Channel 11 Error	Level	
35	UART2_TXDMA_ERR	UART2 DMA Channel 12 Error	Level	
36	UART2_RXDMA_ERR	UART2 DMA Channel 13 Error	Level	
67	UART0_STAT	UART0 Status	Level	
68	UART0_TXDMA	UART0 Transmit DMA Transfer Complete	Level	4
69	UART0_RXDMA	UART0 Receive DMA Transfer Complete	Level	5
78	UART1_STAT	UART1 Status	Level	

Table 19-3: ADSP-CM40x UART Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
79	UART1_TXDMA	UART1 Transmit DMA Transfer Complete	Level	10
80	UART1_RXDMA	UART1 Receive DMA Transfer Complete	Level	11
87	UART2_STAT	UART2 Status	Level	
88	UART2_TXDMA	UART2 Transmit DMA Transfer Complete	Level	12
89	UART2_RXDMA	UART2 Receive DMA Operation Complete	Level	13

UART Block Diagram

The ***UART Block Diagram*** figure shows a simplified block diagram of one UART module and how it interconnects to the processor system.

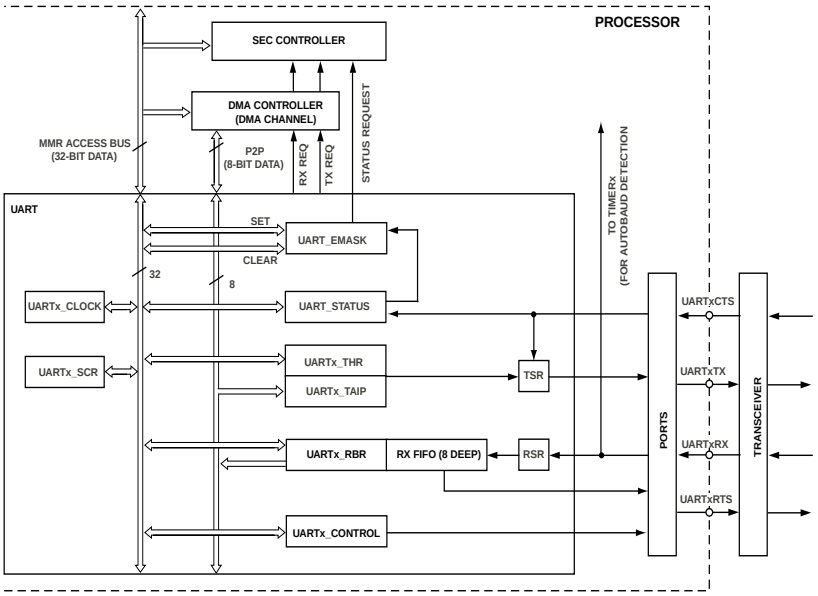

Figure 19-1: UART Block Diagram

UART Architectural Concepts

The following sections provide information about the UART architecture.

Internal Interface

The UART is a DMA-capable peripheral with support for separate transmit and receive DMA master channels. It operates in either DMA or programmed core modes. The core mode requires software management of the data flow

using either interrupts or polling. The DMA method requires minimal software intervention, as the DMA engine itself moves the data. The UART_RBR and UART_THR registers also connect to one of the peripheral DMA buses (8-bit data width).

All UART registers are 32 bits wide and the registers connect to the peripheral MMR bus. Not all MMRs can be used and unused bits are zero-filled. The UART has three interrupt outputs described as follows.

- The transmit and receive request outputs can function as DMA requests and connect to the DMA controller. Therefore, if the DMA is not enabled, the DMA controller simply forwards the request to the system event controller (SEC).
- The status interrupt output connects directly to the SEC. On many processors, the alternative capture input (TIMER_ACI[n]) of one of the GP timers also senses the UART_RX pin. When configured in capture mode, the processor can then use the GP timer to detect the bit rate of the received signal.

External Interface

Each UART features a UART_RX (receive) pin and a UART_TX (transmit) pin available through the general-purpose ports. These two pins usually connect to an external transceiver device that meets the electrical requirements of full-duplex or half-duplex standards. (For example, EIA-232, EIA-422, 4-wire EIA-485 for full-duplex or 2-wire EIA-485, LIN for half-duplex). Additionally, the UART features a pair of clear-to-send, input pins (UART_CTS), and request-to-send, output pins (UART_RTS) for hardware flow control. UART signals are multiplexed with other functions at the pin level.

Hardware Flow Control

To prevent the UART transmitter from sending data while the receiving counterpart is not ready, the UART features a UART_RTS/UART_CTS hardware flow control mechanism. The UART_RTS signal is an output that connects to the UART_CTS input of the communication partner. If data transfer is bidirectional, the figure shows the ***UART Hardware Flow Control*** handshake.

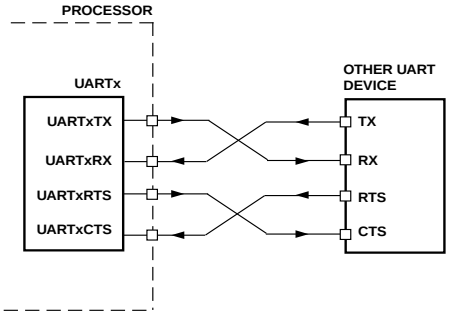

Figure 19-2: UART Hardware Flow Control

In both DMA and core mode, the receiver can de-assert the UART_RTS signal to indicate that its receive buffer is almost full. Continued data transfers can cause an overrun error. The transmitter pauses when the UART_CTS input is in a de-asserted state. In this state, the transmitter completes transmission of the data currently held in the transmit shift register (UART_TSR) but it does not continue with the data in the transmit hold register (UART_THR). If

the UART_CTS pin is asserted again, the transmitter resumes and loads the content of UART_THR register into the UART_TSR register.

NOTE:

Bit Rate Generation

The peripheral clock (SYSCLK) and the 16-bit divisor in the UART_CLK register characterize the sample clock. The UART uses the UART_CTL.EN bit to enable the clock. By default, every serial bit is oversampled 16 times. The bit clock is 1/16th of the sample clock. If not in IrDA mode, the bit clock can equal the sample clock if the UART_CLK.EDBO bit is set, so that the following equation applies:

$$\text{Bit Rate} = \text{SYSCLK}/16^{(1-\text{EDBO})} \times \text{Divisor}$$

ADSP-CM40x Processor Example

The UART Bit Rate Examples with 100 MHz SYSCLK table provides example divide factors required to support standard baud rates at an SYSCLK of 100 MHz.

Table 19-4: UART Bit Rate Examples with 100 MHz SYSCLK

	D Factor = 16			D Factor = 1		
Bit Rate	**DL**	**Actual**	**% Error**	**DL**	**Actual**	**% Error**
2400	2604	2400.15	0.006	41667	2399.98	0.001
4800	1302	4800.31	0.006	20833	4800.08	0.002
9600	651	9600.61	0.006	10417	9599.69	0.003
19200	326	19171.78	0.147	5208	19201.23	0.006
38400	163	38343.56	0.147	2604	38402.46	0.006
57600	109	57339.45	0.452	1736	57603.69	0.006
115200	54	115740.74	0.469	868	115207.37	0.006
921600	7	892857.14	3.119	109	917431.19	0.452
1500000	4	1562500	4.167	67	1492537.31	0.498
3000000	2	3125000	4.167	33	3030303.03	1.01
6250000	1	6250000	0	16	6250000	0

NOTE: Careful selection of SYSCLK frequencies—that is, even multiples of desired bit rates— can result in lower error percentages. Setting the bit clock equal to the sample clock (UART_CLK.EDBO=1) improves bit rate granularity and enables the bit clock to more closely match the bit rate of the communication partner. The disadvantage to this configuration is that the power dissipation is higher and the sample points are not always as accurate. Therefore, use UART_CLK.EDBO=1 mode only when bit rate accuracy is not acceptable in UART_CLK.EDBO=0 mode. The UART_CLK.EDBO=1 mode is not intended to increase operation speed beyond the electrical limitations of the UART transfer protocol.

Autobaud Detection

At the chip level, the UART_RX pin is typically routed to an alternate capture input (TIMER_ACI[n]) of a general-purpose timer. When working in width capture mode, the processor uses this general-purpose timer to detect the bit rate applied to the UART_RX pin automatically by an external device. It often uses the capture capabilities of the timer to supervise the bit rate at run time. If the UART communicates with any device supplied by a weak clock oscillator that drifts over time, the processor can then readjust its UART bit rate dynamically, as required.

Often, the processor uses autobaud detection for initial bit rate negotiations where it is most likely a slave device waiting for the host to send a predefined autobaud character. This situation is common for UART booting. Do not enable the UART_CTL.EN bit while autobaud detection is in-process, to prevent the UART from starting a receive operation with incorrect bit rate matching. Alternatively, set the UART_CTL.LOOP_EN bit to disconnect the UART from its UART_RX pin.

A software routine can detect the pulse widths of serial stream bit cells. The sample base of the timer is synchronous with the UART operation (all derived from the same SYSCLK). The UART uses pulse widths to calculate the bit rate divider as follows:

A software routine can detect the pulse widths of serial stream bit cells. The sample base of the timer is synchronous with the UART operation (all derived from the same SYSCLK). The UART uses pulse widths to calculate the bit rate divider as follows:

Divisor = TIMER_TMR[n]_WID/$16^{(1-EDBO)}$ × Number of captured UART bits

To increase the number of timer counts and the resolution of the captured signal, do not measure just the pulse width of a single bit. Instead, enlarge the pulse of interest over more bits. Traditionally, a NULL character (ASCII 0x00) is used in autobaud detection, as shown in the ***Autobaud Detection*** figure.

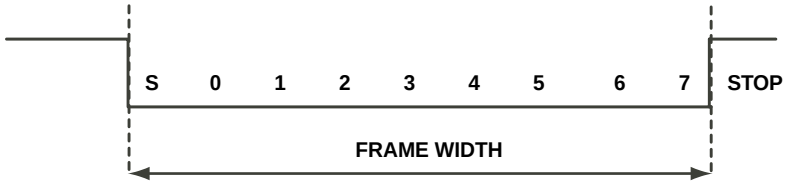

Figure 19-3: Autobaud Detection

Because the example frame encloses 8 data bits and 1 start bit, apply the following formula.

Divisor = TIMER_TMR[n]_WID/$16^{(1-EDBO)}$ × 9

NOTE: For processor-specific mapping of timer alternate capture inputs to the UARTs of the processor, see "Width Capture (WIDCAP) Mode" in the "*General-Purpose Timer (TIMER)*" chapter.

Real receive signals often have asymmetrical falling and rising edges, and the sampling logic level is not exactly in the middle of the signal voltage range. At higher bit rates, such pulse-width-based autobaud detection do not always return adequate results without extra conditioning of the analog signal. Measure signal periods to work around this issue.

For example, predefine ASCII character “@” (0x40) as the autobaud detection character and measure the period between two subsequent falling edges. As shown in the ***Autobaud Detection Character 0x40*** figure, measure the period between the falling edge of the start bit and the falling edge after bit 6. Since this period encloses 8 bits, apply the following formula.

Divisor = `TIMER_TMR[n]_PER`/$16^{(1-EDBO)}$ × 8

or:

- Divisor = `TIMER_TMR[n]_PER`>> 7, if `UART_CLK.EDBO`=0
- Divisor = `TIMER_TMR[n]_PER`>> 3, if `UART_CLK.EDBO`=1

The ***Autobaud Detection Character 0x40*** figure shows the ASCII “@” (0x40) detection character.

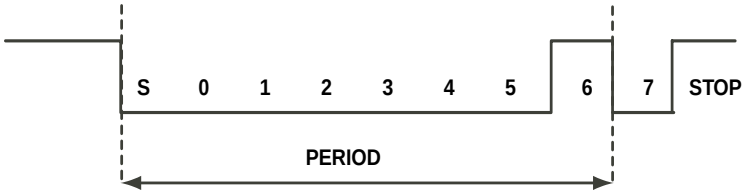

Figure 19-4: Autobaud Detection Character 0x40

UART Debug Features

The UART can automatically calculate and transmit a parity bit. The ***UART Parity*** table summarizes parity behavior assuming 8-bit data words (`UART_CTL.WLS`=b#11).

Table 19-5: UART Parity

PEN	STP	EPS	Data (hex)	Data (binary, LSB first)	Parity
0	x	x	x	x	None
1	0	0	0x60	0000 0110	1
1	0	0	0x57	1110 1010	0
1	0	1	0x60	0000 0110	0
1	0	1	0x57	1110 1010	1
1	1	0	x	x	1
1	1	1	x	x	0

The two force error bits, `UART_CTL.FPE` and `UART_CTL.FFE`, are intended for test purposes. They are useful for debugging software, especially in loopback mode.

The UART can be set to internal loopback mode (`UART_CTL.LOOP_EN`=1). Loopback mode disconnects the input of the receiver from the receive pin and internally redirects the transmit output to the receiver. The transmit pin remains active and continues to transmit data externally as well. Loopback mode also forces the `UART_RTS` pin to

deassert, disconnects the UART_STAT.CTS bit from the $\overline{\text{UART\_CTS}}$ input pin, and connects the internal version of UART_RTS to the UART_STAT.CTS bit.

Additionally, the UART_TX pin can be forced to zero asynchronously using the UART_CTL.SB bit.

UART Operating Modes

The following sections describe the main operating modes of the UART.

- UART Mode
- IrDA SIR Mode
- Multi-Drop Bus Mode

UART Mode

The UART mode follows an asynchronous serial communication protocol with these options:

- 1 start bit
- 5–8 data bits
- Address bit (available in MDB mode only)
- None, even, odd or sticky parity
- 1, 1½, or 2 stop bits (1½ stop bits valid only in 5-bit word length)

The UART_CTL register controls the format of received and transmitted character frames. Data is always transmitted and received with the least significant bit (LSB) first.

The ***Bit Stream on a UART TX Pin Transmitting an "S" Character (0x53)*** figure shows a typical physical bit stream measured on a UART_TX pin.

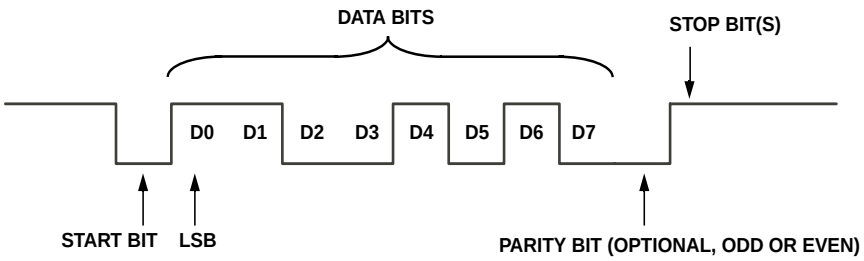

Figure 19-5: Bit Stream on a UART TX Pin Transmitting an "S" Character (0x53)

IrDA SIR Mode

The UART also supports serial data communication by way of infrared signals, according to the recommendations of the Infrared Data Association (IrDA). The physical layer known as IrDA SIR (9.6/115.2 Kbps rate) is based on return-to-zero-inverted (RZI) modulation. The UART does not support pulse position modulation.

Using the 16x data rate clock, RZI modulation is achieved by inverting and modulating the non-return-to-zero (NRZ) code normally transmitted by the UART. On the receive side, the UART uses a 16x clock to determine an IrDA pulse sample window, from which it recovers the RZI modulated NRZ code.

NOTE: The UART_CLK.EDBO bit is not valid in IrDA mode. Clear (=0) this bit in this mode.

Multi-Drop Bus Mode

The UART uses a protocol for point-to-point connections as well as in networks where the EIA-485 standard is representative of UART-based bus systems. (The EIA-232 standard defines point-to-point connections). In such networks, node addressing is important.

In a multidrop bus (MDB) network, for example, an address bit enhances the UART frame. The address bit is inserted between the data bits and the optional parity bit. To configure the UART for MDB mode, set the mode of operation bits (UART_CTL.MOD [5:4]) to 01.

By convention, the address bit is transmitted low for regular data bytes. When set, it marks special address bytes that require the attention of all nodes on the network.

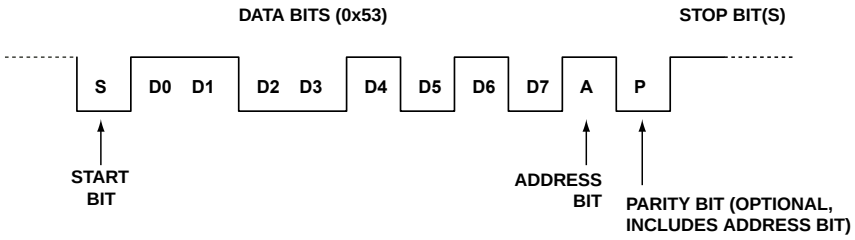

Figure 19-6: UART Frame with Address Bit

All transmit operations are processed through the transmit buffer register (UART_THR), so all DMA data transmissions clear the address bit. If data is written to the transmit address or insert pulse register (UART_TAIP) instead, the same transmit operation is initiated with the only exception that the address bit is sent high.

The UART uses the UART_STAT.ADDR bit of the receiver to signal whether the previously received frame had the address bit set or not. Hardware updates it every time a new frame is received. When the enable address word interrupt bit (UART_IMSK.EAWI) is set, the reception of an address byte triggers a special status interrupt.

The address sticky bit (UART_STAT.ASTKY) is the sticky version of the UART_STAT.ADDR bit. Hardware sets it whenever the UART_STAT.ADDR bit is set. Software can clear the UART_STAT.ASTKY bit with a W1C operation.

In MDB mode, only address bytes progress to the receive FIFO by default. Data bytes are gated unless the UART_STAT.ASTKY bit is set. The receiver ignores all traffic on the UART bus. This way, the processor can go into low-power mode and interrupt activity does not load the processor every time a frame is transmitted on the UART bus. If, however, an address frame is transmitted, the receiver immediately samples all further traffic. A software routine can analyze the received data, decide whether it was of relevance for the local network node, and W1C the UART_STAT.ASTKY bit if it was not.

Software can overrule of the hardware address frame detection by setting the UART_STAT.ADDR bit and (indirectly) the UART_STAT.ASTKY bit with a W1S operation.

The MDB mode follows an asynchronous serial communication protocol with the following options.

- 1 start bit
- 5–8 data bits
- Address bit
- None, even, odd or sticky parity
- 1, 1½, or 2 stop bits (1½ stop bits valid only in 5-bit word length)

NOTE: If the address bit and parity bit are both enabled, the parity check and generation includes the address bit in its parity calculation.

UART Data Transfer Modes

The UART can transfer data using both the core and DMA. Receive and transmit paths operate independently except that the bit rate and the frame format are identical for both transfer directions. Transmit and receive channels are both buffered. The UART_THR register buffers the transmit shift register (UART_TSR) and the UART_RBR register buffers the receive shift register (UART_RSR).

UART Mode Transmit Operation (Core)

In core mode, the processor core moves data to and from the UART. A write to the UART_THR register initiates the transmit operation. If no former operation is pending, the UART_THR register passes the data immediately to the UART_TSR register. There, it is shifted out at the bit rate characterized by the UART_CTL register, with start, stop, and parity bits appended as defined by the UART_CTL register.

The UART_THR register and the UART_TSR register can be modeled as a two-stage transmit buffer. The least significant bit (LSB) always transmits first. This bit is bit 0 of the value written to the UART_THR register.

UART Mode LIN Break Command

Some UART-based protocols demand synchronization methods that are not native to standard UART implementations. For example, the Local Interconnect Network (LIN) protocol requires a low-pulse of well-defined transmit length as a prologue to every multi-byte message. Its length must be at least 13 bit-times.

With previous UARTs, there were two options to implement this protocol:

- A null byte is transmitted with a temporarily lowered bit rate, or
- A software counter generates the period and the asynchronous set break (SB) mechanisms pull the transmit pin low

Since both methods have their disadvantages, the newer UART introduces a new inter-frame gap technique.

The feature is not available in MDB or IrDA operating modes. However, in standard UART mode (bits UART_CTL.MOD [5:4] =00), a write to the UART_TAIP register initiates the transmission of an inter-frame pulse. If the transmit buffer is not empty, the UART first transmits all bytes in the queue. It only initiates with pulse generation after the last stop bit of the last byte has been shifted out.

The value written into the UART_TAIP register defines the nature and the duration of the transmitted pulse. Bits [6:0] control the duration in bit-times and bit [7] controls the value (duration = UART_TAIP[6:0] / UART_CLK[15:0]). If UART_TAIP[7] is set, and an active high pulse is issued, the number of stop bits is extended. If UART_TAIP[7] is cleared, a low pulse is generated. Invert the polarity using the UART_CTL.FCPOL bit. Writing a value of 13 into the UART_TAIP register generates the break command as required by the LIN protocol.

NOTE: If the UART_CTL.TPOLC bit is enabled, an inverted most-significant bit can be transmitted.

NOTE: If another transmission is pending (in the UART_TSR register), the UART_TAIP initiated pulse is queued until after all pending operations have finished and all stop bits are transmitted.

The transmission of break command/inter-frame gap precedes transmission of the number of stop bits as set in the UART_CTL.STB and UART_CTL.STBH bit fields.

The UART receiver can detect break commands through the break indicator (UART_STAT.BI) flag. This flag reports that an entire UART frame has been received in low state. It does not report whether the duration of the received low pulse was exact or at least 13 bit-times as LIN masters transmit. Typically, the break indicator meets LIN requirements. The processor can use GP timers to determine the pulse width more precisely, if necessary.

Each UART_RX pin is also routed to a GP timer through its alternate capture input (TACI). This functionality is not only useful for bit rate detection (*autobaud*) but also helps to measure the pulse widths precisely on the UART_RX input. Additionally, the GP timers can issue an interrupt or a fault condition when the received pulse width is shorter than a bit time or longer than the worst-case break condition. The windowed watchdog width mode of the GP timers controls this functionality.

UART Mode Receive Operation (Core)

The receive operation uses the same data format as the transmit configuration except that one valid stop bit is always sufficient. The UART_CTL.STB and UART_CTL.STBH bits have no impact on the receiver.

The UART receiver senses the falling edges of the receive input. When it detects an edge, the receiver starts sampling the input according to settings in the UART_CLK register. The receiver samples the start bit (majority sampling) close to its midpoint. If sampled low, it assumes a valid start condition. Otherwise, it discards the detected falling edge.

After detection of the start bit, the received word is shifted into the UART_RSR register.

After the corresponding stop bit is received, the content of the UART_RSR register is transferred to the 8-deep receive FIFO and is accessible by reading the UART_RBR register.

The receive FIFOs and the UART_RBR register act as a 9-stage receive buffer. If the stop bit of the ninth word is received before software reads the UART_RBR register, an overrun error is reported. Overruns protect data in the UART_RBR register and the receive FIFO from being overwritten by further data until the software clears the

UART_STAT.OE bit. However, the data in the UART_RSR register is immediately destroyed as soon as the overrun occurs

The sampling clock is 16 times faster than the bit clock. The receiver oversamples every bit 16 times and makes a majority-decision based on the middle three samples. This functionality improves immunity against noise and hazards on the line. The receiver disregards spurious pulses of less than two times the sampling clock period.

Normally, the receiver samples every incoming bit at exactly the 7th, 8th and 9th sample clock. If, however, the UART_CLK.EDBO bit is set to 1, the receiver samples bits roughly at 7/16th, 8/16th, and 9/16th of their period. This configuration achieves better bit rate granularity and accuracy as required at high operation speeds. Hardware design must ensure that the incoming signal is stable between 6/16th and 10/16th of the nominal bit period.

Reception starts when the UART receiver detects a falling edge on the UART_RX input pin. The receiver attempts to see a start bit. The data is shifted into the UART_RSR register. After the ninth sample of the first, the receiver processes the stop bit and copies the received data to the 8-stage receive FIFO. The UART_RSR recovers for further data reception.

The receiver samples data bits close to their midpoint. Because the receiver clock is typically asynchronous to the data rate of the transmitter, the sampling point can drift relative to the center of the data bits. The sampling point is synchronized again with each start bit, so the error accumulates only over the length of a single word. The polarity of received data is selectable, using the UART_CTL.RPOLC bit.

NOTE: The receiver checks for only a single stop bit. After the third sample of the first stop bit has been received (at time 9/16th of the stop bit duration), the receiver immediately acts (status update). It then prepares for new falling edge detection (start detection).

IrDA Transmit Operation

To generate the IrDA pulse transmitted by the UART, the normal NRZ output of the transmitter is first inverted if the UART_CTL.TPOLC bit is configured for active-low operation. In this configuration, a 0 is transmitted as a high pulse of 16 UART clock periods and a 1 is transmitted as a low pulse for 16 UART clock periods. Then, six UART clock periods delay the leading edge of the pulse. Similarly, eight UART clock periods truncate the trailing edge of the pulse. For a 16-cycle UART clock period, this operation results in the final representation of the original 0 as a high pulse of only 3/16 clock periods. The ***IrDA Transmit Pulse*** figure shows how the pulse is centered around the middle of the bit time. The final IrDA pulse is fed to the off-chip infrared driver.

This modulation approach ensures a pulse width output from the UART of three cycles high out of every 16 UART clock cycles. As shown in the ***IrDA Transmit Pulse*** figure, the error terms associated with the bit rate generator are small and well within the tolerance of most infrared transceiver specifications.

NOTE: In IrDA mode, writes to the UART_TAIP register are equivalent to writes to the UART_THR register.

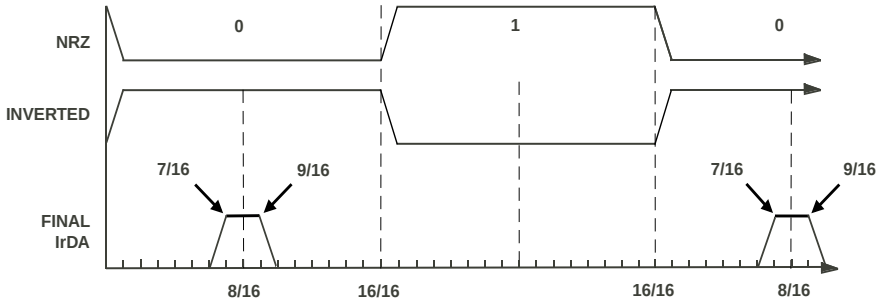

Figure 19-7: IrDA Transmit Pulse

IrDA Receive Operation

The IrDA receiver function is more complex than the transmit function. The receiver must discriminate the IrDA pulse and reject noise. The receiver looks for the IrDA pulse in a narrow window centered around the middle of the expected pulse.

Glitch filtering is accomplished by counting 16 system clocks from the time the receiver detects an initial pulse. If the pulse is absent when the counter expires, the receiver interprets it as a glitch. Otherwise, the receiver interprets it as a 0. This assessment is acceptable because glitches originating from on-chip capacitive cross-coupling typically do not last for more than a fraction of the system clock (SYSCLK) period. Appropriate shielding avoids sources outside of the chip and not part of the transmitter. The only other source of a glitch is the transmitter itself. The processor relies on the transmitter to perform within specification. If the transmitter violates the specification, unpredictable results can occur. The 4-bit counter adds an extra level of protection at a minimal cost.

NOTE: Because SYSCLK can change across systems, the longest glitch tolerated is inversely proportional to the SYSCLK frequency.

A counter that is clocked at the 16x bit-time sample clock determines the receive sampling window. The sampling window is resynchronized with each start bit by centering the sampling window around the start bit.

The polarity of receive data is selectable, using the `UART_CTL.RPOLC` bit. The ***IrDA Receiver Pulse Detection*** figure provides examples of each polarity type.

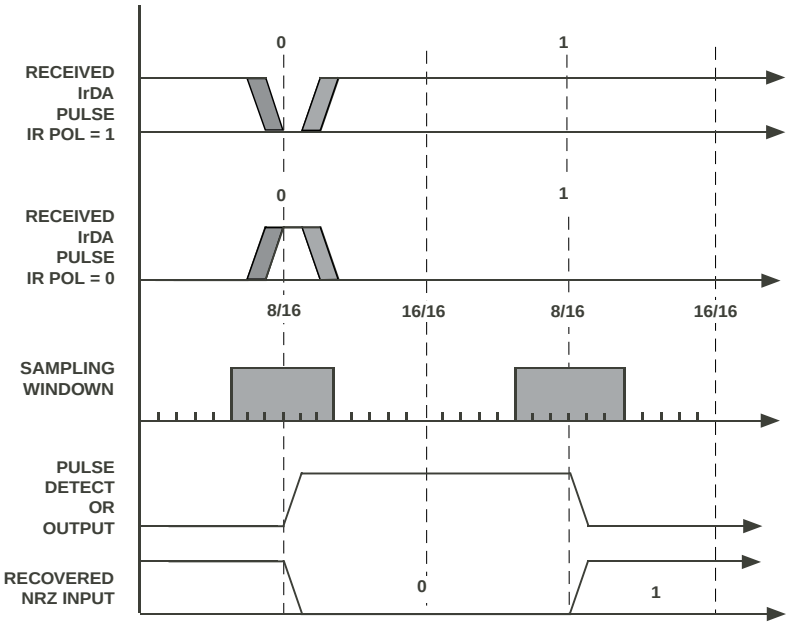

Figure 19-8: IrDA Receiver Pulse Detection

MDB Transmit Operation

In MDB mode, receive and transmit paths operate independently from each other, except for sharing bit rate and frame formats for both transfer directions.

Transmit operation is initiated by writing the `UART_THR` or `UART_TAIP` registers. A write to the `UART_THR` register transmits the written word with the appending address bit set low. A write to the `UART_TAIP` register transmits the written word with the appended address bit set high. The data is moved into the `UART_TSR` register, where it is shifted out at the bit rate programmed by the `UART_CLK` register, with start, stop, address, and parity bits appended, as required.

If DMA is enabled, the DMA engine always writes the data into the `UART_THR` register, and the written word is transmitted with the appending address bit set low.

The polarity of transmit data is selectable, using the `UART_CTL.TPOLC` bit.

MDB Receive Operation

Receive operations use the same data format as the transmit configuration, except that the number of stop bits is always assumed to be 1. After detection of the start bit, the received word is shifted into the `UART_RSR` register at the programmed bit.

Normally, the receiver samples every incoming bit at exactly the 7th, 8th and 9th sample clock. If, however, the `UART_CLK.EDBO` bit is set, the receiver samples the bits roughly at 7/16th, 8/16th, and 9/16th of their period. This configuration achieves better bit rate granularity and accuracy needed at high operation speeds. Hardware design must ensure that the incoming signal is stable between 6/16th and 10/16th of the nominal bit period.

After the appropriate number of bits (including address, parity, and stop bits) is received, the UART_RSR register is transferred to the receive FIFO and accessible through the UART_RBR register.

The polarity of receive data is selectable, using the UART_CTL.RPOLC bit.

DMA Mode

In DMA mode, separate receive and transmit DMA channels move data between the UART and memory. The software does not have to move data; it just has to set up the appropriate transfers either through the descriptor mechanism or through autobuffer mode.

DMA channels provide a 4-deep FIFO, resulting in total buffer capabilities of 6 words at the transmit side and 9 words at the receive side. In DMA mode, the bus activity and arbitration mechanism determine the latency. The processor loading and interrupt priorities do not determine the latency.

To enable UART DMA, first set up the system DMA control registers. Then, enable the UART_IMSK.ERBFI or UART_IMSK.ETBEI interrupts. This sequence is necessary because these interrupt request lines double as DMA request lines. With DMA enabled, once these requests are received, the DMA control unit generates a direct memory access. If DMA is not enabled, the UART interrupt is passed on to the system interrupt handling unit.

NOTE: The status interrupt for the UART goes directly to the system event controller (SEC), bypassing the DMA unit completely.

For transmit DMA, programs must set the DMA_CFG.SYNC bit. With this bit set, interrupt generation is delayed until the entire DMA FIFO is drained to the UART module. The UART transmit DMA interrupt service routine can disable the DMA or to clear the UART_IMSK.ETBEI control bit only when the DMA_CFG.SYNC bit is set. Otherwise, up to four data bytes can be lost.

When the UART_IMSK.ETBEI bit is set, an initial transmit DMA request is issued immediately. The program then clears the UART_IMSK.ETBEI bit through the DMA service routine.

In DMA transmit mode, the UART_IMSK.ETBEI bit enables the peripheral request to the DMA FIFO. The DMA_CFG.EN bit enables the strobe on the memory side. If the DMA count is less than the DMA FIFO depth, which is 4, then the DMA interrupt can be requested before the UART_IMSK.ETBEI bit is set. If this behavior is unwanted, set the DMA_CFG.SYNC bit.

Regardless of the DMA_CFG.SYNC setting, the DMA stream has not left the UART transmitter completely at the time the interrupt is generated. Transmission can abort in the middle of the stream, causing data loss, when the UART clock was disabled without extra synchronization with the UART_STAT.TEMT bit.

The UART provides functionality to avoid resource-consuming polling of the UART_STAT.TEMT bit. The UART_IMSK_SET.EDTPTI bit enables the UART_STAT.TEMT bit to trigger a DMA interrupt. To delay the DMA completion interrupt until the last data word of a STOP DMA has left the UART, keep the DMA_CFG.DI_EN bit cleared and set the UART_IMSK_SET.EDTPTI bit instead. Then, the normal DMA completion interrupt is suppressed. Later, the UART_STAT.TEMT event triggers a DMA interrupt after the last word of the DMA has left the UART transmit buffers. If DI_EN and UART_IMSK.EDTPTI are set, when finishing STOP mode, the DMA requests two interrupts.

The DMA of the UART module supports 8-bit and 16-bit operation, but not 32-bit operation. It does not support sign-extension.

Mixing DMA and Core Modes

Switching from DMA mode to core operation dynamically requires some consideration, especially for transmit operations. By default, the interrupt timing of the DMA is synchronized with the memory side of the DMA FIFOs. Normally, the transmit DMA completion interrupt is generated after the last byte is copied from the memory into the DMA FIFO. The transmit DMA interrupt service routine is not yet permitted to disable the DMA_CFG.EN bit. The interrupt is requested when the DMA_STAT.IRQDONE bit is set. The DMA_STAT.RUN bit, however, remains set until the data has completely left the transmit DMA FIFO.

When planning to switch from a DMA to core mode, set the DMA_CFG.SYNC bit in the word of the last descriptor or work unit before handing over control. Then, after the interrupt occurs, software can write new data into the UART_THR register as soon as the UART_STAT.THRE bit permits. If the DMA_CFG.SYNC bit cannot be set, software can poll the DMA_STAT.RUN bit instead. Alternatively, using the UART_IMSK.EDTPTI bit can avoid expensive status bit polling.

When switching from core to DMA operation, ensure that the first DMA request is issued properly. If the DMA is enabled while the UART is still transmitting, no precaution is required. If, however, the DMA is enabled after the UART_STAT.TEMT bit is high, pulse the UART_IMSK.ETBEI bit to initiate DMA transmission.

Setting Up Hardware Flow Control

Use the following steps to set up UART hardware flow control.

1. Configure automatic or manual hardware flow control for the receiver through the UART_CTL.ARTS bit, or the transmitter through the UART_CTL.ACTS bit.
2. Configure $\overline{\text{UART\_CTS}}$ and $\overline{\text{UART\_RTS}}$ polarity through the UART_CTL.FCPOL bit.

On reset, when the UART is not yet enabled and the port multiplexing has not been programmed, the $\overline{\text{UART\_RTS}}$ pin is not driven. Some applications require a resistor pull the UART_RTS signal to either state during reset.

UART Event Control

Status flags in the UART_STAT register are available to signal data reception, parity, and error conditions, if necessary.

DMA and Interrupt Multiplexing

See the *Direct Memory Access (DMA)* chapter on for information on DMA multiplexing. Several interrupts and DMA channels in thr UART can be multiplexed.

Interrupt Masks

Each UART features a set of interrupt mask registers: UART_IMSK, UART_IMSK_SET, and UART_IMSK_CLR. The UART_IMSK register supports read/write operations. Writing ones to the UART_IMSK_SET register enables interrupts, writing ones to the UART_IMSK_CLR register disables them. Reads from either register return the enabled bits. This way, different interrupt service routines can control transmit, receive, and status interrupts independently and easily.

The UART module uses the UART_IMSK registers to enable requests for system handling of empty or full states of data registers. Unless polling is used as a means of action, the UART_IMSK.ERBFI and UART_IMSK.ETBEI bits in this register are normally set.

Each UART module has three interrupt outputs. It uses one for transmission, one for reception, and one for reporting status events. The UART module routes transmit and receive requests through the DMA controller. The status request goes directly to the system event controller (SEC).

If the associated DMA channel is enabled, the request functions as a DMA request. If the DMA channel is disabled, it simply forwards the request to the SEC. A DMA channel must be associated with the UART module to enable transmit and receive interrupts. Otherwise, transmit and receive requests cannot be forwarded.

NOTE: To operate in interrupt mode without using DMA channels, set the UART_IMSK.ELSI bit. This configuration redirects receive and transmit requests to the status interrupt output. The status interrupt goes directly to the SEC without going through the DMA controller

Interrupt Servicing

Interrupt service routines (ISRs) perform UART writes and reads. Separate interrupt lines are provided for transmit, receive, and status. The UART_IMSK register group enables the independent interrupts individually. To enable UART transmit interrupts, set the UART_CTL.EN bit.

The ISRs evaluate the status bits in the UART_STAT register to determine the signaling interrupt source. The system event controller for the processor assigns and unmasks interrupts. The ISRs must clear the interrupt latches explicitly. To reduce interrupt frequency on the receive side in core mode, use the UART_IMSK.ERFCI status interrupt as an alternative to the regular UART_IMSK.ERBFI receive interrupt. Hardware must ensure that at least two (if UART_CTL.RFIT=0) or four (if UART_CTL.RFIT=1) words are available in the receive buffer by the time the interrupt is requested.

Transmit Interrupts

The UART module uses the UART_IMSK_SET.ETBEI bit to enable transmit interrupts.

The UART_THR and UART_TAIP registers are the same physical register, and both affect the signaling of the UART_STAT.TEMT, UART_STAT.TFI, and UART_STAT.THRE bits similarly.

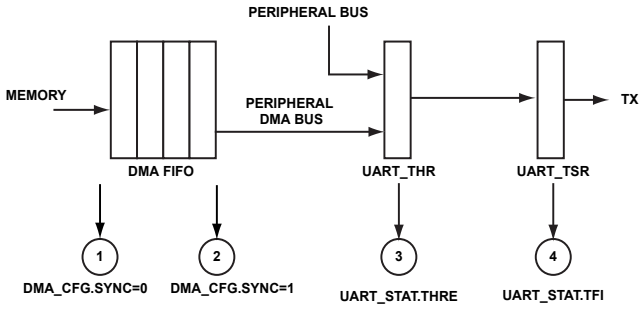

Figure 19-9: Transmit Interrupts

The UART module asserts the transmit request along with the `UART_STAT.THRE` bit, indicating that the transmit buffer is ready for new data. The `UART_STAT.THRE` bit resets to 1. When the `UART_IMSK_SET.ETBEI` bit is set, the UART module immediately issues an interrupt or DMA request. This way, no special handling of the first character is required when transmission of a string is initiated. Set the `UART_IMSK_SET.ETBEI` bit and let the interrupt service routine load the first character from memory and write it to the `UART_THR` register in the normal manner. ISRs can clear the `UART_IMSK.ETBEI` bit through the `UART_IMSK_CLR` register when the string transmission has completed.

Hardware clears the `UART_STAT.THRE` bit when new data is written to the `UART_THR` register. These write operations also clear the transmit interrupt request. However, they also initiate further transmission. If continued transmission is undesirable, the UART module can alternatively clear the transmit request through the `UART_IMSK_CLR.ETBEI` bit register. Transfers of data from the `UART_THR` register to the `UART_TSR` register reset this status flag in the `UART_STAT` register.

ISRs can interrogate the `UART_STAT.TEMT` bit to discover any ongoing transmission. The sticky counterpart of the `UART_STAT.TEMT` bit, `UART_STAT.TFI`, indicates when the transmit buffer has drained and can trigger a status interrupt. When data is pending in either one of these registers, the `UART_STAT.TEMT` flag is low. As soon as all data has left the `UART_TSR` register, the `UART_STAT.TEMT` bit goes high again and indicates that all pending transmit operations (including stop bits) have finished. Then, it is safe to disable the `UART_CTL.EN` bit or to three-state off-chip line drivers. Then, the UART module can generate an interrupt either through the status interrupt channel when the `UART_IMSK.ETFI` bit is set, or through the DMA controller when enabled by the `UART_IMSK.EDTPTI` bit.

When enabled by the `UART_IMSK.ETBEI` bit, the `UART_STAT.THRE` flag requests data along the peripheral command lines to the DMA controller (referred to as *TXREQ*). This signal is routed through the DMA controller. If the associated DMA channel is enabled, the *TXREQ* signal functions as a DMA request, otherwise the DMA controller simply forwards it to the SEC. Alternatively the `UART_IMSK.ETXS` bit can redirect the transmit interrupts to the UART status interrupt.

With interrupts disabled, the UART module can poll the status flags to determine when data is ready to move. Because polling is processor intensive, it is not typically used in real-time signal processing environments. Since read operations from `UART_STAT` registers have no side effects, different software threads can interrogate these registers without mutual impacts.

Receive Interrupts

The UART module uses the UART_IMSK_SET.ERBFI bit to enable receive-interrupts. If set, the UART_STAT.DR flag requests an interrupt on the dedicated *RXREQ* output, indicating that new data is available in the UART_RBR register. This signal is routed through the DMA controller. If the associated DMA channel is enabled, the *RXREQ* signal functions as a DMA request; otherwise the DMA controller simply forwards it to the SEC. Alternatively, if no DMA channel is assigned to the UART, the UART_IMSK.ERXS bit can redirect the receive interrupts to the UART status interrupt. When software reads the UART_RBR register, hardware clears the UART_STAT.DR bit again, which, in turn, clears the receive interrupt request.

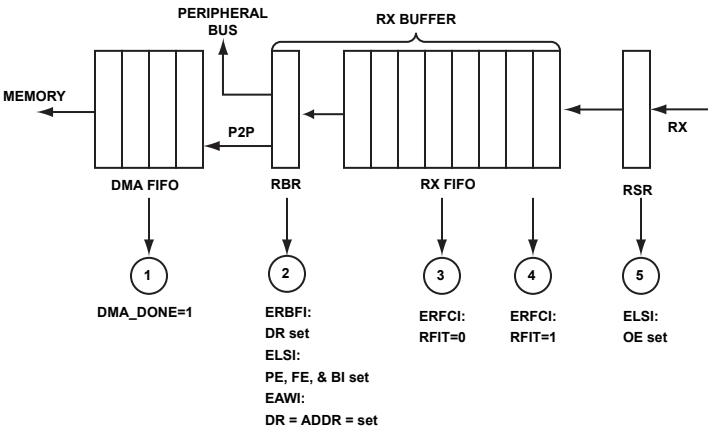

Figure 19-10: Receive Interrupts

Hardware updates the following:

- UART_STAT.DR bits
- UART_STAT.ADDR bits
- UART_STAT.ASTKY bits
- UART_STAT.PE bits
- UART_STAT.FE bits
- UART_STAT.BI bits
- UART_RBR register

The UART_STAT.OE bit is updated as soon as an overflow condition occurs (for example when a stop bit for a frame is received and the receive FIFO is full). When software does not read the UART_RBR register in time, the received data is protected from being overwritten by new data until software clears the UART_STAT.OE bit. Only the content of the UART_RSR register can be overwritten in the overrun case.

The UART module uses the UART_STAT.RFCS bit to monitor the state of the 8-deep receive FIFO. It uses the UART_CTL.RFIT bit to control the behavior of the buffer. If UART_CTL.RFIT is zero, the UART_STAT.RFCS bit is set when the receive buffer holds four or more words. If UART_CTL.RFIT is set, the UART_STAT.RFCS bit is set when the receive buffer holds seven or more words. Hardware clears the UART_STAT.RFCS bit when a core or

DMA reads the UART_RBR register and when the buffer is flushed below the level of four (UART_CTL.RFIT=0) or seven (UART_CTL.RFIT=1). If the associated interrupt bit UART_IMSK.ERFCI is enabled, a status interrupt is reported when the UART_STAT.RFCS bit is set.

If errors are detected during reception, an interrupt can be requested from the status interrupt output. This status interrupt request goes directly to the SEC. The bit enables status interrupt requests.

The controller detects the following error conditions, shown with their associated bits in the UART_STAT register.

- Overrun error (UART_STAT.OE bit)
- Parity error (UART_STAT.PE bit)
- Framing error or invalid stop bit (UART_STAT.FE bit)
- Break indicator (UART_STAT.BI bit)

Status Interrupts

The UART module uses status interrupt channels for the following purposes.

- Line status interrupts
- Flow control interrupts
- Receive FIFO threshold interrupts
- Transmission finished interrupt

The UART module uses the UART_IMSK.ELSI bit to enable the line status interrupts. If set, the status interrupt request is asserted with one of the UART_STAT.BI, UART_STAT.FE, UART_STAT.PE, or UART_STAT.OE receive errors bits. A W1C operation to the UART_STAT register clears the error bits. Once all error conditions are cleared, the interrupt request deasserts.

The UART module uses the UART_IMSK_SET.ERFCI bit to enable the receive FIFO count interrupt. If set, a status interrupt is generated when the UART_STAT.RFCS is active. The UART_STAT.RFCS bit indicates a receive buffer threshold level. If the UART_CTL.RFIT bit is cleared, software can safely read two words out of the UART_RBR register by the time the UART_STAT.RFCS interrupt occurs.

If the UART_CTL.RFIT bit is set, software can safely read four words. The interrupt and the UART_STAT.RFCS bit clear when the UART_RBR is read enough of times, so that the receive buffer drains below the threshold of two (UART_CTL.RFIT=0) or four (UART_CTL.RFIT=1). Because in DMA mode a status service routine may not be permitted to read UART_RBR, this interrupt is only recommended in core mode. In DMA mode, use this functionality for error recovery only.

The UART module uses the UART_IMSK_SET.EDSSI bit to enable the flow control interrupts. If active, a status interrupt is generated when the sticky UART_STAT.SCTS bit register is set, indicating that the UART_CTS input of the transmitter been reasserted. A W1C operation to the SCTS bit clears the interrupt request.

The UART module uses the UART_IMSK_SET.ETFI bit to enable the transmission finished interrupt. If active, a status interrupt request is asserted when the UART_STAT.TFI bit is set. The UART_STAT.TFI is the sticky version

of the UART_STAT.TEMT bit, indicating that a byte that started transmission has finished. A W1C operation to the UART_STAT.TFI bit clears the interrupt request.

Multi-Drop Bus Events

Several status bits and interrupt features in the UART_STAT and UART_IMSK registers facilitate efficient data handling in multi-drop bus mode. These features include the address (UART_STAT.ADDR) bit, address sticky (UART_STAT.ASTKY) bit and enable address word interrupt (UART_IMSK.EAWI). One of the key features of the multi-drop bus protocol is its address bit. The address bit signifies to the slaves that the master is transmitting an address word (all read it) or a data word (only the addressed slave reads its). The UART hardware provides an efficient method of handling the situation described with the use of UART_STAT.ASTKY bit.

NOTE: The UART module uses the UART_STAT.ASTKY bit in multi-drop bus mode to indicate when an address operation for a peripheral is occurring. The UART_STAT.ASTKY bit is a sticky version of the UART_STAT.ADDR bit. Hardware sets the bit whenever the UART_STAT.ADDR bit is set. Only software clears it with a W1C operation. With the ASTKY bit set, words are received irrespective of the mode bit or address bit setting. With the UART_STAT.ASTKY bit cleared, only address words (mode bit =1) are received and words with mode bit =0 are ignored in MDB mode. (Words with mode bit =0 are not moved from the UART_RSR to the receive FIFO). This bit does not affect reception in non-MDB modes.

UART Programming Model

The following sections provide basic procedures for configuring various UART operations.

Detecting Autobaud

Refer to Autobaud Detection for more information. The required steps are:

1. Ensure that the timer is disabled.
2. Configure the following bits: UART_CTL.MOD =00, UART_CTL.LOOP_EN =1, UART_CTL.WLS =11 (8-bit data), and UART_CTL.EN =1
3. Configure the following bits: TIMER_TMR[n]_CFG.TMODE =1101, TIMER_TMR[n]_CFG.OUTDIS =1, TIMER_TMR[n]_CFG.IRQMODE =10 and enable the timer.
4. Send test data through the host device and wait for the timer interrupt and disable the timer.

   The bit rate can be derived from the timer period register value according to the formula provided in the Autobaud Detection section.

Using Common Initialization Steps

When using the core or the DMA to execute transfers, the following steps are common to all UART modes.

1. All UART signals are multiplexed and compete with other functions at pin level. First, program the port registers according to the guidelines in the PORTs chapter.

2. Program the UART_CLK register. Refer to Bit Rate Generation.
3. Program the UART_CTL register and enable the UART clock.

Using Core Transfers

Write data into the UART_THR register, when the UART_STAT.THRE bit is set, to initiate a core transmit operation. If the UART_STAT.DR bit is set, received data can be read from the UART_RBR register.

Using DMA Transfers

1. Make sure that the UART_IMSK.ETBEI or the UART_IMSK.ERBFI bits are cleared before configuring the DMA.
2. Configure the dedicated DMA channel.
3. Set the UART_IMSK.ETBEI or UART_IMSK.ERBFI bits to start the transfer.

Using Interrupts

Each UART features three interrupt signal outputs.

1. Enable individual interrupts in the system event controller (SEC).
2. Register IRQ handlers.
3. Use the interrupts mask registers to enable specific IRQ events.

Setting Up Hardware Flow Control

1. Configure automatic or manual hardware flow control for the receiver through the UART_CTL.ARTS bit, or the transmitter through the UART_CTL.ACTS bit.
2. Configure $\overline{\text{UART\_CTS}}$ and $\overline{\text{UART\_RTS}}$ polarity through the UART_CTL.FCPOL bit.

ADSP-CM40x UART Register Descriptions

UART (UART) contains the following registers.

Table 19-6: ADSP-CM40x UART Register List

Name	Description
UART_CLK	Clock Rate Register
UART_CTL	Control Register
UART_IMSK	Interrupt Mask Register
UART_IMSK_CLR	Interrupt Mask Clear Register
UART_IMSK_SET	Interrupt Mask Set Register

Table 19-6: ADSP-CM40x UART Register List (Continued)

Name	Description
UART_RBR	Receive Buffer Register
UART_RSR	Receive Shift Register
UART_RXCNT	Receive Counter Register
UART_SCR	Scratch Register
UART_STAT	Status Register
UART_TAIP	Transmit Address/Insert Pulse Register
UART_THR	Transmit Hold Register
UART_TSR	Transmit Shift Register
UART_TXCNT	Transmit Counter Register

Clock Rate Register

The UART_CLK register divides the system clock (SYSCLK) down to the bit clock.

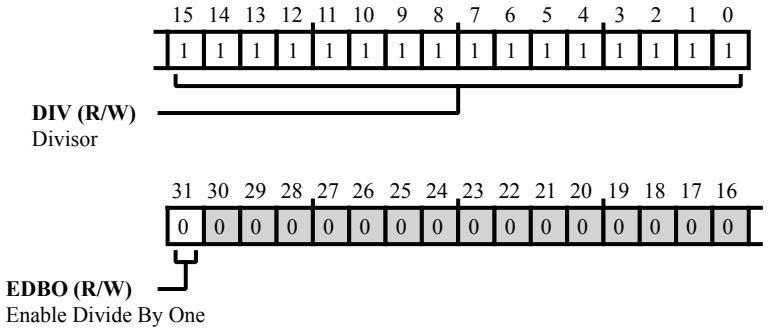

Figure 19-11: UART_CLK Register Diagram

Table 19-7: UART_CLK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	EDBO	Enable Divide By One. The UART_CLK.EDBO bit enables bypassing of the divide-by-16 prescaler in bit clock generation. This improves bit rate granularity, especially at high bit rates. Do not set this bit in IrDA mode.	
		0	Bit clock prescaler = 16
		1	Bit clock prescaler = 1
15:0 (R/W)	DIV	Divisor. The UART_CLK.DIV provides the divisor for the UART's clock bit rate calculation. The bit rate is defined by: Bit Rate = SYSCLK / ($16^{(1-EDBo)}$ x UART_CLK.DIV)	

Control Register

The UART_CTL register provides enable/disable control for internal UART and for the IrDA mode of operation. This register also provides UART line control, permitting selection of the format of received and transmitted character frames. Modem feature control also is available from this register, including partial modem functionality to allow for hardware flow control and loopback mode.

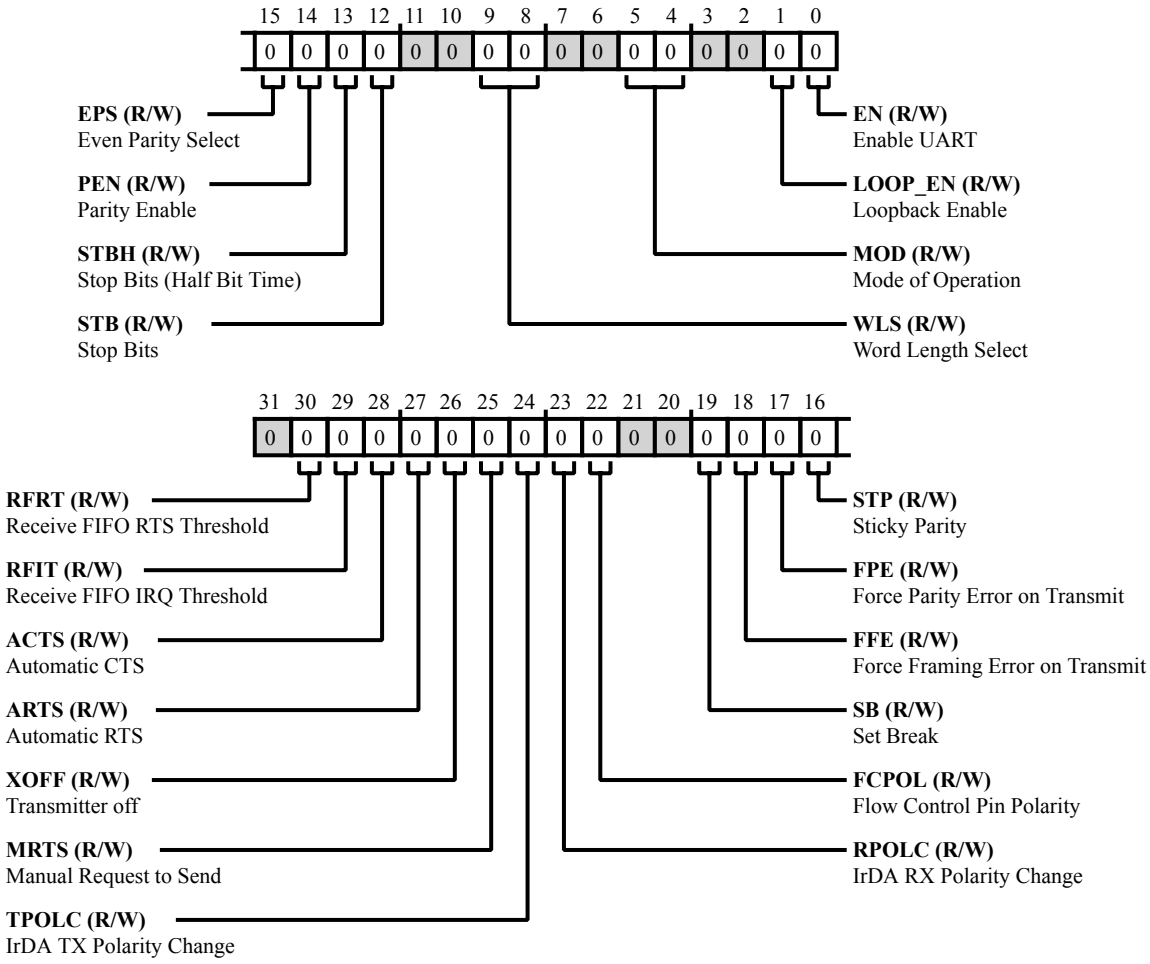

Figure 19-12: UART_CTL Register Diagram

Table 19-8: UART_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	RFRT	Receive FIFO RTS Threshold. The UART_CTL.RFRT bit controls UART_RTS pin assertion and de-assertion timing. This bit is ignored if UART_CTL.ARTS is cleared. If set, the UART_RTS pin is de-asserted when the receive buffer already holds seven words and an eighth start bit is detected. It is re-asserted when the FIFO contains seven words or less. If cleared, de-assert UART_RTS pin when the RX buffer already holds four words and a fifth start bit is detected. The UART_RTS pin is re-asserted when the RX buffer contains no more than 4 words.	
		0	De-assert RTS if RX FIFO word count > 4; assert if <= 4
		1	De-assert RTS if RX FIFO word count > 7; assert if <= 7
29 (R/W)	RFIT	Receive FIFO IRQ Threshold. The UART_CTL.RFIT bit controls the timing of the UART_STAT.RFCS bit. If UART_CTL.RFIT is cleared, the receive threshold is two. If UART_CTL.RFIT is set, the threshold is four words in the receive buffer.	
		0	Set RFCS=1 if RX FIFO count >= 4
		1	Set RFCS=1 if RX FIFO count >= 7
28 (R/W)	ACTS	Automatic CTS. The UART_CTL.ACTS bit must be set to enable the UART_CTS input pin for UART_TX handshaking. If enabled, the UART_STAT.CTS bit holds the value (if UART_CTL.FCPOL is set) or complement value (if UART_CTL.FCPOL is cleared) of the UART_CTS input pin. The UART_STAT.CTS bit can be used to determine whether the external device is ready to receive data (if UART_STAT.CTS set) or whether it is busy (if UART_STAT.CTS cleared). If UART_CTL.ACTS is cleared, the UART_TX handshaking protocol is disabled, and the UART_TX line transmits data whenever there is data to send, regardless of the value of UART_CTS. Software can pause ongoing transmission by setting the UART_CTL.XOFF bit.	
		0	Disable TX handshaking protocol
		1	Enable TX handshaking protocol

Table 19-8: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
27 (R/W)	ARTS	Automatic RTS. The UART_CTL.ARTS bit must be set to enable the UART_RTS input pin for UART_TX handshaking.If set, hardware guarantees minimal UART_RTS pin de-assertion pulse width of at least the number of data bits defined by the UART_CTL.WLS bit field. If cleared, the UART_RTS pin is not generated automatically by hardware. The UART_RTS pin can still be manually controlled by the UART_CTL.MRTS bit, and software is responsible for UART_RTS pulse width control (if needed).	
		0	Disable RX handshaking protocol.
		1	Enable RX handshaking protocol.
26 (R/W)	XOFF	Transmitter off. The UART_CTL.XOFF bit (if set) turns off transmission (XOFF) by preventing the content of THR from being continued to TSR. When set, this bit turns on transmission (XON). The state of the UART_CTL.XOFF bit is ignored if the UART_CTL.ACTS bit is set.	
		0	Transmission ON, if ACTS=0
		1	Transmission OFF, if ACTS=0
25 (R/W)	MRTS	Manual Request to Send. The UART_CTL.MRTS bit controls the state of the UART_RTS output pin when the UART_CTL.ARTS bit is cleared. When UART_CTL.MRTS is cleared, the UART de-asserts the UART_RTS pin, signaling to the external device that the UART is not ready to receive. When UART_CTL.MRTS is set, the UART asserts the UART_RTS pin, signaling to the external device that the UART is ready to receive.	
		0	De-assert RTS pin when ARTS=0
		1	Assert RTS pin when ARTS=0
24 (R/W)	TPOLC	IrDA TX Polarity Change. The UART_CTL.TPOLC bit selects the active low/high polarity for IrDA communications. This bit only is effective in IrDA mode. If set, in IrDA mode, the UART_TX pin idles high. In UART or MDB mode, it is inverted-NRZ. If cleared, in IrDA mode, the UART_TX pin idles low. In UART or MDB mode, it is NRZ.	
		0	Active-low TX polarity setting
		1	Active-high TX polarity setting

Table 19-8: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
23 (R/W)	RPOLC	IrDA RX Polarity Change. The UART_CTL.RPOLC bit selects the active low/high polarity for IrDA communications. This bit only is effective in IrDA mode. If set, in IrDA mode, the UART_RX pin idles high. In UART or MDB mode, it is inverted-NRZ. If cleared, in IrDA mode, the UART_RX pin idles low. In UART or MDB mode, it is NRZ.	
		0	Active-low RX polarity setting
		1	Active-high RX polarity setting
22 (R/W)	FCPOL	Flow Control Pin Polarity. The UART_CTL.FCPOL select the polarities of the UART_CTS and UART_RTS pins. When UART_CTL.FCPOL is cleared, the UART_RTS and UART_CTS pins are active low, and UART is halted when the UART_RTS and UART_CTS pin state is high. When UART_CTL.FCPOL is set, the UART_RTS and UART_CTS pins are active high, and UART is halted when the UART_RTS and UART_CTS pin state is low.	
		0	Active low CTS/RTS
		1	Active high CTS/RTS
19 (R/W)	SB	Set Break. If set, the UART_CTL.SB bit forces the UART_TX pin to low asynchronously, regardless of whether or not data is currently transmitted. This bit functions even when the UART clock is disabled. Because the UART_TX pin normally drives high, it can be used as a flag output pin, if the UART is not used. (For example, if UART_CTL.TPOLC is cleared, drive UART_TX pin low; or if UART_CTL.TPOLC is set, drive UART_TX pin high.)	
		0	No force
		1	Force TX pin to 0
18 (R/W)	FFE	Force Framing Error on Transmit. The UART_CTL.FFE bit is intended for test purposes. This bit is useful for debugging software, especially in loopback mode.	
		0	Normal operation
		1	Force error
17 (R/W)	FPE	Force Parity Error on Transmit. The UART_CTL.FPE bit is intended for test purposes. This bit is useful for debugging software, especially in loopback mode.	
		0	Normal operation
		1	Force parity error

Table 19-8: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	STP	Sticky Parity. The UART_CTL.STP bit controls whether the parity is generated by hardware based on the data bits or whether it is set to a fixed value. If this bit is cleared, the hardware calculates the parity bit value based on the data bits. Then, the EPS bit determines whether odd or even parity mode is chosen. If this bit is set, odd parity is used. That means that the total count of logical-1 data bits including the parity bit must be an odd value. Even parity is chosen by UART_CTL.STP cleared and UART_CTL.EPS set. Then, the count of logical-1 bits must be a even value. If the UART_CTL.STP bit is set, hardware parity calculation is disabled. In this case, the sent and received parity equals the inverted UART_CTL.EPS bit.	
		0	No Forced Parity
		1	Force (Stick) Parity to Defined Value (if PEN=1)
15 (R/W)	EPS	Even Parity Select.	
		0	Odd parity
		1	Even parity
14 (R/W)	PEN	Parity Enable. The UART_CTL.PEN enables parity transmission and parity check. The UART_CTL.PEN bit inserts one additional bit between the most significant data bit and the first stop bit. The polarity of this so-called parity bit depends on data and the UART_CTL.STP and UART_CTL.EPS control bits. Both transmitter and receiver calculate the parity value. The receiver compares the received parity bit with the expected value and issues a parity error if they dont match. If UART_CTL.PEN is cleared, the UART_CTL.STP and the UART_CTL.EPS bits are ignored.	
		0	Disable
		1	Enable parity transmit and check
13 (R/W)	STBH	Stop Bits (Half Bit Time).	
		0	0 half-bit-time stop bit
		1	1 half-bit-time stop bit
12 (R/W)	STB	Stop Bits. The UART_CTL.STB bit controls how many stop bits are appended to transmitted data.	
		0	1 stop bit
		1	2 stop bits

Table 19-8: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/W)	WLS	Word Length Select. The UART_CTL.WLS field determines whether the transmitted and received UART word consists of 5, 6, 7, or 8 data bits.	
		0	5-bit Word
		1	6-bit Word
		2	7-bit Word
		3	8-bit Word
5:4 (R/W)	MOD	Mode of Operation. The UART_CTL.MOD selects the UART operation mode (UMOD).	
		0	UART Mode
		1	MDB Mode
		2	IrDA SIR Mode
1 (R/W)	LOOP_EN	Loopback Enable. The UART_CTL.LOOP_EN enables UART loopback mode. When set, this bit disconnects the receivers input from the UART_RX pin, and internally redirects the transmit output to the receiver. The UART_TX pin remains active and continues to transmit data externally as well. Loopback mode also forces the UART_RTS pin to its de-assertive state, disconnects the UART_CTS bit from the UART_CTS input pin, and directly connects the UART_CTL.MRTS bit to the UART_STAT.CTS bit. In loopback mode, setting the UART_CTL.MRTS bit sets the UART_STAT.CTS bit and enables the UARTs transmitter. Clearing to the UART_CTL.MRTS bit clears the UART_STAT.CTS bit and disables the UARTs transmitter.	
		0	Disable
		1	Enable
0 (R/W)	EN	Enable UART. The UART_CTL.EN enables the UART clocks. This bit also resets the state machine and control registers when cleared. Using this bit to disable the UART -- when not used -- reduces power consumption.	
		0	Disable
		1	Enable

Interrupt Mask Register

The UART_IMSK indicates interrupt mask status (unmasked if set, masked if cleared) of UART status interrupts. This register is not a data register. Instead it is controlled by the UART_IMSK_SET and UART_IMSK_CLR register pair. Writing ones to UART_IMSK_SET enables (unmasks) interrupts, and writing ones to UART_IMSK_CLR disables (masks) them. Reads from either register return the enabled bits.

The UART_IMSK register is used to enable requests for system handling of empty or full states of UART data registers. Unless polling is used as a means of action, the UART_IMSK.ERBFI and/or UART_IMSK.ETBEI bits are normally set. Setting this register without enabling system DMA causes the UART to notify the processor of data inventory state by means of interrupts. For proper operation in this mode, system interrupts must be enabled, and appropriate interrupt handling routines must be present.

Each UART features three separate interrupt channels to handle data transmit, data receive, and line status events independently, regardless whether DMA is enabled or not. If no DMA channels are assigned to the UART, set the UART_IMSK.ELSI bit to reroute transmit and receive interrupts to the status interrupt output.

With system DMA enabled, the UART uses DMA to transfer data to or from the processor. Dedicated DMA channels are available to receive and transmit operation. Line error handling can be configured completely independently from the receive/transmit setup.

The UART's DMA is enabled by first setting up the system DMA control registers and then enabling the UART_IMSK.ERBFI and/or UART_IMSK.ETBEI interrupts. This is because the interrupt request lines double as DMA request lines. Depending on whether DMA is enabled or not, upon receiving these requests, the DMA control unit either generates a direct memory access or passes the UART interrupt on to the system interrupt handling unit. However, UART's error interrupt goes directly to the system interrupt handling unit, bypassing the DMA unit completely.

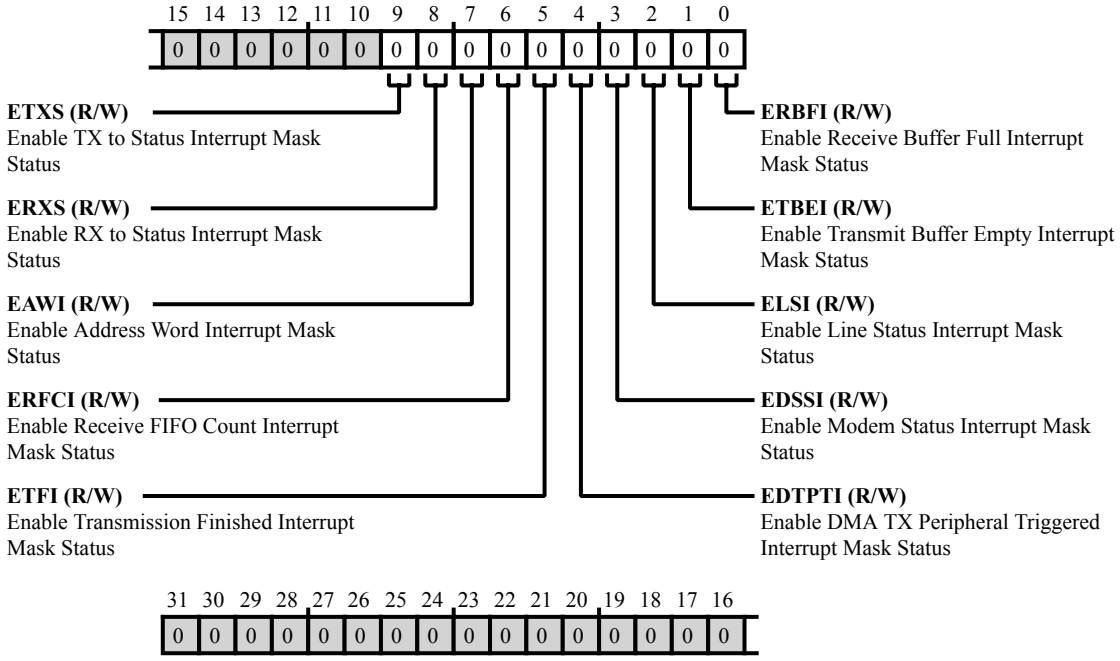

Figure 19-13: UART_IMSK Register Diagram

Table 19-9: UART_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	ETXS	Enable TX to Status Interrupt Mask Status. If set (interrupt unmasked), the `UART_IMSK.ETXS` bit indicates re-direction of the TX interrupts to status interrupt output. If cleared, TX interrupts are routed to normal interrupt outputs.	
		0	Interrupt is masked
		1	Interrupt is unmasked
8 (R/W)	ERXS	Enable RX to Status Interrupt Mask Status. If set (interrupt unmasked), the `UART_IMSK.ERXS` bit indicates re-direction of RX interrupts to status interrupt output. If cleared, RX interrupts are routed to normal interrupt outputs.	
		0	Interrupt is masked
		1	Interrupt is unmasked

Table 19-9: UART_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	EAWI	Enable Address Word Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.EAWI bit indicates generation of a status interrupt when an Address word in MDB-mode is present in the UART_RBR. A received word is an address word if the UART_STAT.ADDR bit is set.	
		0	Interrupt is masked
		1	Interrupt is unmasked
6 (R/W)	ERFCI	Enable Receive FIFO Count Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.ERFCI bit indicates enabling of the receive buffer threshold interrupt if signaled by the UART_STAT.RFCS bit. Read the UART_RBR register sufficient times to clear the interrupt request.	
		0	Interrupt is masked
		1	Interrupt is unmasked
5 (R/W)	ETFI	Enable Transmission Finished Interrupt Mask Status. If set (interrupt unmasked) the UART_IMSK.ETFI bit indicates enabling of interrupt generation on the status interrupt channel when the transmit buffer register, the transmit address register, and the transmit shift register are all empty as indicated by the UART_STAT.TFI. The UART_IMSK.ETFI interrupt can be used to avoid expensive polling of the UART_STAT.TEMT bit, when the UART clock or line drivers should be disabled after transmission has completed. W1C the UART_STAT.TFI bit to clear the interrupt request. In DMA operation, the UART_IMSK.ETFI bits functionality might be preferred.	
		0	Interrupt is masked
		1	Interrupt is unmasked
4 (R/W)	EDTPTI	Enable DMA TX Peripheral Triggered Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.EDTPTI bit indicates enabling of the DMA completion interrupt to be delayed until the data has left the UART completely. This bit is required for DMA transmit operation only. If set, the UART can generate a DMA interrupt by the time the UART_STAT.TEMT bit goes high after the last DMA data word is transmitted. When UART_IMSK.EDTPTI is set, usually the DMA_CFG.INT field is cleared to 00 in a STOP mode DMA. This set up suppresses the normal completion interrupt, and the UART_STAT.TEMT event is signaled through the DMA controller and triggers the DMA interrupt. If both (DMA_CFG.INT not 00 and UART_IMSK.EDTPTI set), two interrupts are requested at the end of a STOP mode DMA.	
		0	Interrupt is masked
		1	Interrupt is unmasked

Table 19-9: UART_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	EDSSI	Enable Modem Status Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.EDSSI bit indicates enabling of a modem status interrupt on the same status interrupt channel when the UART_STAT.SCTS bit is set. This indicates UART_CTS pin re-assertion. Write-1-to-clear (W1C) the UART_STAT.SCTS bit to clear the interrupt request.	
		0	Interrupt is masked
		1	Interrupt is unmasked
2 (R/W)	ELSI	Enable Line Status Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.ELSI bit indicates that redirection of TX and RX interrupt requests to the status interrupt output of the UART by OR'ing them with the UART_STAT.OE, UART_STAT.PE, UART_STAT.FE, and UART_STAT.BI interrupt requests. Set this bit when no DMA channel is associated with the UART. Enabling UART_IMSK.ELSI disables the RX/TX interrupt channels and negates the UART_IMSK.EDTPTI bit.	
		0	Interrupt is masked
		1	Interrupt is unmasked
1 (R/W)	ETBEI	Enable Transmit Buffer Empty Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.ETBEI bit indicates generation of a TX interrupt if the UART_STAT.THRE bit is set.	
		0	Interrupt is masked
		1	Interrupt is unmasked
0 (R/W)	ERBFI	Enable Receive Buffer Full Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.ERBFI indicates generation of an RX interrupt if the UART_STAT.DR bit is set.	
		0	Interrupt is masked
		1	Interrupt is unmasked

Interrupt Mask Clear Register

The UART_IMSK indicates interrupt mask status (unmasked if set, masked if cleared) of UART status interrupts. This register is not a data register. Instead it is controlled by the UART_IMSK_SET and UART_IMSK_CLR register pair. Writing ones to UART_IMSK_SET enables (unmasks) interrupts, and writing ones to UART_IMSK_CLR disables (masks) them. Reads from either register return the enabled bits. For more information, see the UART_IMSK register description.

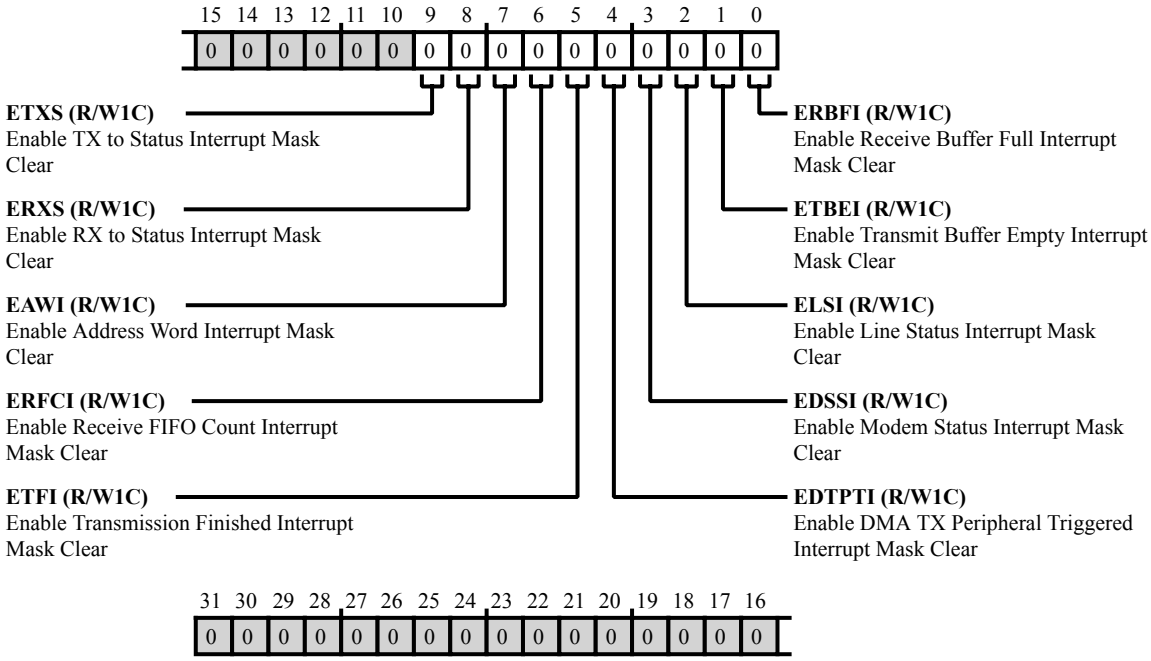

Figure 19-14: UART_IMSK_CLR Register Diagram

Table 19-10: UART_IMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W1C)	ETXS	Enable TX to Status Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
8 (R/W1C)	ERXS	Enable RX to Status Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
7 (R/W1C)	EAWI	Enable Address Word Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt

Table 19-10: UART_IMSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1C)	ERFCI	Enable Receive FIFO Count Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
5 (R/W1C)	ETFI	Enable Transmission Finished Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
4 (R/W1C)	EDTPTI	Enable DMA TX Peripheral Triggered Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
3 (R/W1C)	EDSSI	Enable Modem Status Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
2 (R/W1C)	ELSI	Enable Line Status Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
1 (R/W1C)	ETBEI	Enable Transmit Buffer Empty Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
0 (R/W1C)	ERBFI	Enable Receive Buffer Full Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt

Interrupt Mask Set Register

The UART_IMSK indicates interrupt mask status (unmasked if set, masked if cleared) of UART status interrupts. This register is not a data register. Instead it is controlled by the UART_IMSK_SET and UART_IMSK_CLR register pair. Writing ones to UART_IMSK_SET enables (unmasks) interrupts, and writing ones to UART_IMSK_CLR disables (masks) them. Reads from either register return the enabled bits. For more information, see the UART_IMSK register description.

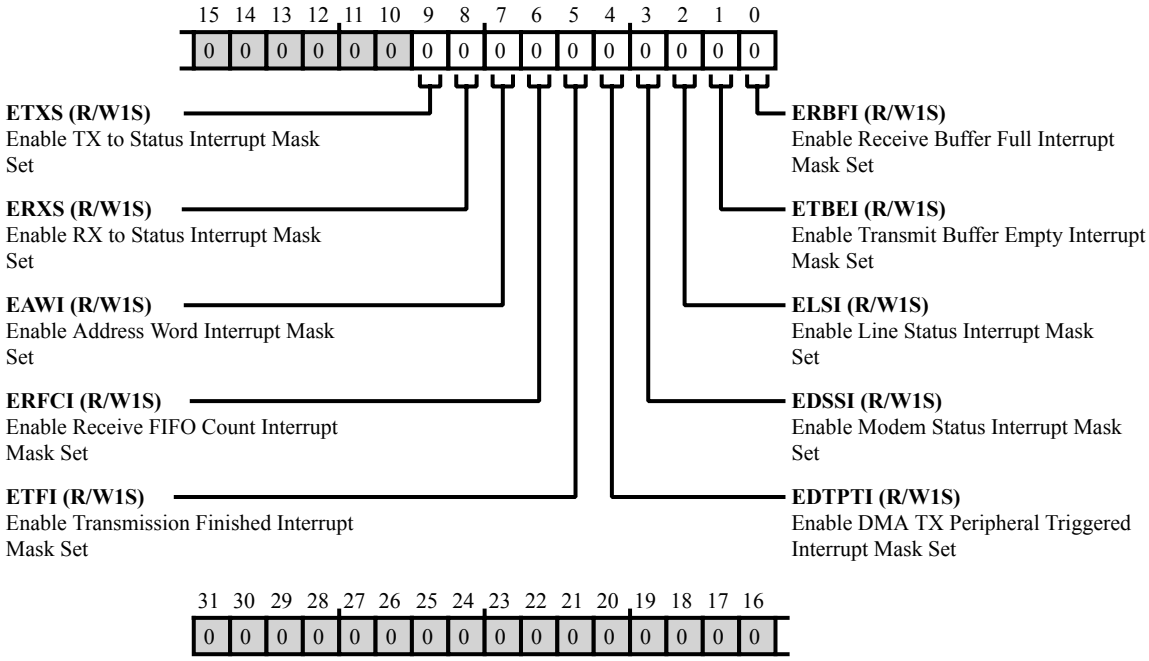

Figure 19-15: UART_IMSK_SET Register Diagram

Table 19-11: UART_IMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W1S)	ETXS	Enable TX to Status Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
8 (R/W1S)	ERXS	Enable RX to Status Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
7 (R/W1S)	EAWI	Enable Address Word Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt

Table 19-11: UART_IMSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1S)	ERFCI	Enable Receive FIFO Count Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
5 (R/W1S)	ETFI	Enable Transmission Finished Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
4 (R/W1S)	EDTPTI	Enable DMA TX Peripheral Triggered Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
3 (R/W1S)	EDSSI	Enable Modem Status Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
2 (R/W1S)	ELSI	Enable Line Status Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
1 (R/W1S)	ETBEI	Enable Transmit Buffer Empty Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
0 (R/W1S)	ERBFI	Enable Receive Buffer Full Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt

Receive Buffer Register

The read-only UART_RBR register is the UART's receive buffer. It is updated when there is pending data in the receive FIFO. Newly available data is signaled by the UART_STAT.DR bit.

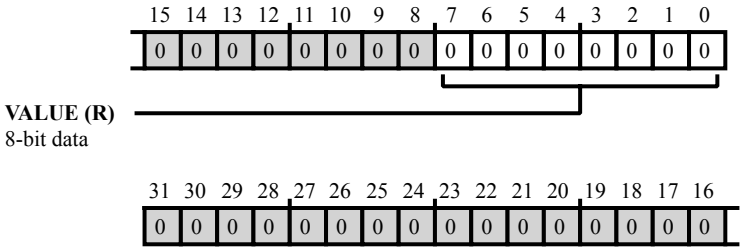

Figure 19-16: UART_RBR Register Diagram

Table 19-12: UART_RBR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	VALUE	8-bit data.

Receive Shift Register

The read only UART_RSR register which returns the content of the UART's receive shift register.

The frame data is moved into this shift register after polarity inversion, if any (including the native polarity inversion in the IrDA case).

In the case of the longest frame (MDB, with parity mode, and 8 bit data word-length), the start bit may be shifted out and not available for reading at the end of the frame reception. This register is NOT reset at the start of frame. If read, in the middle of a frame reception, data corresponding the previous frame may not have entirely shifted out (for example, the read data that have been read may NOT correspond entirely to the frame being received).

Because the UART is receiving only 1 stop bit, the UART_RSR contains only 1 stop bit even if more than one stop bit is present in the actual transfer. This register may be considered as storing the 10 most recently received bits (taking into consideration the stop bit receive limitation above).

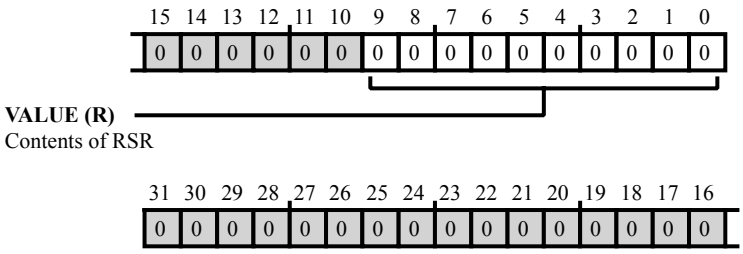

Figure 19-17: UART_RSR Register Diagram

Table 19-13: UART_RSR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/NW)	VALUE	Contents of RSR.

Receive Counter Register

The UART_RXCNT register returns the content of 16-bit counter in the UART receiver. This count is used for baud rate clock generation (the lower [15:0] is the count data).

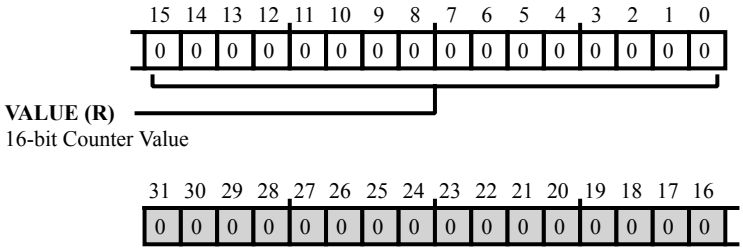

Figure 19-18: UART_RXCNT Register Diagram

Table 19-14: UART_RXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	16-bit Counter Value.

Scratch Register

The UART_SCR registers contain 8-bit scratch pad data. These registers are used for general purpose data storage and do not control the UART hardware in any way.

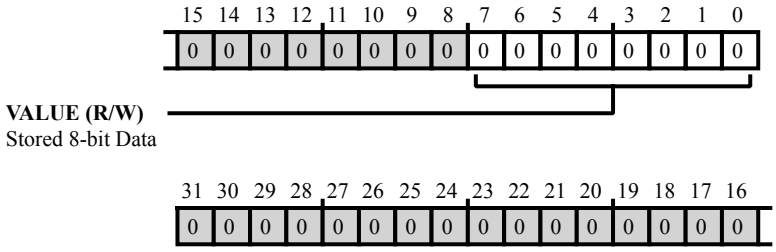

Figure 19-19: UART_SCR Register Diagram

Table 19-15: UART_SCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Stored 8-bit Data.

Status Register

The UART_STAT register contains the UART line status and UART modem status, as indicated by the current states of the UART's UART_CTS pin and internal receive buffers. Writes to this register can perform write-one-to-clear (W1C) operations on most status bits. Reading this register has no side effects.

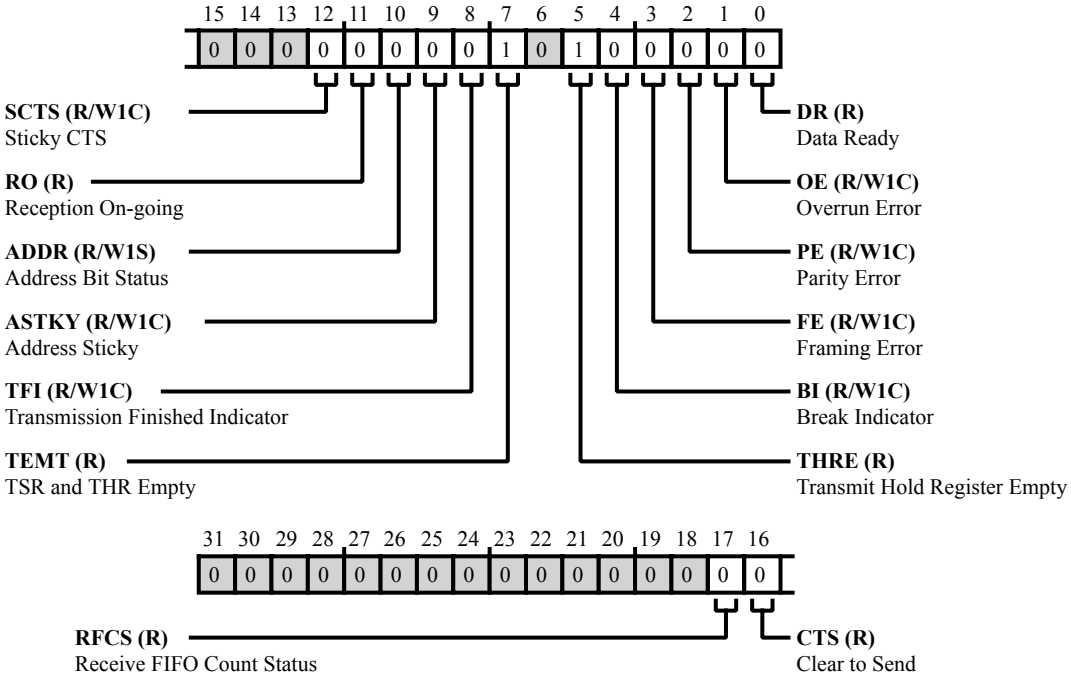

Figure 19-20: UART_STAT Register Diagram

Table 19-16: UART_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/NW)	RFCS	Receive FIFO Count Status. The UART_STAT.RFCS bit is set when the receive buffer holds more or equal entries than a certain threshold. The threshold is controlled by the UART_CTL.RFIT bit. If UART_CTL.RFIT is cleared, the threshold is four entries. If UART_CTL.RFIT is set, the threshold is seven entries. The UART_STAT.RFCS bit is cleared when the UART_RBR register is read sufficient times until the buffer is drained below the threshold. The UART_STAT.RFCS bit can trigger a status interrupt if enabled by the UART_IMSK_SET.ERFCI bit.	
		0	RX FIFO has less than 4 (7) entries when RFIT=0 (1)
		1	RX FIFO has at least 4 (7) entries when RFIT=0 (1)

Table 19-16: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/NW)	CTS	Clear to Send. The UART_STAT.CTS bit holds the value (if UART_CTL.FCPOL set) or the complement value (if UART_CTL.FCPOL cleared) of the UART_CTS input pin. The UART_CTL.ACTS bit must be set to enable this feature. The core can read the value of the UART_STAT.CTS bit to determine whether the external device is ready to receive (UART_STAT.CTS set) or if it is busy (UART_STAT.CTS cleared). If UART_CTL.ACTS is cleared, the UART_TX handshaking protocol is disabled, and the UART transmits data as long as there is data to transmit, regardless of the value of UART_STAT.CTS. When UART_CTL.ACTS is cleared, the software can pause transmission temporarily by setting the XOFF bit. Note that in loopback mode (UART_CTL.LOOP_EN set), the UART_STAT.CTS bit is disconnected from the UART_CTS input pin. Instead, the bit is directly connected to the UART_CTL.MRTS bit.	
		0	Not clear to send (External device not ready to receive)
		1	Clear to send (External device ready to receive)
12 (R/W1C)	SCTS	Sticky CTS. The UART_STAT.SCTS bit is a sticky bit that is set when UART_STAT.CTS transitions from 0 to 1. The UART_STAT.SCTS bit is cleared by software with a W1C operation. This bit can trigger a line status interrupt if enabled by the UART_IMSK_SET.EDSSI bit.	
		0	CTS has not transitioned from low to high
		1	CTS has transitioned from low to high
11 (R/NW)	RO	Reception On-going.	
		0	No data reception in progress
		1	Data reception in progress
10 (R/W1S)	ADDR	Address Bit Status. The UART_STAT.ADDR bit is used to mirror the address bit of the word in UART_RBR in multi-drop bus protocol, and is enabled only in MDB mode. The UART_STAT.ADDR bit is updated by hardware upon detecting a received word with the address bit in UART_RBR set or cleared. Additionally, software can set the ADDR bit with a write-1-to-set (W1S) operation.	
		0	Address bit is low
		1	Address bit is high

Table 19-16: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W1C)	ASTKY	Address Sticky. The UART_STAT.ASTKY bit is used in multi-drop bus mode to indicate whether a peripheral is currently being addressed. This bit is a sticky version of the UART_STAT.ADDR bit and is set by hardware when setting the UART_STAT.ADDR bit. The UART_STAT.ASTKY bit can only be cleared by software with a write-one-to-clear (W1C) operation. With the UART_STAT.ASTKY bit set, words will be received irrespective of the UART_CTL.MOD bit or UART_STAT.ADDR bit selection. With the UART_STAT.ASTKY bit cleared, only address words (UART_CTL.MOD bit set) will be received and words with UART_CTL.MOD bit cleared are ignored (not moved from the RSR to the RX FIFO) in MDB mode. The UART_STAT.ASTKY bit does not affect reception in non-MDB modes.	
		0	ADDR bit has not been set
		1	ADDR bit has been set
8 (R/W1C)	TFI	Transmission Finished Indicator. The UART_STAT.TFI bit is a sticky version of the UART_STAT.TEMT bit. While UART_STAT.TEMT is automatically cleared by hardware when new data is written to the UART_THR register, the sticky UART_STAT.TFI bit remains set, until it is cleared by software (W1C). The UART_STAT.TFI bit enables more flexible transmit interrupt timing.	
		0	TEMT did not transition from 0 to 1
		1	TEMT transition from 0 to 1
7 (R/NW)	TEMT	TSR and THR Empty. The UART_STAT.TEMT bit indicates that the UART_THR and UART_TAIP registers and the UART_TSR register are empty. In this case, the program is permitted to write to the UART_THR and UART_TAIP registers twice without losing data. The UART_STAT.TEMT bit can also be used as indicator that pending UART transmission is completed. At that time, it is safe to disable the UART_CTL.EN bit or to three-state the off-chip line driver.	
		0	Not empty TSR/THR
		1	TSR/THR Empty
5 (R/NW)	THRE	Transmit Hold Register Empty. The UART_STAT.THRE bit indicates that the UART transmit channel is ready for new data and software can write to the UART_THR and UART_TAIP registers. Writes to the UART_THR and UART_TAIP registers clear the UART_STAT.THRE. The bit is set again when the UART_THR and UART_TAIP registers are empty and ready to accept data.	
		0	Not empty THR/TAIP
		1	Empty THR/TAIP

Table 19-16: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W1C)	BI	Break Indicator. The UART_STAT.BI bit indicates that the first stop bit is sampled low and the entire data word, including parity bit, consists of low bits only. (This condition indicates that UART_RX was held low for more than the maximum word length.) The UART_STAT.BI bit is updated simultaneously with the UART_STAT.DR bit, that is, by the time the first stop bit is received or when data is loaded from the receive FIFO to the UART_RBR register. The bit is sticky and can be cleared by W1C operations.	
		0	No break interrupt
		1	Break interrupt this indicates UARTxRX was held low(RPOLC=0) / high (RPOLC=1) for more than the maximum word length
3 (R/W1C)	FE	Framing Error. The UART_STAT.FE bit indicates that the first stop bit is sampled. This bit is updated simultaneously with the UART_STAT.DR bit, that is, by the time the first stop bit is received or when data is loaded from the receive FIFO to the UART_RBR register. The UART_STAT.FE bit is sticky and can be cleared by W1C operations. Note that invalid stop bits can be simulated by setting the UART_CTL.FFE bit.	
		0	No error
		1	Invalid stop bit error
2 (R/W1C)	PE	Parity Error. The UART_STAT.PE bit indicates that the received parity bit does not match the expected value. This bit is updated simultaneously with the UART_STAT.DR bit, that is, by the time the first stop bit is received or when data is loaded from the receive FIFO to the UART_RBR register. The UART_STAT.PE bit is sticky and can be cleared by W1C operations. Note that invalid parity bits can be simulated by setting the UART_CTL.FPE bit.	
		0	No parity error
		1	Parity error

Table 19-16: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	OE	Overrun Error. The UART_STAT.OE bit indicates that further data is received while the internal receive buffer was full. This bit is set when sampling the stop bit of the sixth data word. To avoid overruns, read the UART_RBR register in time. In DMA receive mode, overruns are very unlikely to happen ever. After an overrun occurs, the UART_RBR and receive FIFO are protected from being overwritten by new data until the UART_STAT.OE bit is cleared by software. The content of the UART_RSR register is lost as soon as the overrun occurs. The UART_STAT.OE bit is sticky and can be cleared by W1C operations.	
		0	No overrun
		1	Overrun error
0 (R/NW)	DR	Data Ready. The UART_STAT.DR bit indicates that data is available in the receiver and can be read from the UART_RBR register. The bit is set by hardware when the receiver detects the first valid stop bit. The bit is cleared by hardware when the UART_RBR register is read.	
		0	No new data
		1	New data in RBR

Transmit Address/Insert Pulse Register

The UART_TAIP register and the UART_THR register share the same physical register, but UART_TAIP has different effect than the UART_THR register when UART_TAIP is written to in MDB and UART modes.

In MDB mode, data written to the UART_TAIP register is transmitted as an address frame (as with the UART_CTL.MOD bit set).

In UART mode, a write to UART_TAIP causes a pulse of value UART_TAIP [7] for a duration of UART_TAIP [6:0] x bit time. (There is additional inversion if the UART_CTL.TPOLC bit is set).

Bit time is defined by the UART_CLK register. The transmission of the pulse is followed by stop bit transmission as specified by the UART_CTL.STB and UART_CTL.STBH bits. This could be used for supporting line break command and inter-frame gap.

In IrDA mode, writes to UART_TAIP is treated the same as writes to UART_THR.

Accesses to the UART_TAIP register have the same affects as the UART_THR register with respect to the UART_STAT.THRE, UART_STAT.TEMT, and UART_STAT.TFI flags.

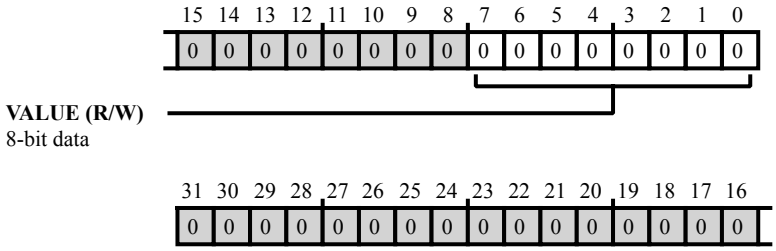

Figure 19-21: UART_TAIP Register Diagram

Table 19-17: UART_TAIP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	8-bit data.

Transmit Hold Register

The write-only UART_THR register is the UART's transmit buffer. The UART_STAT.THRE bit indicates whether data can be written to UART_THR. Writes to this register automatically propagate to the internal UART_TSR register as soon as UART_TSR is ready. Then, transmit operation is initiated immediately.

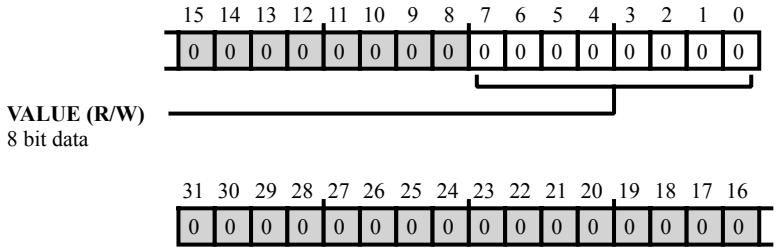

Figure 19-22: UART_THR Register Diagram

Table 19-18: UART_THR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	8 bit data.

Transmit Shift Register

The read only UART_TSR register which returns the content of the UART's transmit shift register.

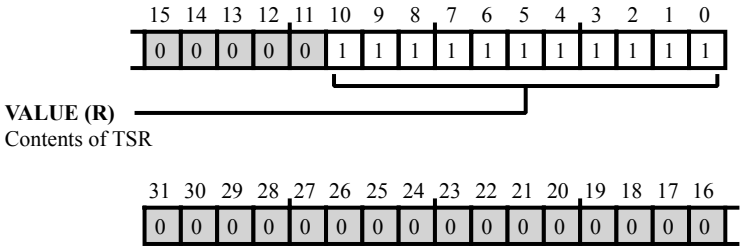

Figure 19-23: UART_TSR Register Diagram

Table 19-19: UART_TSR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/NW)	VALUE	Contents of TSR.

Transmit Counter Register

The UART_TXCNT read only register returns the content of 16-bit counter in the UART transmitter. This count is used for baud rate clock generation (the lower [15:0] is the count data).

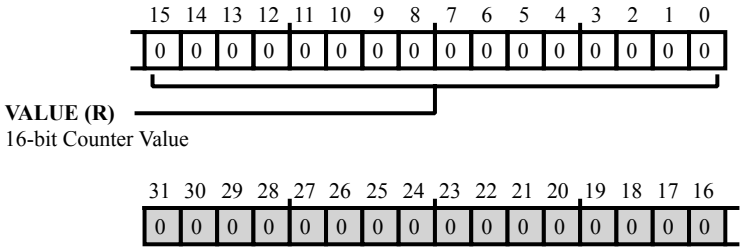

Figure 19-24: UART_TXCNT Register Diagram

Table 19-20: UART_TXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	16-bit Counter Value.

20 Two-Wire Interface (TWI)

The processor has a two-wire interface (TWI), that provides a simple exchange method of control data between multiple devices. The TWI module is compatible with the widely used I^2C bus standard. Additionally, the TWI module is fully compatible with serial camera control bus (SCCB) functionality for easier control of various CMOS camera sensor devices.

The TWI module offers the capabilities of simultaneous master and slave operation and support for both 7-bit addressing and multimedia data arbitration. The TWI interface uses two pins for transferring clock (`TWI_SCL`) and data (`TWI_SDA`) and supports the protocol at speeds up to 400K bits/sec. The TWI interface pins are compatible with 5-V logic levels.

To preserve processor bandwidth, the TWI module can be set up with transfer-initiated interrupts to only service FIFO buffer data reads and writes. Protocol-related interrupts are optional. The TWI externally moves 8-bit data while maintaining compliance with the I^2C bus protocol.

TWI Features

The TWI is fully compatible with the widely used I^2C bus standard.

The TWI controller includes the following features.

- Simultaneous master and slave operation on multiple device systems
- Support for multi-master bus arbitration
- 7-bit addressing
- 100K bits/second and 400K bits/second data rates
- General call address support
- Master clock synchronization and support for clock low extension
- Separate multiple-byte receive and transmit FIFOs
- Low interrupt rate
- Individual override control of data and clock lines in the event of bus lock-up
- Input filter for spike suppression

- Serial camera control bus support as specified in the *OmniVision Serial Camera Control Bus (SCCB) Functional Specification*

TWI Functional Description

The TWI interface is a shift register that serially transmits and receives data bits. It moves data 1 bit at a time at the SCL rate, to and from other TWI devices. The SCL signal synchronizes the shifting and sampling of the data on the serial data pin.

ADSP-CM40x TWI Register List

The Two-Wire Interface controller TWI allows a device to interface to an inter-IC bus as specified by the Philips I^2C Bus Specification version 2.1, dated January 2000. A set of registers governs TWI operations. For more information on TWI functionality, see the TWI register descriptions.

Table 20-1: ADSP-CM40x TWI Register List

Name	Description
TWI_CLKDIV	SCL Clock Divider Register
TWI_CTL	Control Register
TWI_FIFOCTL	FIFO Control Register
TWI_FIFOSTAT	FIFO Status Register
TWI_IMSK	Interrupt Mask Register
TWI_ISTAT	Interrupt Status Register
TWI_MSTRADDR	Master Mode Address Register
TWI_MSTRCTL	Master Mode Control Registers
TWI_MSTRSTAT	Master Mode Status Register
TWI_RXDATA16	Rx Data Double-Byte Register
TWI_RXDATA8	Rx Data Single-Byte Register
TWI_SLVADDR	Slave Mode Address Register
TWI_SLVCTL	Slave Mode Control Register
TWI_SLVSTAT	Slave Mode Status Register
TWI_TXDATA16	Tx Data Double-Byte Register
TWI_TXDATA8	Tx Data Single-Byte Register

ADSP-CM40x TWI Interrupt List

Table 20-2: ADSP-CM40x TWI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
99	TWI0_DATA	TWI0 Data Interrupt	Level	

External Interface

The TWI_SDA (serial data) and TWI_SCL (serial clock) signals are open drain and require pull-up resistors. These bidirectional signals externally interface the TWI controller to the I^2C bus and no other external connections or logic are necessary.

Serial Clock Signal (SCL)

The serial clock signal (TWI_SCL) is an input in slave mode. In master mode, the TWI controller must set this signal to the desired frequency.

The TWI controller supports the standard mode of operation (up to 100 kHz) or fast mode (up to 400 kHz). The TWI control register (TWI_CTL) sets the TWI_CTL.PRESCALE value which sets the relationship between the system clock (SYSCLK) and the internally timed events of the TWI controller. The internal time reference is derived from SYSCLK using a prescaled value. The prescale value is the number of SYSCLK periods used in the generation of one internal time reference. Set the value of prescale to create an internal time reference with a period of 10 MHz. It is represented as a 7-bit binary value as follows.

PRESCALE = f_{SYSCLK}/10MHz

NOTE: It is not always possible to achieve 10-MHz accuracy. In such cases, it is safe to round up the PRESCALE value to the next highest integer. For example, if SYSCLK is 100 MHz, the PRESCALE value is calculated as 100 MHz/10 MHz = 10. A prescale value of 14 in this case ensures that all timing requirements are met.

During master mode operation, the TWI module uses the TWI_CLKDIV register values to create the minimum TWI_CLKDIV.CLKHI and TWI_CLKDIV.CLKLO durations of the TWI_SCL signal. The TWI_CLKDIV.CLKHI field specifies the minimum number of 10-MHz time reference periods the TWI_SCL waits before a new clock low period begins, assuming a single master. (The 10-MHz time reference periods are represented as an 8-bit binary value). The TWI uses the TWI_CLKDIV.CLKLO field to specify the minimum number of internal time reference periods (represented as an 8-bit binary value). The TWI_SCL signal is held low.

Serial clock frequencies can vary from 400 kHz to less than 20 kHz. The resolution of the clock generated is 1/10 MHz or 100 ns. The following equation describes the frequency.

TWI_CLKDIV = TWI_SCL period/10 MHz time reference.

For example, for an TWI_SCL of 400 kHz (period = 1/400 kHz = 2500 ns) and an internal time reference of 10 MHz (period = 100 ns), the following equation applies:

TWI_CLKDIV = 2500 ns/100 ns = 25

Therefore, a TWI_SCL signal with a 30% duty cycle has TWI_CLKDIV.CLKLO=17 and TWI_CLKDIV.CLKHI=8. Adding TWI_CLKDIV.CLKLO and TWI_CLKDIV.CLKHI equals TWI_CLKDIV.

NOTE: The TWI_CLKDIV.CLKHI and TWI_CLKDIV.CLKLO fields are not intended to guarantee a certain frequency. Rather, they guarantee a certain minimum high and low duration for the TWI_SCL signal. Slew rate controls falling edges. The *RC* time constant governs the rising edges. The pull-up resistor and the TWI_SCL capacitance form the time constant. See the "Register Descriptions" section for more details.

Serial Data Signal (SDA)

The TWI transmits and receives serial data, depending on the direction of the transfer, on the bidirectional serial data signal (SDA).

Internal Interface

The peripheral bus interface supports the transfer of 16-bit wide data. The processor uses the interface in the support of register and FIFO buffer reads and writes. The TWI internal interface is comprised of the blocks described as follows.

Register block. Contains all control and status bits and reflects what can be written or read as outlined by the programming model. Each function block updates their corresponding status bits.

FIFO buffer. Configured as a 1-byte-wide, 2-deep transmit FIFO buffer and a 1-byte-wide, 2-deep receive FIFO buffer.

Transmit shift register. Serially shifts its data out externally off chip. The output can be controlled for generation of acknowledgments or it can be manually overwritten.

Receive shift register. Receives its data serially from off chip. The receive shift register is 1 byte wide and data received can either be transferred to the FIFO buffer or used in an address comparison.

Address compare block. Supports address comparison in the event the TWI controller module is accessed as a slave.

Prescaler block. Must be programmed to generate a 10-MHz time reference relative to the system clock. The block uses this time base for filtering of data and timing events specified by the electrical data sheet (See the Philips specification). The block uses the time base to generate the TWI_SCL clock as well.

Clock generation module. Generates an external TWI_SCL clock when in master mode. It includes the logic necessary for synchronization in a multi-master clock configuration and clock stretching when configured in slave mode.

TWI Architectural Concepts

The TWI controller follows the transfer protocol of the Philips I^2C Bus specification version 2.1 dated January 2000.

TWI Protocol

The *Data Transfer* figure shows a simple complete transfer.

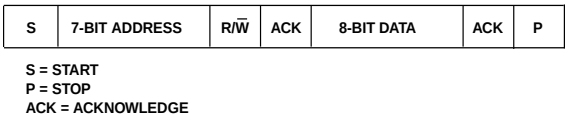

Figure 20-1: Data Transfer

The TWI controller register contents maps to a basic transfer. The *Data Transfer with Bit Illustration* figure details the same transfer from the *Data Transfer* figure noting the corresponding TWI controller bit names. In this illustration, the TWI controller successfully transmits 1 byte of data. The slave has acknowledged both address and data.

Figure 20-2: Data Transfer with Bit Illustration

Clock Generation and Synchronization

The TWI controller implementation only issues a clock during master mode operation and only at the time a transfer initiates. If arbitration for the bus is lost, the serial clock output immediately three-states. If multiple clocks attempt to drive the serial clock line, the TWI controller synchronizes its clock with the other remaining clocks. The *Clock Synchronization* figure shows this functionality.

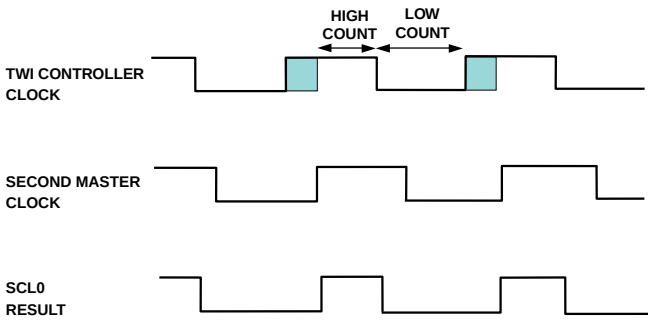

Figure 20-3: Clock Synchronization

The TWI controller serial clock (`TWI_SCL`) output follows these rules:

- Once the clock high (`TWI_CLKDIV.CLKHI`) count is complete, the serial clock output is driven low and the clock low (`TWI_CLKDIV.CLKLO`) count begins.
- Once the clock low count is complete, the serial clock line is three-stated. This state allows the external pull-up resistor to pull the `TWI_SCL` signal high. The clock synchronization logic enters into a delay mode (shaded area) until the `TWI_SCL` signal is detected at logic 1 level. Now, the clock high count begins.

Bus Arbitration

The TWI controller initiates a master mode transmission only when the bus is idle. If the bus is idle and two masters initiate a transfer, arbitration for the bus begins. The ***Bus Arbitration*** figure shows the arbitration.

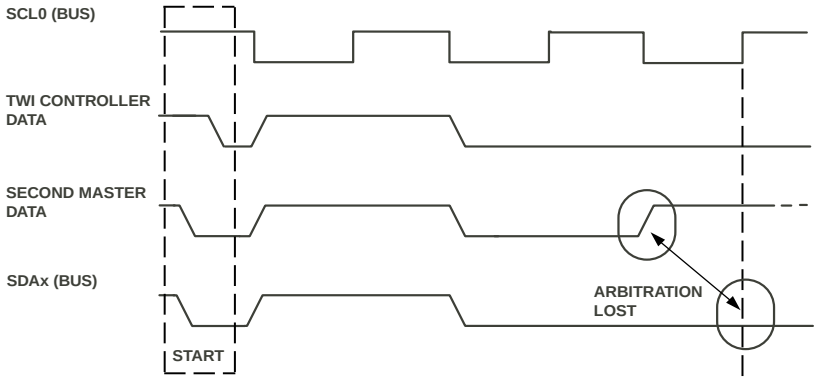

Figure 20-4: Bus Arbitration

The TWI controller monitors the serial data bus (SDA) while the `TWI_SCL` signal is high. If the `TWI_SDA` signal is determined to be an active logic 0 level while the data of the TWI controller is a logic 1 level, the TWI controller has lost arbitration. It stops generating the clock and data signals. Arbitration is not only performed at the serial clock edges, but also during the entire time the `TWI_SCL` signal is high.

Start and Stop Conditions

Start and stop conditions involve serial data transitions while the serial clock is a logic 1 level. The TWI controller generates and recognizes these transitions. Typically, start and stop conditions occur at the beginning and at the conclusion of a transmission, except repeated start combined transfers. The ***Start and Stop Conditions*** figure shows the transitions.

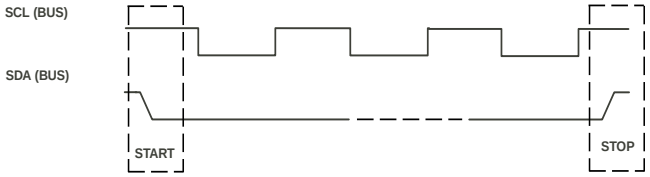

Figure 20-5: Start and Stop Conditions

The TWI special case start and stop conditions of the TWI controller include the following.

- Controller addressed as a slave-receiver. If the master asserts a stop condition during the data phase of a transfer, the TWI controller concludes the transfer (`TWI_ISTAT.SCOMP`).
- Controller addressed as a slave-transmitter. If the master asserts a stop condition during the data phase of a transfer, the TWI controller concludes the transfer (`TWI_ISTAT.SCOMP`) and indicates a slave transfer error (`TWI_ISTAT.SERR`).

- Controller as a master-transmitter or master-receiver. If the stop bit (TWI_MSTRCTL.STOP) is set during an active master transfer, the TWI controller issues a stop condition as soon as possible avoiding any error conditions. (The TWI controller operates as if data transfer count had been reached).

General Call Support

The TWI controller always decodes and acknowledges a general call address if:

- The TWI controller is enabled as a slave, and
- General call is enabled

The TWI_SLVCTL.GEN bit configures general call addressing (0x00) only when the TWI controller is a slave-receiver.

If the data associated with the transfer is (NAK) not acknowledged, the TWI_SLVCTL.NAK bit can be set. If the TWI controller issues a general call as a master-transmitter, set the appropriate address (TWI_MSTRADDR register) and transfer direction (TWI_MSTRCTL.DIR bit) and load the transmit FIFO data.

NOTE: The byte following the general call address usually defines the slaves response to the call. The interpretation of the command in the second byte is based on the value of its LSB. For a TWI slave device, the bytes received after the general call address are considered data.

Fast Mode

Fast mode essentially uses the same mechanics as the standard mode of operation. Fast mode affects electrical specifications and timing. When fast mode is enabled, (FAST) timing is modified to meet the following electrical requirements.

- Serial data rise times before arbitration evaluation (t_r)
- Stop condition set-up time from serial clock to serial data (t_{SUSTO})
- Bus free time between a stop and start condition (t_{BUF})

TWI Operating Modes

The TWI has two modes of operation, *repeated start* and *clock stretching*. The following sections describe the operating modes.

Repeated Start

A repeated start condition is the absence of a stop condition between two transfers. The two transfers can be of any direction type. Examples include a transmit followed by a receive, or a receive followed by a transmit. The following sections guide the programmer in developing a service routine.

Transmit Receive Repeated Start

The ***Repeated Start Followed by Data Receive*** figure shows a repeated start followed by a data receive sequence. The shading in the figure indicates that the slave has control of the bus.

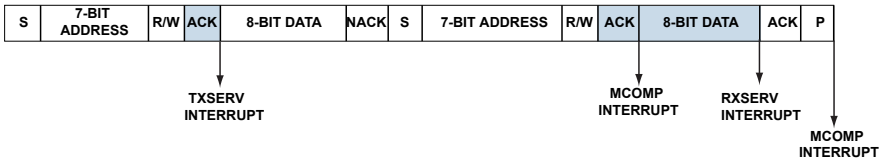

Figure 20-6: Repeated Start Followed by Data Receive

The tasks performed at each interrupt are:

- Transmit FIFO service (`TWI_ISTAT.TXSERV`) interrupt. This interrupt is generated due to a FIFO access. Since this byte is the last of this transfer, the TWI uses the `TWI_FIFOSTAT` register to indicate that the transmit FIFO is empty. When read, `TWI_MSTRCTL.DCNT` bit field=0. Set the `TWI_MSTRCTL.RSTART` bit to indicate a repeated start and set the `TWI_MSTRCTL.DIR` bit if the following transfer is a data receive.
- Master transfer complete (`TWI_ISTAT.MCOMP`) interrupt. This interrupt is generated when all data transfers (`TWI_MSTRCTL.DCNT` bit field=0). If no errors occur, a start condition initiates. Clear the `TWI_MSTRCTL.RSTART` bit and program the `TWI_MSTRCTL.DCNT` bits with the desired number of bytes to receive.
- Receive FIFO service (`TWI_ISTAT.RXSERV`) interrupt. This interrupt is generated due to the arrival of a byte in the receive FIFO. Simple data handling is the only requirement.
- Master transfer complete (`TWI_ISTAT.MCOMP`) interrupt. The transfer completes.

Receive Transmit Repeated Start

The ***Repeated Start Data Receive Followed by Data Transmit*** figure illustrates a repeated start data receive followed by a data transmit sequence. The shading in the figure indicates that the slave has control of the bus.

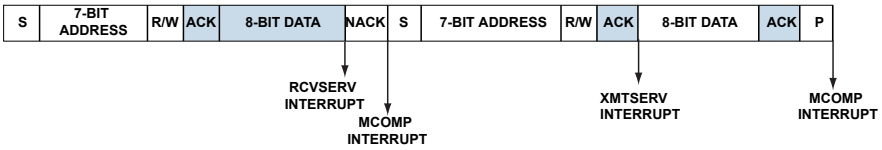

Figure 20-7: Repeated Start Data Receive Followed by Data Transmit

The tasks performed at each interrupt are:

- Receive FIFO service (`TWI_ISTAT.RXSERV`) interrupt. This interrupt is generated due to the arrival of a data byte in the receive FIFO. Set the `TWI_MSTRCTL.RSTART` bit to indicate a repeated start and clear the `TWI_MSTRCTL.DIR` bit if the following transfer is a data transmit.

- Master transfer complete (TWI_ISTAT.MCOMP) interrupt. This interrupt has occurred due to the completion of the data receive transfer. If no errors occur, a start condition initiates. Clear the TWI_MSTRCTL.RSTART bit and program the TWI_MSTRCTL.DCNT bits with the desired number of bytes to transmit.
- Transmit FIFO service (TWI_ISTAT.TXSERV) interrupt. This interrupt is generated due to a FIFO access. Simple data handling is the only requirement.
- Master transfer complete (TWI_ISTAT.MCOMP) interrupt. The transfer is complete.

NOTE: There is no timing constraint to meet the conditions—program the bits as required. Refer to Clock Stretching During Repeated Start section for more on how the controller stretches the clock during repeated start transfers.

Clock Stretching

Clock stretching is an added function of the TWI controller in master mode operation. This behavior uses self-induced stretching of the I ^{2}C clock while waiting to service interrupts. Hardware initiates stretching automatically. No programming is necessary. The TWI controller as a master supports three modes of clock stretching:

- Clock Stretching During FIFO Underflow
- Clock Stretching During FIFO Overflow
- Clock Stretching During Repeated Start

Clock Stretching During FIFO Underflow

During a master mode transmit, an interrupt occurs the instant the transmit FIFO becomes empty. The most recent byte begins transmission. If the TWI_ISTAT.TXSERV interrupt is not serviced, the concluding acknowledge phase of the transfer stretches.

Stretching of the clock continues until new data bytes are written to the transmit FIFO (TWI_TXDATA8 or TWI_TXDATA16 registers). No other action is required to release the clock and continue the transmission. This behavior continues until the transmission completes (TWI_MSTRCTL.DCNT=0). The transmission concludes (TWI_ISTAT.MCOMP). The ***Clock Stretching during FIFO Underflow*** figure and table show the stretching.

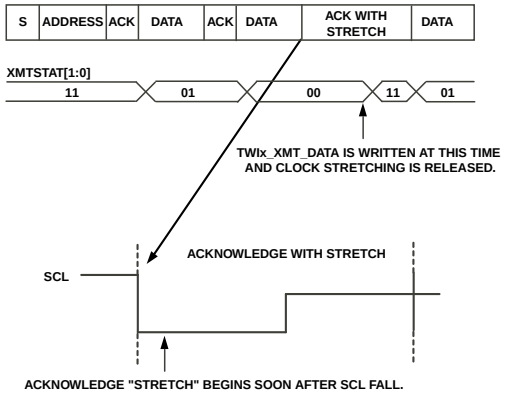

Figure 20-8: Clock Stretching during FIFO Underflow

TWI Controller	Processor
Interrupt: XMTSERV – Transmit FIFO buffer is empty.	Acknowledge: Clear the interrupt source bits. Write to the transmit FIFO buffer.
...	...
Interrupt: MCOMP – Master transmit complete (DCNT= 0x00).	Acknowledge: Clear the interrupt source bits.

Clock Stretching During FIFO Overflow

During a master mode receive operation, an interrupt occurs at the instant the receive FIFO becomes full. It is during the acknowledge phase of this received byte that clock stretching begins. The TWI module makes no attempt to initiate the reception of another byte. Stretching of the clock continues until the data bytes previously received are read from the receive FIFO buffer (`TWI_RXDATA8` or `TWI_RXDATA16` registers). No other action is required to release the clock and continue the reception of data. This behavior continues until the reception is complete (`TWI_MSTRCTL.DCNT`=0). Reception concludes (`TWI_ISTAT.MCOMP`). The ***Clock Stretching During FIFO Overflow*** figure and table show the clock stretching.

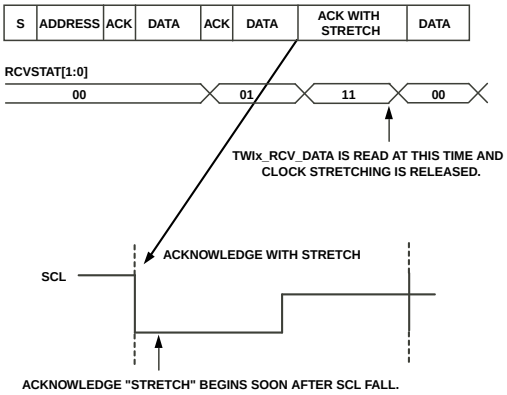

Figure 20-9: Clock Stretching During FIFO Overflow

TWI Controller	Processor
Interrupt: RCVSERV – Receive FIFO buffer is full.	Acknowledge: Clear the interrupt source bits. Read the receive FIFO buffer.
...	...
Acknowledge: Clear the interrupt source bits.	Interrupt: MCOMP – Master receive complete.

Clock Stretching During Repeated Start

The repeated start feature in I^2C protocol requires a transition between two subsequent transfers. With the use of clock stretching, the task of managing transitions becomes simpler and common to all transfer types.

Once an initial TWI master transfer completes (transmit or receive), the clock initiates a stretch during the repeated start phase between transfers. Concurrent with this event, the initial transfer generates a `TWI_ISTAT.MCOMP` interrupt to signify the initial transfer has completed (`TWI_MSTRCTL.DCNT`=0). This initial transfer is handled without any special bit setting sequences or timing.

The clock stretching logic described applies here. With no system-related timing constraints, the subsequent transfer (receive or transmit) is set up and activated. This sequence can repeat as many times as required to string a series of repeated start transfers together. The ***Clock Stretching during Repeated Start Condition*** figure and table show the clock stretching.

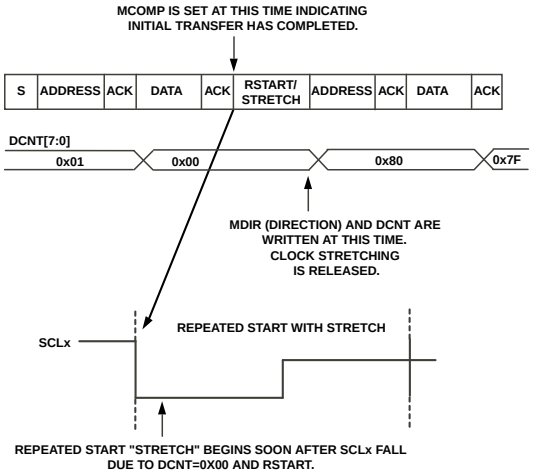

Figure 20-10: Clock Stretching during Repeated Start Condition

TWI Controller	Processor
Interrupt: MCOMP – Initial transmit has completed and DCNT = 0x00. Note: transfer in progress, RSTART previously set.	Acknowledge: Clear the interrupt source bits. Write to TWIx_MASTER_CTL, setting MDIR (receive), clearing RSTART, and setting new DCNT value (nonzero).
Interrupt: RCVSERV – Receive FIFO is full.	Acknowledge: Clear the interrupt source bits. Read the receive FIFO buffer.

TWI Controller	Processor
...	...
Interrupt: MCOMP – Master receive complete	Acknowledge: Clear the interrupt source bits.

TWI Programming Model

The topics in this section provide information on the basic programming steps required to set up and run the two wire interface.

The following sections provide programming steps for general setup, and master and slave modes.

General Setup

General setup refers to register writes that are required for both slave mode and master mode operations.

Perform general setup before setting either the master or slave enable bits.

1. Program the `TWI_CTL.EN` bit to enable the TWI controller and set the prescale value (`TWI_CTL.PRESCALE` bit).
2. Program the prescale value to the binary representation of f_{SCLK}/10 MHz. Round up all values to the next whole number.
3. Set the `TWI_CTL.EN` bit to enable the controller.

Once the TWI controller is enabled, a bus busy condition can be detected. This condition clears after t_{BUF} has expired, assuming no additional bus activity has been detected.

Slave Mode

When enabled, slave mode operation supports both receive and transmit data transfers.

It is not possible to enable only one data transfer direction and not acknowledge (NAK) the other. The following setup reflects this functionality.

1. Program the `TWI_SLVADDR` register. The TWI uses the appropriate 7 bits in determining a match during the address phase of the transfer.
2. Program the `TWI_TXDATA8.VALUE` or `TWI_TXDATA16` registers. These values are the initial data values for transmission when the slave is addressed and transmission is needed. This step is optional. If no data is written when the slave is addressed and transmission is needed, the serial clock (`TWI_SCL`) stretches. An interrupt is generated until data is written to the transmit FIFO.
3. Program the `TWI_IMSK` register. There are enable-bits associated with the desired interrupt sources. For example, programming the value 0x000F results in an interrupt output to the processor, when the TWI module

detects a valid address match. An interrupt also occurs when a valid slave transfer completes or has an error, or a subsequent transfer has begun and the previous transfer has not been serviced.

4. Program the TWI_SLVCTL register. This step prepares and enables slave mode operation. For example, programming the value 0x0005 enables slave mode operation and requires 7-bit addressing. It indicates that data in the transmit FIFO buffer is for slave mode transmission.

The ***Slave Mode Interaction*** table and ***TWI Slave Mode Program Flow*** diagram represent the interaction between the TWI controller and the processor using this example.

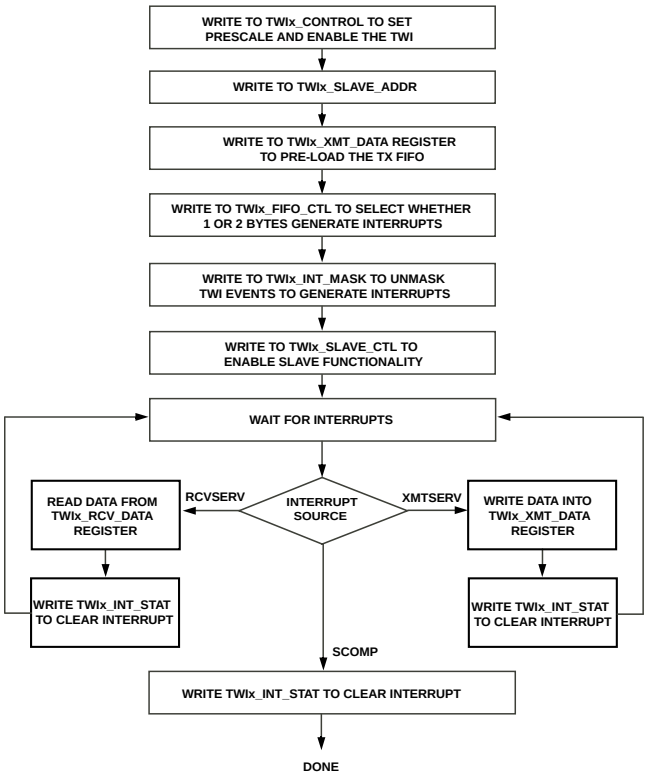

Figure 20-11: TWI Slave Mode Program Flow

Table 20-3: Slave Mode Interaction

TWI Controller	Processor
Interrupt: SINIT – Slave transfer in progress.	Acknowledge: Clear the interrupt source bits.
Interrupt: RCVSERV – Receive buffer is full.	Acknowledge: Clear the interrupt source bits. Read TWIx_FIFO_STAT. Read the receive FIFO buffer.
...	...
Interrupt: SCOMP – Slave transfer complete.	Acknowledge: Clear the interrupt source bits. Read the receive FIFO buffer.

Master Mode Program Flow

The ***Master Mode Program Flow*** figure shows the program for the TWI in master mode.

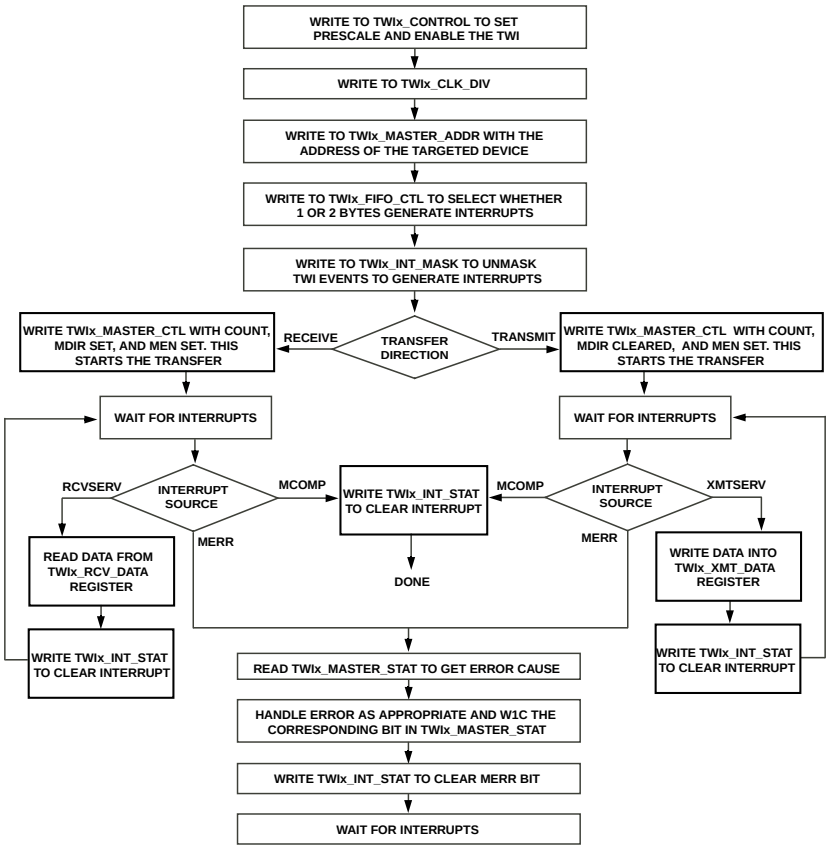

Figure 20-12: Master Mode Program Flow

Master Mode Clock Setup

Master mode operation is set up and executed on a per-transfer basis.

An example of programming steps for a receive and for a transmit is given separately in following sections. The programming step for clock setup listed here is common to both transfer types.

1. Program the `TWI_CLKDIV` register to define the minimum high and minimum low duration for the clock.

The `TWI_CLKDIV.CLKHI` and `TWI_CLKDIV.CLKLO` fields do not guarantee a certain frequency. Rather, they guarantee a certain minimum high and low duration for `TWI_SCL`. The slew rate controls falling edges. The RC time constant formed by the pull-up resistor and the SCL capacitance govern rising edges. See the “Register Descriptions” section for more details.

Master Mode Transmit

Follow these programming steps for a single master mode transmission:

1. Program the `TWI_MSTRADDR` register. This step defines the address transmitted during the address phase of the transfer.

2. Program the TWI_TXDATA8 or TWI_TXDATA16 register. This step configures the initial data transmitted. It is an error to complete the address phase of the transfer and not have data available in the transmit FIFO buffer.
3. Program the TWI_FIFOCTL register. The programming indicates if the transmit FIFO buffer interrupts occur with each byte transmitted (8-bits) or with every 2 bytes transmitted (16-bits).
4. Program the TWI_IMSK register. This step enables the bits associated with the desired interrupt sources. For example, programming the value 0x0030 results in an interrupt output to the processor when the master transfer completes, and the master transfer has an error.
5. Program the TWI_MSTRCTL register. This step prepares and enables master mode operation. For example, programming the value 0x0201: enables master mode operation, generates a 7-bit address, sets the direction to master-transmit, uses standard mode timing, and transmits 8 data bytes before generating a stop condition.

The ***Master Mode Transmit Setup Interaction*** table represents the interaction between the TWI controller and the processor using this example.

Table 20-4: Master Mode Transmit Setup Interaction

TWI Controller	Processor
Interrupt: XMTSERV – Transmit buffer is empty.	Acknowledge: Clear the interrupt source bits. Write to the transmit FIFO buffer.
...	...
Interrupt: MCOMP – Master transfer complete.	Acknowledge: Clear the interrupt source bits.

Master Mode Receive

Follow these programming steps for a single master mode receive.

1. Program the TWI_MSTRADDR register. This step defines the address transmitted during the address phase of the transfer.
2. Program the TWI_FIFOCTL register. This step indicates if the receive FIFO buffer interrupts occur with each byte received (8-bits) or with every 2 bytes received (16-bits).
3. Program the TWI_IMSK register. This step configures the enable bits associated with the desired interrupt sources. For example, programming the value 0x0030 results in an interrupt output to the processor when the master transfer completes, and the master transfer has an error.
4. Program the TWI_MSTRCTL register. This step prepares and enables master mode operation. For example, programming the value 0x0205: enables master mode operation, generates a 7-bit address, sets the direction to master-receive, uses standard mode timing, and receives 8 data bytes before generating a stop condition.

The ***Master Mode Receive Setup Interaction*** table shows the interaction between the TWI controller and the processor using this example.

Table 20-5: Master Mode Receive Setup Interaction

TWI Controller	Processor
Interrupt: RCVSERV – Receive buffer is full.	Acknowledge: Clear the interrupt source bits. Read the receive FIFO buffer.
...	...
Interrupt: MCOMP – Master transfer complete.	Acknowledge: Clear the interrupt source bits. Read the receive FIFO buffer.

NOTE: After the `TWI_MSTRCTL.DCNT` bit decrements to zero, the TWI master device sends a NAK to indicate to the slave transmitter to release the bus. This operation allows the master to send the stop signal to terminate the transfer.

ADSP-CM40x TWI Register Descriptions

2-Wire Interface (TWI) contains the following registers.

Table 20-6: ADSP-CM40x TWI Register List

Name	Description
TWI_CLKDIV	SCL Clock Divider Register
TWI_CTL	Control Register
TWI_FIFOCTL	FIFO Control Register
TWI_FIFOSTAT	FIFO Status Register
TWI_IMSK	Interrupt Mask Register
TWI_ISTAT	Interrupt Status Register
TWI_MSTRADDR	Master Mode Address Register
TWI_MSTRCTL	Master Mode Control Registers
TWI_MSTRSTAT	Master Mode Status Register
TWI_RXDATA16	Rx Data Double-Byte Register
TWI_RXDATA8	Rx Data Single-Byte Register
TWI_SLVADDR	Slave Mode Address Register
TWI_SLVCTL	Slave Mode Control Register
TWI_SLVSTAT	Slave Mode Status Register
TWI_TXDATA16	Tx Data Double-Byte Register
TWI_TXDATA8	Tx Data Single-Byte Register

SCL Clock Divider Register

During master mode operation, the TWI_CLKDIV holds values, which the TWI uses to create the high and low durations of the serial clock (SCL). The clock signal SCL is an output in master mode and an input in slave mode. The values in the TWI_CLKDIV.CLKLO and TWI_CLKDIV.CLKHI fields add up to the CLKDIV value the following equation.

CLKDIV = TWI SCL period / 10 MHz time reference

Serial clock frequencies can vary from 400 KHz to less than 20 KHz. The resolution of the clock generated is 1/10 MHz or 100 ns. For example, for an SCL of 400 KHz (period = 1/400 KHz = 2500 ns) and an internal time reference of 10 MHz (period = 100 ns):

CLKDIV = 2500 ns / 100 ns = 25

For an SCL with a 30% duty cycle, use TWI_CLKDIV.CLKLO = 17 and TWI_CLKDIV.CLKHI = 8.

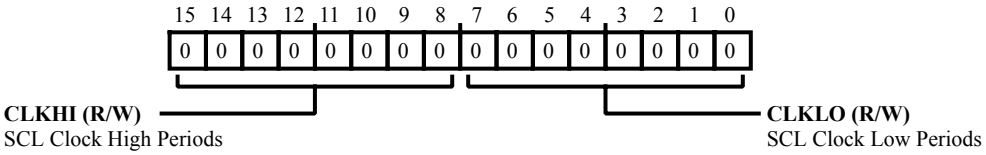

Figure 20-13: TWI_CLKDIV Register Diagram

Table 20-7: TWI_CLKDIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	CLKHI	SCL Clock High Periods. The TWI_CLKDIV.CLKHI specifies the number of 10 MHz time reference periods the serial clock (SCL) waits before a new clock low period begins, assuming a single master.
7:0 (R/W)	CLKLO	SCL Clock Low Periods. The TWI_CLKDIV.CLKLO specifies the number of internal time reference periods the serial clock (SCL) is held low.

Control Register

The TWI_CTL enables the TWI, establishes a relationship between the system clock (SYSCLK) and the TWI controller's internally timed events, and enables SCCB compatibility.

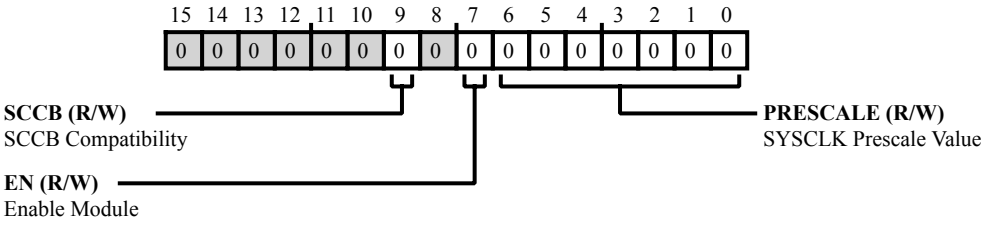

Figure 20-14: TWI_CTL Register Diagram

Table 20-8: TWI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	SCCB	SCCB Compatibility. The TWI_CTL.SCCB enables SCCB compatible operation for the TWI. SCCB compatibility is an optional feature and should not be used in an I^2C bus system. When this feature is enabled, all slave asserted acknowledgement bits are ignored by this master. This feature is valid only during transfers where the TWI is mastering an SCCB bus. Slave mode transfers should be avoided when this feature is enabled because the TWI controller always generates an acknowledge in slave mode.	
		0	Disable SCCB compatibility When disabled, Master transfers are not SCCB compatible.
		1	Enable SCCB compatibility When enabled, Master transfers are SCCB compatible. All slave-asserted acknowledgement bits are ignored by this master.
7 (R/W)	EN	Enable Module. The TWI_CTL.EN enables TWI controller operation for either master and/or slave mode of operation. It is recommended that this bit be set at the time TWI_CTL.PRESCALE is initialized and remain set. This method guarantees accurate operation of bus busy detection logic.	
		0	Disable
		1	Enable
6:0 (R/W)	PRESCALE	SYSCLK Prescale Value. The TWI_CTL.PRESCALE holds the pre-scaled value for the TWI internal time reference. This reference is derived from SYSCLK according to the formula: TWI_CTL.PRESCALE = f_{SYSCLK}/10MHz	

Table 20-8: TWI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		The TWI_CTL.PRESCALE specifies the number of system clock (SYSCLK) periods used in the generation of one internal time reference. The value of TWI_CTL.PRESCALE must be set to create an internal time reference with a period of 10 MHz. It is represented as a 7-bit binary value.

FIFO Control Register

The TWI_FIFOCTL control bits affect only the FIFO and are not tied in any way with master or slave mode operation.

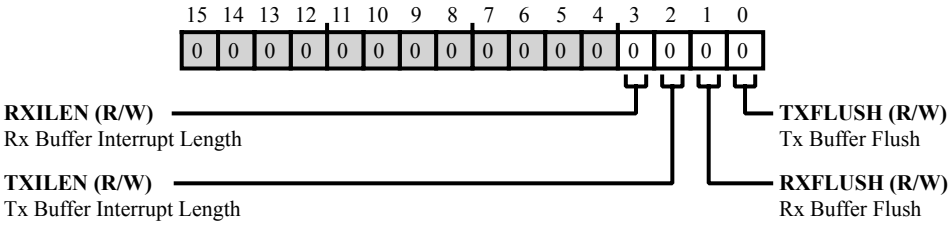

Figure 20-15: TWI_FIFOCTL Register Diagram

Table 20-9: TWI_FIFOCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	RXILEN	Rx Buffer Interrupt Length. The TWI_FIFOCTL.RXILEN determines the rate at which receive buffer interrupts are to be generated. Interrupts may be generated with each byte received or after two bytes are received. Interrupt status is available in TWI_FIFOSTAT.RXSTAT.	
		0	RXSERVI on 1 or 2 Bytes in FIFO
		1	RXSERVI on 2 Bytes in FIFO
2 (R/W)	TXILEN	Tx Buffer Interrupt Length. The TWI_FIFOCTL.TXILEN determines the rate at which transmit buffer interrupts are to be generated. Interrupts may be generated with each byte transmitted or after two bytes are transmitted. Interrupt status is available in TWI_FIFOSTAT.TXSTAT.	
		0	TXSERVI on 1 Byte of FIFO Empty
		1	TXSERVI on 2 Bytes of FIFO Empty
1 (R/W)	RXFLUSH	Rx Buffer Flush. The TWI_FIFOCTL.RXFLUSH directs the TWI to flush the contents of the receive buffer and update TWI_FIFOSTAT.RXSTAT to indicate the buffer is empty. This state is held until this bit is cleared. During an active receive, the receive buffer in this state responds to the receive logic as if it is full.	
		0	Normal Operation of Rx Buffer
		1	Flush Rx Buffer

Table 20-9: TWI_FIFOCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	TXFLUSH	Tx Buffer Flush. The `TWI_FIFOCTL.TXFLUSH` directs the TWI to flush the contents of the transmit buffer and update `TWI_FIFOSTAT.TXSTAT` to indicate the buffer is empty. This state is held until this bit is cleared. During an active transmit, the transmit buffer in this state responds to the transmit logic as if it is empty.	
		0	Normal Operation of Tx Buffer
		1	Flush Tx Buffer

FIFO Status Register

The TWI_FIFOSTAT fields indicate the state of the FIFO buffers' receive and transmit contents. The FIFO buffers do not discriminate between master data and slave data. By using the status and control bits provided, the FIFO can be managed to allow simultaneous master and slave operation.

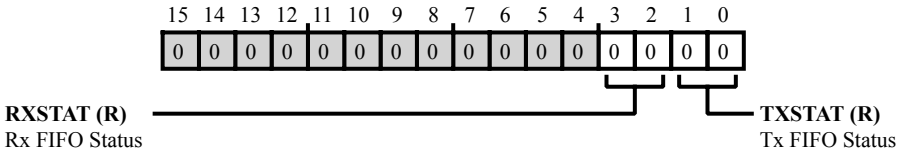

Figure 20-16: TWI_FIFOSTAT Register Diagram

Table 20-10: TWI_FIFOSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3:2 (R/NW)	RXSTAT	Rx FIFO Status. The read-only TWI_FIFOSTAT.RXSTAT indicates the number of valid data bytes in the receive FIFO buffer. The status is updated with each FIFO buffer read using the peripheral data bus or write access by the receive shift register. Simultaneous accesses are allowed.	
		0	Empty The FIFO is empty.
		1	Contains 1 Byte The FIFO contains one byte of data. A single byte peripheral read of the FIFO is allowed.
		2	Reserved
		3	Full The FIFO is full and contains two bytes of data. Either a single or double byte peripheral read of the FIFO is allowed.
1:0 (R/NW)	TXSTAT	Tx FIFO Status. The read-only TWI_FIFOSTAT.TXSTAT field indicates the number of valid data bytes in the FIFO buffer. The status is updated with each FIFO buffer write using the peripheral data bus or read access by the transmit shift register. Simultaneous accesses are allowed.	
		0	Empty The FIFO is empty. Either a single or double byte peripheral write of the FIFO is allowed.
		1	Contains 1 Byte The FIFO contains one byte of data. A single byte peripheral write of the FIFO is allowed.
		2	Reserved
		3	Full The FIFO is full and contains two bytes of data.

Interrupt Mask Register

The TWI_IMSK enables interrupt sources to assert the interrupt output. Each mask bit corresponds with one interrupt source bit in TWI_ISTAT. Reading and writing TWI_IMSK does not affect the contents of the TWI_ISTAT.

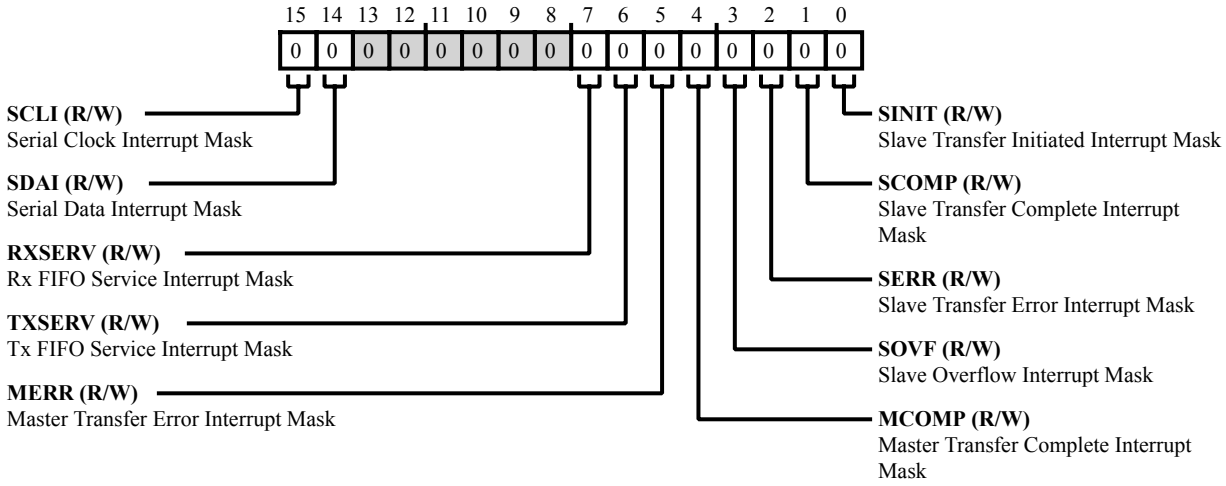

Figure 20-17: TWI_IMSK Register Diagram

Table 20-11: TWI_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	SCLI	Serial Clock Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
14 (R/W)	SDAI	Serial Data Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
7 (R/W)	RXSERV	Rx FIFO Service Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
6 (R/W)	TXSERV	Tx FIFO Service Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
5 (R/W)	MERR	Master Transfer Error Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt

Table 20-11: TWI_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	MCOMP	Master Transfer Complete Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
3 (R/W)	SOVF	Slave Overflow Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
2 (R/W)	SERR	Slave Transfer Error Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
1 (R/W)	SCOMP	Slave Transfer Complete Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
0 (R/W)	SINIT	Slave Transfer Initiated Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt

Interrupt Status Register

The TWI_ISTAT contains information about functional areas requiring servicing. Many of the bits serve as an indicator to further read and service various status registers. After servicing the interrupt source associated with a bit, the user must clear that interrupt source bit by writing a 1 to it.

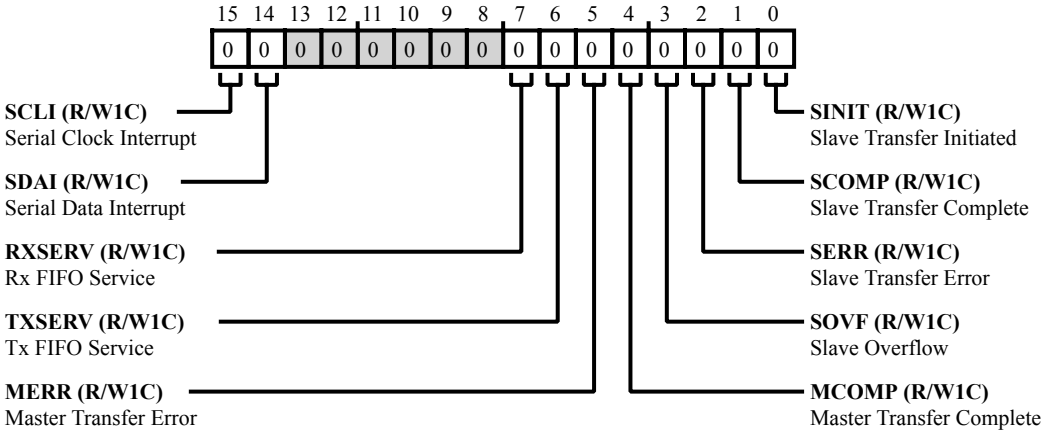

Figure 20-18: TWI_ISTAT Register Diagram

Table 20-12: TWI_ISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	SCLI	Serial Clock Interrupt. If the TWI is enabled (TWI_CTL.EN), SCLI is set on a high-to-low transition of the serial clock pin (SCLx). Normally, this bit is not required for I^2C bus transfers. It will be initially set on an I^2C transfer and does not require clearing.	
		0	No Interrupt No transition was detected on the SCLx pin.
		1	Interrupt Detected A high-to-low transition was detected on the SCLx pin. This bit is W1C.
14 (R/W1C)	SDAI	Serial Data Interrupt. If the TWI is enabled (TWI_CTL.EN), SDAI is set on a high-to-low transition of the serial data pin (SDAx). Normally, this bit is not required for I^2C bus transfers. It will be initially set on an I^2C transfer and does not require clearing.	
		0	No Interrupt No transition was detected on the SDAx pin.
		1	Interrupt Detected A high-to-low transition was detected on the SDAx pin. This bit is W1C.

Table 20-12: TWI_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	RXSERV	Rx FIFO Service. If TWI_FIFOCTL.RXILEN =0, the TWI_ISTAT.RXSERV is set each time the TWI_FIFOSTAT.RXSTAT field is updated to either 01 or 11. If TWI_FIFOCTL.RXILEN =1, the TWI_ISTAT.RXSERV is set each time TWI_FIFOSTAT.RXSTAT is updated to 11.	
		0	No Interrupt The FIFO does not require servicing, or the TWI_FIFOSTAT.RXSTAT field has not changed since this bit was last cleared.
		1	Interrupt Detected The receive FIFO buffer has one or two 8-bit words of data available to be read.
6 (R/W1C)	TXSERV	Tx FIFO Service. If TWI_FIFOCTL.TXILEN =0, the TWI_ISTAT.TXSERV is set each time the TWI_FIFOSTAT.TXSTAT field is updated to either 01 or 00. If TWI_FIFOCTL.TXILEN =1, the TWI_ISTAT.TXSERV is set each time TWI_FIFOSTAT.TXSTAT is updated to 00.	
		0	No Interrupt FIFO does not require servicing, or the TWI_FIFOSTAT.TXSTAT field has not changed since this bit was last cleared.
		1	Interrupt Detected The transmit FIFO buffer has one or two 8-bit locations available to be written.
5 (R/W1C)	MERR	Master Transfer Error. The TWI_ISTAT.MERR indicates that a master error has occurred. The conditions surrounding the error are indicated by the master status register (TWI_MSTRSTAT).	
		0	No Interrupt
		1	Interrupt Detected
4 (R/W1C)	MCOMP	Master Transfer Complete. The TWI_ISTAT.MCOMP indicates that the initiated master transfer has completed. In the absence of a repeat start, the bus has been released.	
		0	No Interrupt
		1	Interrupt Detected
3 (R/W1C)	SOVF	Slave Overflow. The TWI_ISTAT.SOVF indicates that the TWI_ISTAT.SCOMP bit was set at the time a subsequent transfer has acknowledged an address phase. The transfer continues, however, it may be difficult to delineate data of one transfer from another.	
		0	No Interrupt
		1	Interrupt Detected

Table 20-12: TWI_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	SERR	Slave Transfer Error. The `TWI_ISTAT.SERR` indicates that a slave error has occurred. A restart or stop condition has occurred during the data receive phase of a transfer.	
		0	No Interrupt
		1	Interrupt Detected
1 (R/W1C)	SCOMP	Slave Transfer Complete. The `TWI_ISTAT.SCOMP` indicates that the transfer is complete and either a stop, or a restart was detected.	
		0	No Interrupt
		1	Interrupt Detected
0 (R/W1C)	SINIT	Slave Transfer Initiated. The `TWI_ISTAT.SINIT` indicates whether or not a slave transfer is in progress.	
		0	No Interrupt A transfer is not in progress, or an address match has not occurred since the last time this bit was cleared.
		1	Interrupt Detected The slave has detected an address match, and a transfer has been initiated.

Master Mode Address Register

During the addressing phase of a transfer, the TWI controller, with its master enabled, transmits the contents of TWI_MSTRADDR. When programming this register, omit the read/write bit. That is, only the upper 7 bits that make up the slave address should be written to this register. For example, if the slave address is b#1010000X, where X is the read/write bit, the TWI_MSTRADDR is programmed with b#1010000, which corresponds to 0x50. When sending out the address on the bus, the TWI controller appends the read/write bit as appropriate based on the state of the TWI_MSTRCTL.DIR bit.

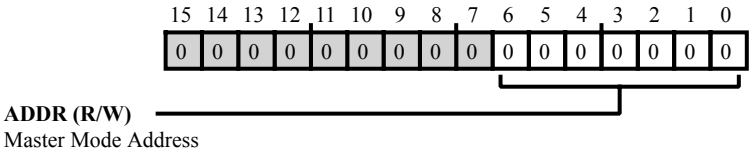

Figure 20-19: TWI_MSTRADDR Register Diagram

Table 20-13: TWI_MSTRADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	ADDR	Master Mode Address.

Master Mode Control Registers

The TWI_MSTRCTL controls the logic associated with master mode operation. Bits in this register do not affect slave mode operation and should not be modified to control slave mode functionality.

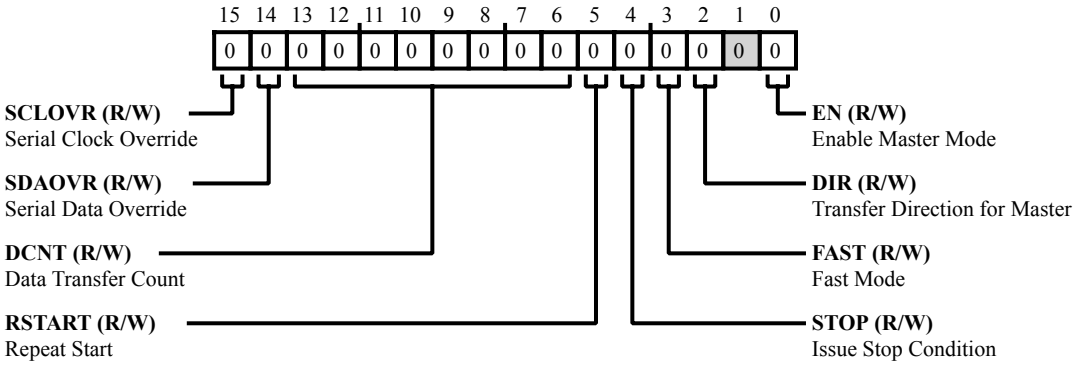

Figure 20-20: TWI_MSTRCTL Register Diagram

Table 20-14: TWI_MSTRCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	SCLOVR	Serial Clock Override. The TWI_MSTRCTL.SCLOVR provides direct control of the serial clock line when required. Normal master and slave mode operation should not require override operation. When TWI_MSTRCTL.SCLOVR is set, the TWI overrides normal serial clock output, driving it to an active 0 level and overriding all other logic. This state is held until this bit is cleared. When TWI_MSTRCTL.SCLOVR is cleared, the TWI permits normal serial clock operation under the control of master mode clock generation and slave mode clock stretching logic.	
		0	Permit Normal SCL Operation
		1	Override Normal SCL Operation
14 (R/W)	SDAOVR	Serial Data Override. The TWI_MSTRCTL.SDAOVR provides direct control of the serial data line when required. Normal master and slave mode operation should not require override operation. When TWI_MSTRCTL.SDAOVR is set, the TWI overrides normal serial data operation under the control of the transmit shift register and acknowledge logic, driving serial data output to an active 0 level and overriding all other logic. This state is held until this bit is cleared. When TWI_MSTRCTL.SDAOVR is cleared, the TWI permits normal serial data operation.	
		0	Permit Normal SDA Operation
		1	Override Normal SDA Operation
13:6 (R/W)	DCNT	Data Transfer Count.	

Table 20-14: TWI_MSTRCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		The TWI_MSTRCTL.DCNT indicates the number of data bytes to transfer. As each data word is transferred, the TWI decrements this counter. When TWI_MSTRCTL.DCNT decrements to 0, a stop condition is generated. Setting TWI_MSTRCTL.DCNT to 0xFF disables the counter. In this transfer mode, data continues to be transferred until it is concluded by setting the TWI_MSTRCTL.STOP bit. In the event a master transmit is aborted due to a slave data NAK, the value of TWI_MSTRCTL.DCNT equals the number of bytes not sent. The byte which was NAK'ed by the slave is counted as a sent byte.	
5 (R/W)	RSTART	Repeat Start. The TWI_MSTRCTL.RSTART enables the TWI to issue a repeat start condition at the conclusion of the current transfer (TWI_MSTRCTL.DCNT =0) and begin the next transfer. The current transfer concludes with updates to the appropriate status and interrupt bits. If errors occurred during the previous transfer, a repeat start does not occur. In the absence of any errors, master enable (TWI_MSTRCTL.EN) does not self clear on a repeat start.	
		0	Disable Repeat Start
		1	Enable Repeat Start
4 (R/W)	STOP	Issue Stop Condition. The TWI_MSTRCTL.STOP directs the TWI to issue a stop condition. The transfer concludes as soon as possible avoiding any error conditions (as if data transfer count had been reached). At that time, the TWI_IMSK is updated along with any associated status bits.	
		0	Permit Normal Operation
		1	Issue Stop
3 (R/W)	FAST	Fast Mode. The TWI_MSTRCTL.FAST selects whether the TWI operates in fast mode or standard mode. In fast mode, the TWI uses timing specifications for transfers at up to 400K bits/s. In standard mode, the TWI uses timing specifications for transfers at up to 100K bits/s.	
		0	Select Standard Mode
		1	Select Fast Mode
2 (R/W)	DIR	Transfer Direction for Master. The TWI_MSTRCTL.DIR selects the transfer direction for the TWI as master initiated receive or transmit.	
		0	Master Transmit
		1	Master Receive

Table 20-14: TWI_MSTRCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Enable Master Mode. The `TWI_MSTRCTL.EN` enables master mode functionality. A start condition is generated if the bus is idle. This bit self clears at the completion of a transfer (after `TWI_MSTRCTL.DCNT` decrements to zero), including transfers terminated due to errors. If disabled (=0) during operation, the transfer is aborted, and all logic associated with master mode transfers are reset. Serial data and serial clock (SDA, SCL) are no longer driven. Write-1-to-clear status bits are not affected.	
		0	Disable
		1	Enable

Master Mode Status Register

The TWI_MSTRSTAT holds information during master mode transfers and at their conclusion. Generally, master mode status bits are not directly associated with the generation of interrupts, but these bits offer information on the current transfer. Slave mode operation does not affect master mode status bits.

Note that while TWI_MSTRSTAT.SCLSEN is set (this condition could be due to having no pull-up resistor on TWI_SCL or another agent is driving TWI_SCL low), the acknowledge bits (TWI_MSTRSTAT.ANAK and TWI_MSTRSTAT.DNAK) do not update. This result occurs because the acknowledge conditions are sampled during the high phase of TWI_SCL.

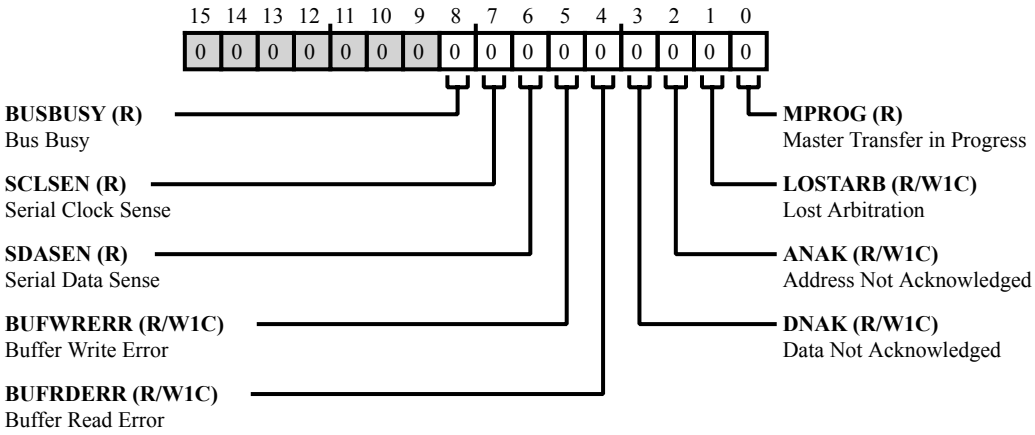

Figure 20-21: TWI_MSTRSTAT Register Diagram

Table 20-15: TWI_MSTRSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	BUSBUSY	Bus Busy. The TWI_MSTRSTAT.BUSBUSY indicates whether the bus is currently busy or free. This indication is not limited to only this device but is for all devices. On a start condition, the setting of the register value is delayed due to the input filtering. On a stop condition the clearing of the register value occurs after t_{BUF}.	
		0	Bus Free The bus is free. The clock and data bus signals have been inactive for the appropriate bus free time.
		1	Bus Busy The bus is busy. Clock or data activity has been detected.

Table 20-15: TWI_MSTRSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	SCLSEN	Serial Clock Sense. The TWI_MSTRSTAT.SCLSEN indicates the active or inactive state of the serial clock. Use this status bit when direct sensing of the serial clock line is required. The register value is delayed due to the input filter (nominally 50 ns). Normal master and slave mode operation should not require this feature.	
		0	SCL Inactive "One" An inactive "one" is being sensed on the serial clock.
		1	SCL Active "Zero" An active "zero" is being sensed on the serial clock. The source of the active driver is not known and can be internal or external.
6 (R/NW)	SDASEN	Serial Data Sense. The TWI_MSTRSTAT.SDASEN indicates the active or inactive status of the serial data. Use this status bit when direct sensing of the serial data line is required. The register value is delayed due to the input filter (nominally 50 ns). Normal master and slave mode operation should not require this feature.	
		0	SDA Inactive "One" An inactive "one" is currently being sensed on the serial data line.
		1	SDA Active "Zero" An active "zero" is currently being sensed on the serial data line. The source of the active driver is not known and can be internal or external.
5 (R/W1C)	BUFWRERR	Buffer Write Error. The TWI_MSTRSTAT.BUFWRERR indicates whether the current master transfer was aborted due to a receive buffer write error. The receive buffer and receive shift register were both full at the same time. This bit is W1C.	
		0	No Status
		1	Buffer Write Error
4 (R/W1C)	BUFRDERR	Buffer Read Error. The TWI_MSTRSTAT.BUFRDERR indicates whether the current master transfer was aborted due to the detection of a NAK during data transmission. This bit is W1C.	
		0	No Status
		1	Buffer Read Error
3 (R/W1C)	DNAK	Data Not Acknowledged. The TWI_MSTRSTAT.DNAK indicates whether the current master transfer was aborted due to the detection of a NAK during data transmission. This bit is W1C.	
		0	No Status
		1	Data NAK

Table 20-15: TWI_MSTRSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	ANAK	Address Not Acknowledged. The `TWI_MSTRSTAT.ANAK` indicates whether the current master transfer was aborted due to the detection of a NAK during the address phase of the transfer. This bit is W1C.	
		0	No Status
		1	Address NAK
1 (R/W1C)	LOSTARB	Lost Arbitration. The `TWI_MSTRSTAT.LOSTARB` indicates whether the current transfer was aborted due to the loss of arbitration with another master. This bit is W1C.	
		0	No Status
		1	Lost Arbitration
0 (R/NW)	MPROG	Master Transfer in Progress. The `TWI_MSTRSTAT.MPROG` indicates whether or not a master transfer is in progress. If clear (`TWI_MSTRSTAT.MPROG` =0), currently no transfer is taking place. This can occur after a transfer is complete or while an enabled master is waiting for an idle bus.	
		0	No Status
		1	Master Transfer in Progress

Rx Data Double-Byte Register

The TWI_RXDATA16 holds a 16-bit data value read from the FIFO buffer. To reduce interrupt output rates and peripheral bus access times, a double byte receive data access can be performed. Two data bytes can be read, effectively emptying the receive FIFO buffer with a single access.

The data is read in little endian byte order, where byte 0 is the first byte received and byte 1 is the second byte received. With each access, the receive status (TWI_FIFOSTAT.RXSTAT) field is updated to indicate it is empty. If an access is performed while the FIFO buffer is not full, the read data is unknown and the existing FIFO buffer data and its status remains unchanged.

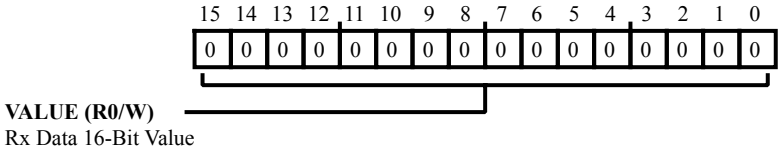

Figure 20-22: TWI_RXDATA16 Register Diagram

Table 20-16: TWI_RXDATA16 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R0/W)	VALUE	Rx Data 16-Bit Value.

Rx Data Single-Byte Register

The TWI_RXDATA8 holds an 8-bit data value read from the FIFO buffer. Receive data is read from the corresponding receive buffer in a first-in first-out order. Although peripheral bus reads are 16 bits, a read access to TWI_RXDATA8 accesses only one transmit data byte from the FIFO buffer. With each access, the receive status (TWI_FIFOSTAT.RXSTAT) field is updated. If an access is performed while the FIFO buffer is empty, the data is unknown and the FIFO buffer status remains indicating it is empty.

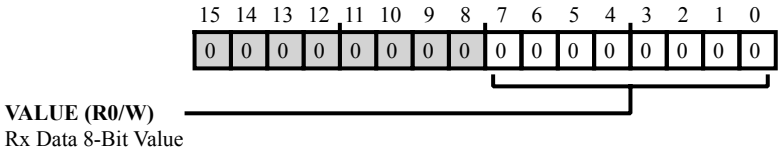

Figure 20-23: TWI_RXDATA8 Register Diagram

Table 20-17: TWI_RXDATA8 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W)	VALUE	Rx Data 8-Bit Value.

Slave Mode Address Register

The TWI_SLVADDR holds the slave mode address, which is the valid address to which the slave-enabled TWI controller responds. The TWI controller compares this value with the received address during the addressing phase of a transfer.

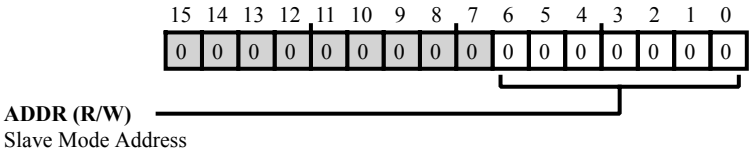

Figure 20-24: TWI_SLVADDR Register Diagram

Table 20-18: TWI_SLVADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	ADDR	Slave Mode Address.

Slave Mode Control Register

The TWI_SLVCTL controls the logic associated with slave mode operation. Settings in this register do not affect master mode operation and should not be modified to control master mode functionality.

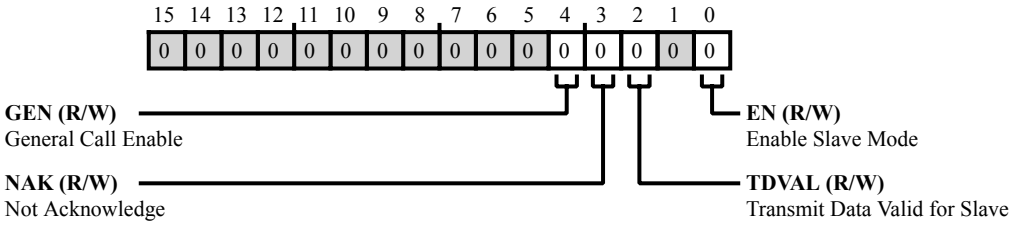

Figure 20-25: TWI_SLVCTL Register Diagram

Table 20-19: TWI_SLVCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	GEN	General Call Enable. The TWI_SLVCTL.GEN enables general call address matching. When enabled, a general call slave receive transfer is accepted. All status and interrupt source bits associated with transfers are updated. Note that general call address detection is available only when slave mode is enabled.	
		0	Disable General Call Matching
		1	Enable General Call Matching
3 (R/W)	NAK	Not Acknowledge. The TWI_SLVCTL.NAK directs the TWI to generate a NAK (if set) or an ACK (if cleared) at the conclusion of data transfer for slave receive. For NAK, the slave is still considered to be addressed at the conclusion of transfer.	
		0	Generate ACK
		1	Generate NAK
2 (R/W)	TDVAL	Transmit Data Valid for Slave. The TWI_SLVCTL.TDVAL selects whether the data in the transmit FIFO is available (valid) for slave transmission (TWI_SLVCTL.TDVAL set). If the FIFO data is not available (invalid) for slave transmission (TWI_SLVCTL.TDVAL cleared), the data in the transmit FIFO is for master mode transmits, and the data is not allowed to be used during a slave transmit; the transmit FIFO is treated as if it is empty.	
		0	Data Invalid for Slave Tx
		1	Data Valid for Slave Tx

Table 20-19: TWI_SLVCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Enable Slave Mode. The `TWI_SLVCTL.EN` enables slave operation. Enabling slave and master modes of operation concurrently is allowed. If disabled, no attempt is made to identify a valid address. If `TWI_SLVCTL.EN` is cleared during a valid transfer, clock stretching ceases, the serial data line is released, and the current byte is not acknowledged.	
		0	Disable
		1	Enable

Slave Mode Status Register

During and at the conclusion of register slave mode transfers, the TWI_SLVSTAT holds information on the current transfer. Generally slave mode status bits are not associated with the generation of interrupts. Master mode operation does not affect slave mode status bits.

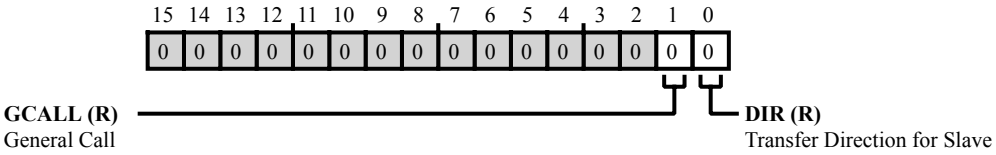

Figure 20-26: TWI_SLVSTAT Register Diagram

Table 20-20: TWI_SLVSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	GCALL	General Call. The TWI_SLVSTAT.GCALL indicates whether or not--at the time of addressing--the address was determined to be a general call. This bit self clears if slave mode is disabled (TWI_SLVCTL.EN =0).	
		0	Not a General Call Address
		1	General Call Address
0 (R/NW)	DIR	Transfer Direction for Slave. The TWI_SLVSTAT.DIR indicates whether--at the time of addressing--the transfer direction was determined to be slave transmit or receive. This bit self clears if slave mode is disabled (TWI_SLVCTL.EN =0).	
		0	Slave Receive
		1	Slave Transmit

Tx Data Double-Byte Register

The TWI_TXDATA16 register holds a 16-bit data value written into the FIFO buffer. To reduce interrupt output rates and peripheral bus access times, a double byte transfer data access can be done. Two data bytes can be written, effectively filling the transmit FIFO buffer with a single access.

The data is written in little endian byte order, where byte 0 is the first byte to be transferred and byte 1 is the second byte to be transferred. With each access, the transmit status (TWI_FIFOSTAT.TXSTAT) field is updated. If an access is performed while the FIFO buffer is not empty, the write is ignored and the existing FIFO buffer data and its status remains unchanged. This register when read back returns zero.

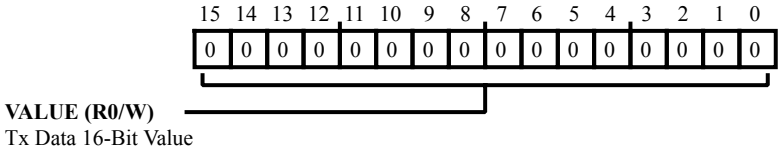

Figure 20-27: TWI_TXDATA16 Register Diagram

Table 20-21: TWI_TXDATA16 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R0/W)	VALUE	Tx Data 16-Bit Value.

Tx Data Single-Byte Register

The TWI_TXDATA8 register holds an 8-bit data value written into the FIFO buffer. Transmit data is entered into the corresponding transmit buffer in a first-in first-out order. For 16-bit peripheral bus writes, a write access to this register adds only one transmit data byte to the FIFO buffer. With each access, the transmit status (TWI_FIFOSTAT.TXSTAT) field is updated. If an access is performed while the FIFO buffer is full, the write is ignored and the existing FIFO buffer data and its status remains unchanged. This register returns zero when read back.

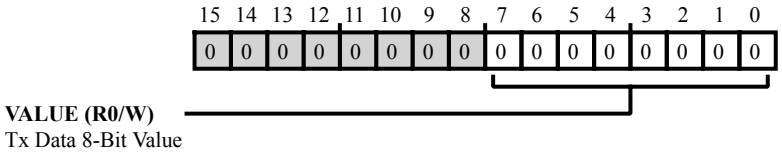

Figure 20-28: TWI_TXDATA8 Register Diagram

Table 20-22: TWI_TXDATA8 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W)	VALUE	Tx Data 8-Bit Value.

21 Controller Area Network (CAN)

The processor contains a Controller Area Network (CAN) module based on the CAN 2.0B (active) protocol. This protocol is an asynchronous communications protocol used in both industrial and automotive control systems. The CAN protocol is compatible with the control applications. It can communicate reliably over a network and incorporates CRC checking, message error tracking, and fault node confinement.

NOTE: This document assumes familiarity with the CAN standard. For more information, refer to Version 2.0 of the CAN specification from Robert Bosch GmbH.

CAN Features

Key features of the CAN module include:

- Conformity to the CAN 2.0B (active) standard
- Dedicated acceptance mask for each mailbox
- Support for data rates of up to 1M bit/s
- Support for standard (11-bit) and extended (29-bit) identifiers
- 32 mailboxes (8 transmit, 8 receive, 16 configurable)
- Data filtering (first 2 bytes) for acceptance filtering (DeviceNet™ mode)
- Error status and warning registers
- Universal counter-module
- Readable receive and transmit pin values
- Support for remote frames
- Active or passive network support
- Interrupts, including transmit or receive complete, error, and global
- Clock derived from SYSCLK through a programmable divider, eliminating the need for an extra crystal

CAN Functional Description

The following sections provide information on the functional operation of the CAN module. This section also provides listings of the CAN registers and interrupts.

ADSP-CM40x CAN Register List

The controller area network (CAN) module implements the CAN 2.0B (active) protocol. This protocol is an asynchronous communications protocol used in both industrial and automotive control systems. A set of registers govern CAN operations. For more information on CAN functionality, see the CAN register descriptions.

Table 21-1: ADSP-CM40x CAN Register List

Name	Description
CAN_AA1	Abort Acknowledge 1 Register
CAN_AA2	Abort Acknowledge 2 Register
CAN_AM[nn]H	Acceptance Mask (H) Register
CAN_AM[nn]L	Acceptance Mask (L) Register
CAN_CEC	Error Counter Register
CAN_CLK	Clock Register
CAN_CTL	CAN Master Control Register
CAN_DBG	Debug Register
CAN_ESR	Error Status Register
CAN_EWR	Error Counter Warning Level Register
CAN_GIF	Global CAN Interrupt Flag Register
CAN_GIM	Global CAN Interrupt Mask Register
CAN_GIS	Global CAN Interrupt Status Register
CAN_INT	Interrupt Pending Register
CAN_MBIM1	Mailbox Interrupt Mask 1 Register
CAN_MBIM2	Mailbox Interrupt Mask 2 Register
CAN_MBRIF1	Mailbox Receive Interrupt Flag 1 Register
CAN_MBRIF2	Mailbox Receive Interrupt Flag 2 Register
CAN_MBTD	Temporary Mailbox Disable Register
CAN_MBTIF1	Mailbox Transmit Interrupt Flag 1 Register
CAN_MBTIF2	Mailbox Transmit Interrupt Flag 2 Register
CAN_MB[nn]_DATA0	Mailbox Word 0 Register
CAN_MB[nn]_DATA1	Mailbox Word 1 Register

Table 21-1: ADSP-CM40x CAN Register List (Continued)

Name	Description
CAN_MB[nn]_DATA2	Mailbox Word 2 Register
CAN_MB[nn]_DATA3	Mailbox Word 3 Register
CAN_MB[nn]_ID0	Mailbox ID 0 Register
CAN_MB[nn]_ID1	Mailbox ID 1 Register
CAN_MB[nn]_LENGTH	Mailbox Length Register
CAN_MB[nn]_TIMESTAMP	Mailbox Timestamp Register
CAN_MC1	Mailbox Configuration 1 Register
CAN_MC2	Mailbox Configuration 2 Register
CAN_MD1	Mailbox Direction 1 Register
CAN_MD2	Mailbox Direction 2 Register
CAN_OPSS1	Overwrite Protection/Single Shot Transmission 1 Register
CAN_OPSS2	Overwrite Protection/Single Shot Transmission 2 Register
CAN_RFH1	Remote Frame Handling 1 Register
CAN_RFH2	Remote Frame Handling 2 Register
CAN_RML1	Receive Message Lost 1 Register
CAN_RML2	Receive Message Lost 2 Register
CAN_RMP1	Receive Message Pending 1 Register
CAN_RMP2	Receive Message Pending 2 Register
CAN_STAT	Status Register
CAN_TA1	Transmission Acknowledge 1 Register
CAN_TA2	Transmission Acknowledge 2 Register
CAN_TIMING	Timing Register
CAN_TRR1	Transmission Request Reset 1 Register
CAN_TRR2	Transmission Request Reset 2 Register
CAN_TRS1	Transmission Request Set 1 Register
CAN_TRS2	Transmission Request Set 2 Register
CAN_UCCNF	Universal Counter Configuration Mode Register
CAN_UCCNT	Universal Counter Register
CAN_UCRC	Universal Counter Reload/Capture Register

ADSP-CM40x CAN Interrupt List

Table 21-2: ADSP-CM40x CAN Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
123	CAN0_RX	CAN0 Receive Transfer Complete	Level	
124	CAN0_TX	CAN0 Transmit Transfer Complete	Level	
125	CAN0_STAT	CAN0 Status	Level	
126	CAN1_RX	CAN1 Receive Transfer Complete	Level	
127	CAN1_TX	CAN1 Transmit Transfer Complete	Level	
128	CAN1_STAT	CAN1 Status	Level	

External Interface

The interface to the CAN bus is a simple two-wire line. The following figure shows a symbolic representation of the CAN transceiver interconnection. Typically, the CAN_TX output and CAN_RX input pins of the processor connect to an external CAN CAN_TX and CAN_RX pins (respectively) of the tranceiver. The CAN_TX and CAN_RX pins operate with TTL levels and are appropriate for operation with CAN bus transceivers according to ISO/DIS 11898.

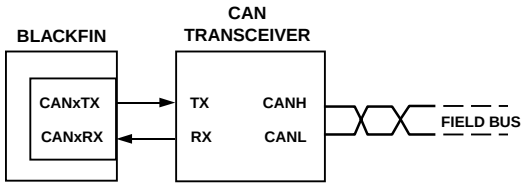

Figure 21-1: Representation of CAN Transceiver Interconnection

CAN data is either dominant (logic 0) or recessive (logic 1). The default state of the CAN_TX output is recessive.

Architectural Concepts

The full-CAN controller features 32 message buffers, called mailboxes. Eight mailboxes are dedicated for message transmission, eight are for reception, and 16 are programmable in direction.

The CAN module architecture is based around a 32-entry mailbox RAM. The CAN serial interface or the processor core accesses the mailbox sequentially. Each mailbox consists of eight 16-bit control and data registers and two optional 16-bit acceptance mask registers. Configure all of these registers before enabling the mailbox.

Since the mailbox area is implemented as RAM, the reset values of these registers are undefined. The ***CAN Mailbox Area*** figure shows the mailbox area. The data is divided into fields, which include a message identifier, a time stamp, a byte count, up to 8 bytes of data, and several control bits.

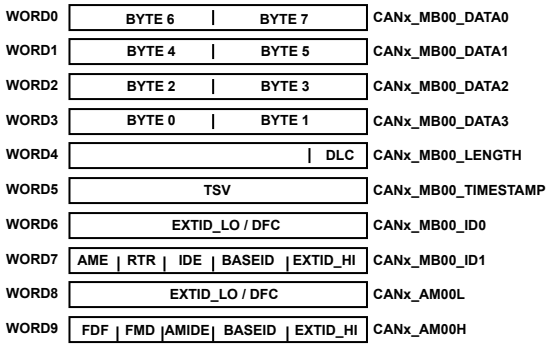

Figure 21-2: CAN Mailbox Area

The CAN mailbox identification (CAN_MB[nn]_ID0/1) register pair includes:

- The 29-bit identifier (base part CAN_AM[nn]H.BASEID plus extended part CAN_AM[nn]L.EXTID/ CAN_AM[nn]H.EXTID)
- The acceptance mask enable bit (CAN_MB[nn]_ID1.AME)
- The remote transmission request bit (CAN_MB[nn]_ID1.RTR)
- The identifier extension bit (CAN_MB[nn]_ID1.IDE)

NOTE: Do not write to the identifier of a message object while the mailbox is enabled for the CAN module (the corresponding bit in CAN_MC1 is set).

The other mailbox area registers and bits are:

- The data length code bit (CAN_MB[nn]_LENGTH.DLC). The upper 12 bits of this register of each mailbox are marked as reserved. Always, set these 12 bits to 0.
- The mailbox word registers (CAN_MB[nn]_DATA0/1/2/3) supply up to 8 bytes for the data field, sent MSB first from based on the number of bytes defined in the CAN_MB[nn]_LENGTH.DLC bit. For example, if only one byte is transmitted or received (CAN_MB[nn]_LENGTH.DLC=1), then it is stored in the most significant byte of the CAN_MB[nn]_DATA3 register.
- The time stamp value bits (CAN_MB[nn]_TIMESTAMP.TSV).

The final registers in the mailbox area are the acceptance mask registers (CAN_AM[nn]H and CAN_AM[nn]L). The acceptance mask is enabled when the CAN_MB[nn]_ID1.AME bit is set.

Enable the *filtering on data field* option by setting the CAN_CTL.DNM and CAN_AM[nn]H.FDF bits. When enabled, the CAN_MB[nn]_ID0.EXTID[15:0] bits are reused as acceptance code (DFC) for the data field filtering.

Block Diagram

The ***CAN Controller Block Diagram*** figure shows a block diagram of the CAN module.

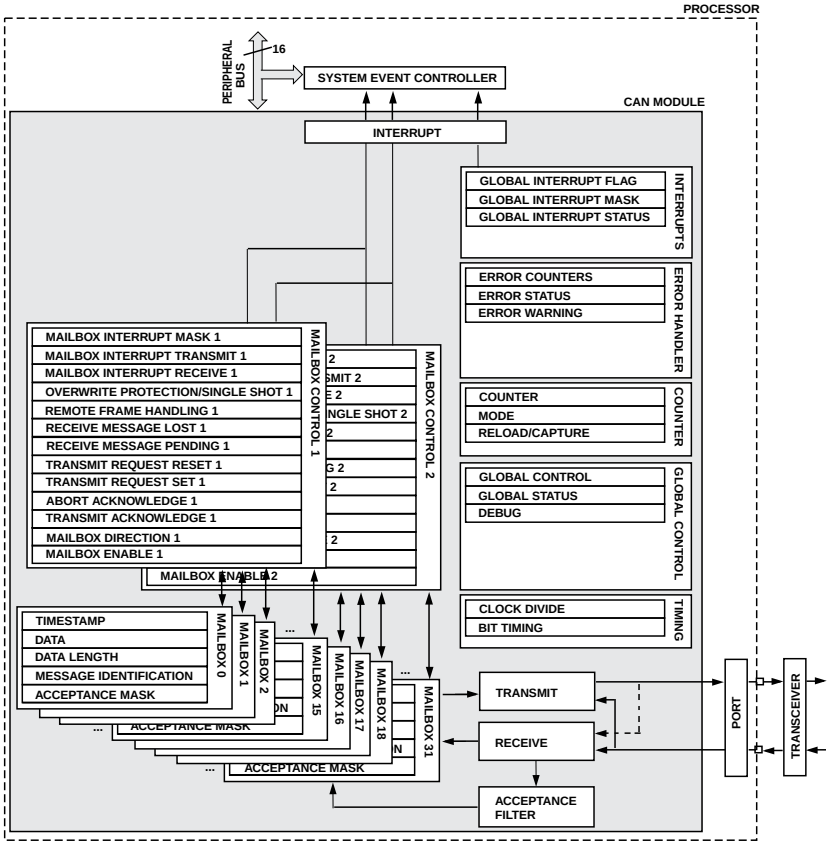

Figure 21-3: CAN Controller Block Diagram

Mailbox Control

Mailbox control memory-mapped registers (MMRs) function as control and status registers for the 32 mailboxes. Each bit in these registers represents one specific mailbox. Since CAN MMRs are all 16 bits wide, pairs of registers manage certain functionality for all 32 individual mailboxes. Mailboxes 0–15 are configured or monitored in registers with a suffix of 1. Similarly, mailboxes 16–31 use the same named register with a suffix of 2. For example, the CAN mailbox direction registers (`CAN_MD1` / `CAN_MD2`) control mailboxes. See the ***CAN Mailbox Register Pair*** figure. The ***CAN Register List*** table shows the mailbox control registers.

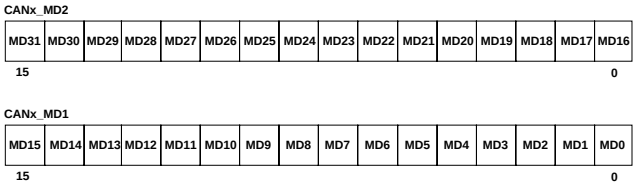

Figure 21-4: CAN Mailbox Register Pair

Mailboxes 24–31 support transmit operation only and mailboxes 0–7 are receive-only mailboxes. Therefore, the lower 8 bits in the 1 registers and the upper 8 bits in the 2 registers are sometimes reserved or are restricted in their use.

Protocol Fundamentals

Although the CAN_RX and CAN_TX pins are TTL-compliant signals, the CAN signals beyond the transceiver have asymmetric drivers. A low state on the CAN_TX pin activates strong drivers while a high state activates weak drivers. So, the active low state is the *dominant* state and the active high state is the *recessive* state. If the CAN module is passive, the CAN_TX pin is always high. If two CAN nodes transmit at the same time, dominant bits overwrite recessive bits.

The CAN protocol specifies that all nodes trying to send a message on the CAN bus attempt to send a frame once the bus is available. (The ***Standard CAN Frame*** figure shows the frame). The start of frame indicator (SOF) signals the beginning of a new frame. Each CAN node then begins transmitting its message starting with the message ID.

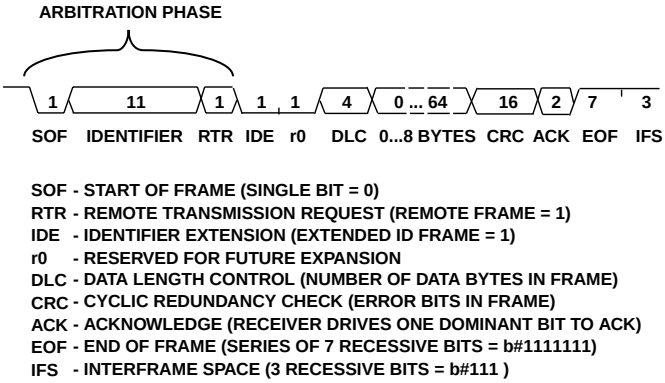

Figure 21-5: Standard CAN Frame

While transmitting, the CAN controller samples the CAN_RX pin to verify that the driven logic level is the value it placed on the CAN_TX pin. The names for the logic levels apply here. When a transmitting node places a recessive 1 on the CAN_TX pin and detects a dominant 0 on the CAN_RX pin, another node has placed a dominant bit on the bus. In this case the dominant bit from from the other node has a higher priority.

Therefore, if the value sensed on the CAN_RX pin is the value driven on the CAN_TX pin, transmission continues. Otherwise, the CAN controller senses that it has lost arbitration. Module configuration determines the next course of action.

The ***Standard CAN Frame*** figure shows a basic 11-bit identifier frame. The CAN_MB[nn]_ID1.RTR bit follows the SOF and identifier. The CAN_MB[nn]_ID1.RTR bit indicates whether the frame contains data (data frame) or is a request for data associated with the message identifier in the frame sent (remote frame).

NOTE: In the CAN protocol, a dominant bit in the CAN_MB[nn]_ID1.RTR field wins arbitration against a remote frame request (CAN_MB[nn]_ID1.RTR=1) for the same message ID. This functionality allows a remote request to be a lower priority than a data frame.

The next field of interest in the frame is the CAN_MB[nn]_ID1.IDE bit. When set, it indicates that the message is an extended frame with a 29-bit identifier instead of an 11-bit identifier. In an extended frame, the first part of the message resembles the following figure.

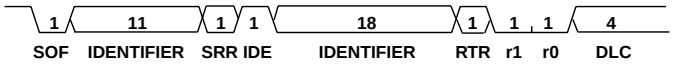

Figure 21-6: Extended CAN Frame

For the CAN_MB[nn]_ID1.RTR field, a dominant bit in the CAN_MB[nn]_ID1.IDE field wins arbitration against an extended frame with the same lower 11 bits. Standard frames are higher priority than extended frames.

The internal logic automatically generates the substitute remote request (SRR), the reserved bits r0 and r1, and the checksum (CRC). (The SRR is always sent as recessive; reserved bits r0 and r1 are always sent as dominant).

Data Transfer Modes

The following sections provide information on the data transfer modes supported by the CAN controller.

Transmit Operations

The *CAN Transmit Operation Flowchart* shows the CAN transmit operation. Mailboxes 24–31 are dedicated transmitters. Configure mailboxes 8–23 as transmitters by writing 0 to the corresponding bit in the CAN_MD1 or CAN_MD2 registers. Enable mailbox n (CAN_MC1.MB=1). After writing the data and the identifier into the mailbox area, the message is sent. Then, the corresponding transmit request bit is set (CAN_TRS1.MB=1).

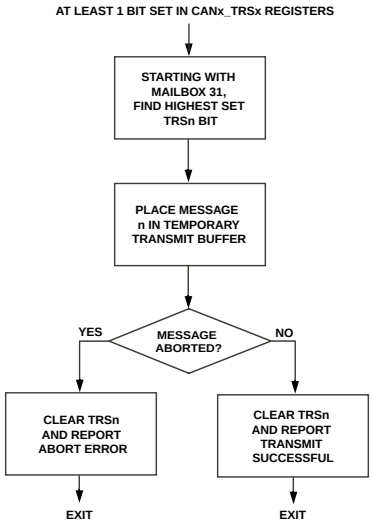

Figure 21-7: CAN Transmit Operation Flowchart

When a transmission completes, the corresponding bits in the CAN_TRS1 or CAN_TRS2 and CAN_TRR1 or CAN_TRR2 registers are cleared. If the transmission is successful, the corresponding bit in the CAN_TA1/CAN_TA2 register is set. If the transmission aborts due to lost arbitration or a CAN error, the corresponding bit in the CAN_AA1/CAN_AA2 register is set. A requested transmission can also be manually aborted by setting the corresponding bit in the CAN_TRR1/CAN_TRR2 register.

The software sets multiple CAN_TRS1.MB bits simultaneously. These bits are reset after either a successful or an aborted transmission.

The CAN hardware sets these bits in the following cases:

- When using the auto-transmit mode of the universal counter,
- When a message loses arbitration and the single-shot CAN_OPSS1.MB bit is not set, or
- When a remote frame request occurs (only possible for receive or transmit mailboxes if the feature for automatic remote frame handling is enabled (CAN_RFH1.MB=1).

NOTE: Manage the mailbox area when a CAN_TRS1 or CAN_TRS2 bit is set. Write access to the mailbox is permissible with a bit set. But, changing data in such a mailbox can lead to unexpected data during transmission.

Enabling and disabling mailboxes has an impact on transmit requests. Setting the CAN_TRS1 or CAN_TRS2 bit associated with a disabled mailbox can result in erroneous behavior. Similarly, disabling a mailbox before the associated CAN_TRS1 or CAN_TRS2 bit is reset by the internal logic can cause unpredictable results.

Retransmission

Normally, the current message object is resent after loss of arbitration or error frame detection on the CAN bus line. If there is more than one transmit message object pending, the message object with the highest mailbox transmits first (See the ***Transmit Flow*** figure). The currently aborted transmission restarts after any messages with higher priority are sent.

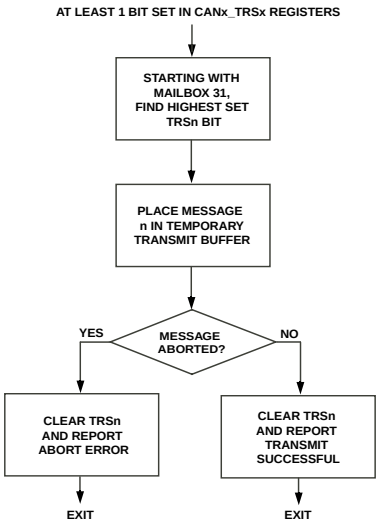

Figure 21-8: Transmit Flow

A message written into the mailbox does not replace a message under preparation.The message under preparation is copied into the temporary transmit buffer when the internal transmit request for the CAN core module is set. The message in the buffer is not replaced until

- The message is sent successfully
- The arbitration on the CAN bus line is lost, or
- There is an error frame on the CAN bus line

Single-Shot Transmission

When using the single-shot transmission feature (CAN_OPSS1.MB=1), the corresponding CAN_TRS1 bit is cleared after the message is successfully sent. The bit is cleared even if the transmission aborts due to a lost arbitration or an error frame on the CAN bus line. Therefore, there is no further attempt to transmit the message again when the initial try failed, and the abort error is reported (CAN_AA1.MB=1).

Auto-Transmission

In auto-transmit mode, the message in mailbox 11 (MB11) can be sent periodically using the universal counter. This mode often broadcasts heartbeats to all CAN nodes. So, messages sent this way usually have high priority.

The period value is written to the CAN_UCRC register. Auto-Transmission mode is enabled by setting the CAN_UCCNF.UCCNF field to 0x03. When enabled, the counter CAN_UCCNT is loaded with the value in the CAN_UCRC register. The counter decrements to 0 at the CAN bit clock rate and is then reloaded from CAN_UCRC. Each time the counter reaches a value of 0, internal logic automatically sends the CAN_TRS1.MB bit. The corresponding message from mailbox 11 transfers.

For proper auto-transmit operation, configure mailbox 11 as a transmit mailbox. The mailbox must contain valid data (identifier, control bits, and data) before the counter expires after this mode is enabled.

Receive Operation

The CAN hardware autonomously receives messages and discards invalid messages. Once a valid message is successfully received, the receive logic interrogates all enabled receive mailboxes. The logic interrogates sequentially, from mailbox 23 down to mailbox 0, whether the message is of interest to the local node or not.

Each incoming data frame is compared to all identifiers stored in active receive and transmit mailboxes with the feature for remote frame handling enabled (=1). (The active receive mailboxes indices of CAN_MD1 and CAN_MC1 registers are set to 1). The message identifier of the received message, along with the identifier extension (CAN_MB[nn]_ID1.IDE) and remote transmission request (CAN_MB[nn]_ID1.RTR) bits, are compared with the register settings of each mailbox. In standard mode, the message is compared with the content of the CAN_MB[nn]_ID1 register. In extended mode, the content of the CAN_MB[nn]_ID0 register must also match.

If the acceptance mask enable CAN_MB[nn]_ID1.AME bit is not set, a match is signaled only if CAN_MB[nn]_ID1.IDE, CAN_MB[nn]_ID1.RTR, and all (11 or 29) identifier bits are exact. If, however, the CAN_MB[nn]_ID1.AME bit is set, the acceptance mask registers (CAN_AM[nn]H/L) determine which of the CAN_MB[nn]_ID1.IDE and CAN_MB[nn]_ID1.RTR bits must match.

The following logic applies:

[(Received Message ID) XNOR (CAN_MB[nn]_ID0/1)] OR [(CAN_MB[nn]_ID1.AME) AND (CAN_AM[nn]H / L)].

This logic appears graphically in the ***CAN Message Receive Logic*** figure.

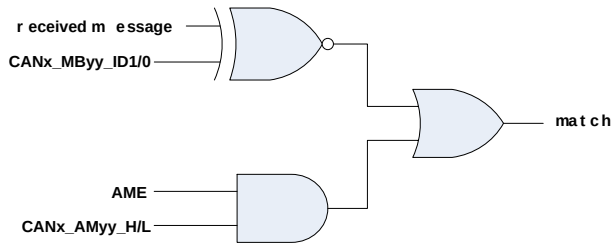

Figure 21-9: CAN Message Receive Logic

A one (1) at the respective bit position in the CAN_AM[nn]H/CAN_AM[nn]L mask registers means that the bit does not need to match when CAN_MB[nn]_ID1.AME=1. This way, a mailbox can accept a group of messages.

Table 21-3: Mailbox Used for Acceptance Filtering

MCn	MDn	RFHn	Mailbox n	Comment
0	X	X	Ignored	Mailbox n disabled
1	0	0	Ignored	Mailbox n enabled, Mailbox n configured for transmit, Remote frame handling disabled
1	0	1	Used	Mailbox n enabled, Mailbox n configured for transmit, Remote frame handling enabled
1	1	X	Used	Mailbox n enabled, Mailbox n configured for receive

If the acceptance filter finds a matching identifier, the content of the received data frame is stored in that mailbox. A received message is stored only once, even if multiple receive mailboxes match its identifier. If the current identifier does not match any mailbox, the message is not stored.

The ***CAN Receive Operation Flowchart*** illustrates the decision tree of the receive logic when processing the individual mailboxes.

If a message is received for a mailbox and that mailbox still contains unread data (CAN_RMP1.MB), then the program decides whether to overwrite the old message. If the CAN_OPSS1.MB bit is cleared, the corresponding CAN_RML1.MB bit is set, and the stored message is overwritten. The receive message lost interrupt occurs (CAN_GIS.RMLIS sets). If, however, the CAN_OPSS1.MB bit is set, the next mailboxes are checked for another matching identifier. If no match is found, the message is discarded, and the next message is checked.

NOTE: If a receive mailbox is disabled, an ongoing receive message for that mailbox is lost even if a second mailbox is configured to receive the same identifier.

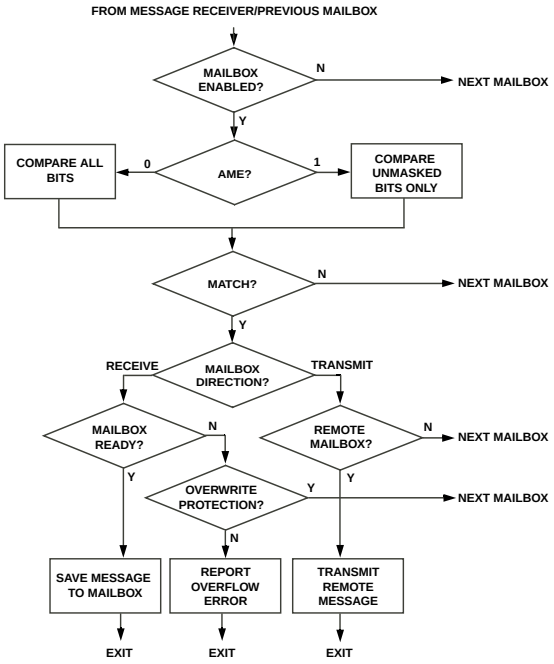

Figure 21-10: CAN Receive Operation Flowchart

Data Acceptance Filtering

If DeviceNet mode is enabled (`CAN_CTL.DNM` = 1) and the mailbox is set-up for filtering on data field, the filtering occurs on the standard ID of the message and data fields. The data field filtering can be programmed for either the first byte only or the first 2 bytes, as shown the ***Data Field Filtering*** table.

If the `CAN_AM[nn]H.FDF` bit is set, the corresponding `CAN_AM[nn]L` register holds the data field mask (DFM bits 15–0]). If the `CAN_AM[nn]H.FDF` bit is cleared, the corresponding `CAN_AM[nn]L` register holds the extended identifier mask (`CAN_AM[nn]H.EXTID` bits 15–0).

Table 21-4: Data Field Filtering

FDF (Filter on Data Field)	FMD (Full Mask Data Field)	Description
0	0	Do not allow filtering on the data field
0	1	Not allowed. FMF must be 0 when FDF is 0
1	0	Filter on first data byte only
1	1	Filter on first two data bytes

Watchdog Mode

Watchdog mode ensures that messages are received periodically. It also observes whether a certain node on the network is alive and functioning properly. Watchdog mode detects and manages the failure cases, as needed.

Enable this mode by programming the universal counter to watchdog mode by setting theCAN_UCCNF.UCCNF to 0x2. Once enabled, the CAN_UCCNT register is loaded with the predefined value contained in CAN_UCRC. This counter decrements at the CAN bit rate.

If the CAN_UCCNF.UCCT and CAN_UCCNF.UCRC bits are set and a message is received in mailbox 4 before the counter counts down to 0, the counter is reloaded with the CAN_UCRC contents. If the counter has counted down to 0 without receiving a message in mailbox 4, then the CAN_GIS.UCEIS bit sets. The counter reloads automatically with the contents of the CAN_UCRC register. If an interrupt is desired for this event, set the CAN_GIM.UCEIM bit. With the mask bit set, when a watchdog interrupt occurs, the CAN_GIF.UCEIF bit also sets.

Write to the CAN_UCCNF register to reload the counter with the contents of CAN_UCRC or to disable the register.

The CAN_UCRC register controls the time period it takes for the watchdog interrupt to occur.

Timestamps

To get an indication of the time of the receive or transmit time for each message, program the CAN universal counter to timestamp mode. Enable this mode by setting the CAN_UCCNF.UCCNF field to 0x01.

If enabled, the value of the 16-bit free-running counter (CAN_UCCNT) is written into the CAN_MB[nn]_TIMESTAMP register of the corresponding mailbox. The operation occurs when a received message is stored or a message is transmitted.

The timestamp value is captured at the sample point of the start of frame (SOF) bit of each incoming or outgoing message. Afterwards, this timestamp value is copied to the CAN_MB[nn]_TIMESTAMP register of the corresponding mailbox.

If the mailbox is configured for automatic remote frame handling (CAN_RFH1.MB = 1), the timestamp value is written for transmission of a data frame or the reception of the requested data frame. (The mailbox is configured for transmit or receive).

Clear the counter by setting the CAN_UCCNF.UCRC bit to 1. Or disable the counter by clearing the CAN_UCCNF.UCE bit. Write to the CAN_UCCNT register to load the counter with a value.

It is also possible to clear the counter (CAN_UCCNT) by reception of a message in mailbox number 4 (synchronization of all time stamp counters in the system). This operation is accomplished by setting the CAN_UCCNF.UCCT bit.

The CAN_GIS.UCEIS bit is set when the counter overflows. A global CAN interrupt can optionally occur by unmasking the CAN_GIM.UCEIM bit. If the interrupt source is unmasked, the CAN_GIF.UCEIF bit is also set.

Remote Frame Handling

Enable automatic handling of remote frames for a transmit mailbox by setting the corresponding CAN_RFH1.MB bit.

Remote frames are data frames with no data field and the CAN_MB[nn]_ID1.RTR bit is set. The data length code (DLC) of the requesting remote frame overrules the DLC of the responding data frame. A DLC can be programmed with values in the range of 0–15, but DLC values greater than 8 are considered as 8.

A remote frame contains:

- The identifier bits
- The control field DLC (data length count)
- The remote transmission request (CAN_MB[nn]_ID1.RTR) bit

Only configurable mailboxes, MB8–MB23, can process remote frames, but all mailboxes can receive and transmit remote frame requests. The CAN_OPSS1 register has no effect when configured for automatic remote frame handling. All content of a mailbox is always overwritten by an incoming message.

NOTE: If a remote frame is received, the DLC of the corresponding mailbox is overwritten with the received value.

Erroneous behavior can result when the CAN_RFH1.MB bit is changed while the corresponding mailbox is processing. To avoid the risk of inconsistent messages, programs must temporarily disable the mailbox while its data registers are updating.

Temporarily Disabling CAN Mailbox

If a mailbox is enabled and configured to transmit, monitor the write accesses to the data field to avoid transmitting inconsistent messages. Be careful if the mailbox is transmitting (or attempting to transmit) repeatedly. Also, if this mailbox is used for automatic remote frame handling, the data field must be updated without losing an incoming remote request frame and without sending inconsistent data. Therefore, the CAN controller allows for temporary disabling the mailbox using the mailbox temporary disable register (CAN_MBTD).

Write the pointer to the requested mailbox to the CAN_MBTD.TDPTR field, and set the CAN_MBTD.TDR bit. Internal logic then sets the corresponding CAN_MBTD.TDA flag.

If a mailbox is configured as transmit (CAN_MD1 = 0) and the CAN_MBTD.TDA bit is set, the content of the data field of that mailbox can be updated. If there is an incoming remote request frame while the mailbox is temporarily disabled,

- Internal logic sets the corresponding transmit request bit (CAN_TRS1.MB), and
- The data length code (DLC) of the incoming message is written to the corresponding mailbox.

However, the requested message is not sent until the CAN_MBTD.TDR bit is cleared. Similarly, all transmit requests for temporarily disabled mailboxes are ignored until the CAN_MBTD.TDR bit is cleared. Additionally, transmission of a message immediately aborts when the mailbox is temporarily disabled and the corresponding transmission request reset (CAN_TRR1.MB) bit for this mailbox sets.

If a mailbox is configured to receive (CAN_MD1 = 1), then after issuing a temporary disable request, the CAN_MBTD.TDA flag sets. The mailbox is not processed. If there is an incoming message for a temporarily disabled mailbox, the internal logic waits until reception is complete or there is an error on the CAN bus before setting CAN_MBTD.TDA. Once this flag sets, the mailbox can then be disabled (CAN_MC1 = 0) without the risk of losing an incoming frame. The CAN_MBTD.TDR bit must then be reset as soon as possible.

When the CAN_MBTD.TDA flag sets for a given mailbox, only the data field of that mailbox can be updated. Accesses to the control bits and the identifier are denied.

CAN Operating Modes

The CAN controller is in configuration mode when coming out of processor reset. Hardware behavior can be altered only when CAN is in configuration mode. Before initializing the mailboxes, configure the CAN bit timing to work on the CAN bus. The controller connect to the CAN bus.

Bit Timing

The CAN controller does not have a dedicated clock. Instead, the CAN clock is derived from the system clock based on a configurable number of time quanta. The time quantum (TQ) is derived from the formula:

TQ = (BRP + 1)/SYSCLK,

where BRP is the 10-bit bit rate prescaler field in the CAN_CLK register.

Although the CAN_CLK.BRP field can be set to any value, it is recommended that the value be greater than or equal to 4. Restrictions apply to the bit timing configuration when BRP is less than 4.

The CAN_CLK register defines the TQ value, and multiple time quanta make up the duration of a CAN bit on the bus. The CAN_TIMING register controls the nominal bit time and the sample point of the individual bits in the CAN protocol. The ***Three Phases of a CAN Bit*** figure shows the three phases of a CAN bit: the synchronization segment, the segment before the sample point, and the segment after the sample point.

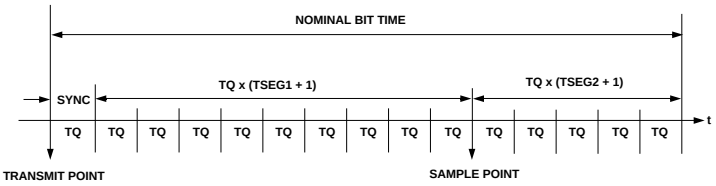

Figure 21-11: Three Phases of a CAN Bit

The synchronization segment is fixed to one TQ. Synchronize the nodes on the bus. All signal edges are expected to occur within this segment.

The CAN_TIMING.TSEG1 and CAN_TIMING.TSEG2 fields control how many TQs the CAN bits consist of, resulting in the CAN bit rate. The following formula gives the nominal bit time.

t_{BIT} = TQ × [1 + (1 + TSEG1) + (1 + TSEG2)]

For safe receive operations on given physical networks, the sample point is programmable by the CAN_TIMING.TSEG1 field. The CAN_TIMING.TSEG2 field holds the number of TQs to complete the bit time. Often, best sample reliability is achieved with sample points in the high 80% range of the bit time. Never use sample points lower than 50%. Therefore, CAN_TIMING.TSEG1 must always be greater than or equal to CAN_TIMING.TSEG2.

The CAN module does not distinguish between the propagation segment and the phase segment-1 as defined by the standard. The CAN_TIMING.TSEG1 value is intended to cover both of them. The CAN_TIMING.TSEG2 value represents the phase segment-2.

If the CAN module detects a recessive-to-dominant edge outside the synchronization segment, it can automatically move the sampling point such that the CAN bit is still handled properly. The synchronization jump width (CAN_TIMING.SJW) field specifies the maximum number of TQs, ranging from 1 to 4 (SJW + 1), allowed for such a resynchronization attempt. The SJW value must not exceed CAN_TIMING.TSEG2 or CAN_TIMING.TSEG1. Therefore, the fundamental rule for writing CAN_TIMING is:

SJW ≤ TSEG2 ≤ TSEG1

In addition to this fundamental rule, CAN_TIMING.TSEG2 must also be greater than or equal to the information processing time (IPT). IPT is the time required by the logic to sample the CAN_RX input, which is 3 system clock cycles.

Therefore, restrictions apply to the minimal value of CAN_TIMING.TSEG2 if CAN_CLK.BRP is less than 2. If CAN_CLK.BRP is set to 0, the CAN_TIMING.TSEG2 field must be greater than or equal to 2. If CAN_CLK.BRP is set to 1, the minimum CAN_TIMING.TSEG2 value is 1.

NOTE: Use the same nominal bit rate for all nodes on a CAN bus.

With all the timing parameters set, the final consideration is sampling performance. The default behavior of the CAN controller is to sample the CAN bit once. The controller samples at the point described by the CAN_TIMING register and controlled by the CAN_TIMING.SAM bit. If this bit is set, however, the input signal is oversampled three times at the system clock rate. The resulting value is generated by a majority decision of the three sample values. Always keep the CAN_TIMING.SAM bit cleared if the BRP value is less than 4.

Do not modify the CAN_CLK and CAN_TIMING registers during normal operation. Always enter configuration mode first. Writes to these registers have no effect when CAN is not in configuration or debug mode. If not coming out of processor reset, enter configuration mode by setting the CAN_CTL.CCR bit and poll the CAN_STAT register until CAN_STAT.CCA is set.

NOTE: If the CAN_TIMING.TSEG1 field is programmed to 0, the module does not leave the configuration mode.

During configuration mode, the module is not active on the CAN bus line. The CAN_TX output pin remains recessive and the module does not receive or transmit messages or error frames. After leaving the configuration mode, all CAN internal core registers and the CAN error counters are set to their initial values.

A soft reset does not change the values of CAN_CLK and CAN_TIMING. Therefore, an ongoing transfer through the CAN bus cannot be corrupted by changing the bit timing parameter or initiating the soft reset (by setting the CAN_CTL.SRS bit).

CAN Low Power Features

The CAN module includes built-in sleep and suspend modes to save power.

The following sections describe the behavior of the CAN module in these modes.

Built-In Suspend Mode

The most modest of power savings mode is the suspend mode. This mode is entered by setting the CAN_CTL.CSR bit. The module enters the suspend mode after the current operation of the CAN bus finishes. Then, the internal logic sets the CAN_STAT.CSA bit. Once CAN enters this mode, the module is no longer active on the CAN bus line, slightly reducing power consumption.

In suspend mode, the CAN_TX output pin remains in a recessive state, and the module does not receive or transmit messages or error frames. The content of the CAN_CEC register remains unchanged. Clear CAN_CTL.CSR to exit suspend mode.

The only difference between suspend mode and configuration mode is that the CAN_CTL and CAN_STAT registers are not reset when exiting suspend mode.

Built-In Sleep Mode

The next level of power savings can be realized by using the built-in sleep mode for the module. This mode is entered by setting the CAN_CTL.SMR bit. The module enters the sleep mode after the current operation of the CAN bus finishes. Once this mode is entered, many of the internal CAN module clocks are shut off, reducing power consumption, and the CAN_INT.SMACK bit sets.

When the CAN module is in sleep mode, all register reads return the contents of CAN_INT instead of the usual contents. All register writes, except to CAN_INT, are ignored in sleep mode. A small part of the module is clocked continuously to allow for wake-up out of sleep mode.

A write to the CAN_INT register ends sleep mode. If the CAN_CTL.WBA bit is set before entering sleep mode, a dominant bit on the CAN_RX pin also ends sleep mode. When software sets the CAN_CTL.SMR bit, hardware sets the CAN_CTL.CSR bit as well, making sleep mode a super set of suspend mode. When the controller wakes up from sleep mode, hardware automatically clears CAN_CTL.SMR and CAN_CTL.CSR. If, however, the controller never enters sleep mode because the wake-up condition was met before CAN_INT.SMACK bit turns to 1, the CAN_CTL.SMR and CAN_CTL.CSR bits do not always automatically clear. Therefore, clear the two bits using software, when returning from sleep mode.

Soft Reset

The CAN controller features a build-in reset mechanism called soft reset. Soft reset is entered immediately after software has set the CAN_CTL.SRS bit. Soft reset brings all control registers to a defined state. Mailbox and error registers remain unaffected. Soft reset does not alter the CAN_TIMING and CAN_CLK registers and does not disturb the on-going transmission of a currently pending message, acknowledge bit or error frame. However, when recovering from soft reset, software can lose track of transmission or reception reports and interrupts.

CAN Event Control

The following is a description of how CAN generates and controls events.

CAN Interrupt Signals

The CAN module provides three independent interrupts: two mailbox interrupts (mailbox receive interrupt (MBRIRQ) and mailbox transmit interrupt (MBTIRQ)) and a global CAN status interrupt (GIRQ). The values of these three interrupts can also be read through the CAN_GIS registers.

Mailbox Interrupts

Each of the 32 mailboxes in the CAN module can generate a receive or transmit interrupt, depending on the mailbox configuration. To enable a mailbox to generate an interrupt, set the corresponding CAN_MBIM1 bit.

If a mailbox is configured as a receive mailbox, the corresponding CAN_MBRIF1 bit and CAN_RMP1 bit are set after a received message is stored in mailbox n. When using the feature for automatic remote frame handling (CAN_RFH1=1), the receive interrupt flag is set after the requested data frame is stored in the mailbox.

If any CAN_MBRIF1 bits are set, the mailbox generates a CAN_INT.MBRIRQ interrupt. To clear the CAN_INT.MBRIRQ interrupt request, software must clear all of the set CAN_MBRIF1 bits by writing a 1 to those bit locations in CAN_MBRIF1. Prior to this operation, software must clear the corresponding CAN_RMP1 bit.

If a mailbox is configured as a transmit mailbox, the corresponding CAN_MBTIF1 bit in the transmit interrupt flag is set after the message in mailbox n is sent correctly. The corresponding CAN_TA1 bit is also set. The CAN_TA1 bits maintain their state even after the corresponding mailbox n is disabled (CAN_MC1=0). When using the feature for automatic remote frame handling, the transmit interrupt flag is set after the requested data frame is sent from the mailbox.

If any CAN_MBTIF1.MB bits are set, the MBTIRQ interrupt output is raised in the CAN_INT register. To clear the MBTIRQ interrupt request, software must clear all of the bits that are set in the CAN_MBTIF1 register by writing a 1 to those bit locations. Additionally, software must clear the associated CAN_TA1 bit or set the associated CAN_TRS1 bit to clear the interrupt source that asserts the CAN_MBTIF1 bit.

Global Interrupt

The global CAN interrupt logic implements with three registers:

- The CAN_GIM register, where each interrupt source can be enabled or disabled separately
- The CAN_GIS register
- The CAN_GIF register

The interrupt mask bits only affect the content of the CAN_GIF register. If the mask bit is not set in the CAN_GIM register, the corresponding flag bit is not set when the event occurs. The interrupt status bits in the CAN_GIS register, however, are always set when the corresponding interrupt event occurs, independent of the mask bits. Thus, the interrupt status bits can be used to poll interrupt events.

The CAN_INT.GIRQ bit is only asserted if a bit in the CAN_GIF register is set. The read-only CAN_INT.GIRQ bit remains set as long as at least 1 bit in CAN_GIF is set. All bits in the interrupt status and interrupt flag registers remain set until cleared by software or a soft reset has occurred.

NOTE: The CAN_GIF register is read-only (RO). In the global CAN interrupt ISR, clear the interrupt latch by writing to 1 to the corresponding bit of the CAN_GIS register. The operations clear the related bits of the CAN_GIS and CAN_GIF registers, as well as the CAN_INT.GIRQ bit.

There are several interrupt events that can activate this GIRQ interrupt:

- Access denied interrupt (CAN_GIM.ADIM, CAN_GIS.ADIS, CAN_GIF.ADIF): At least one access to the mailbox RAM occurred during a data update by internal logic.
- Universal counter exceeded interrupt (CAN_GIM.UCEIM, CAN_GIS.UCEIS, CAN_GIF.UCEIF): There is an overflow of the universal counter (in timestamp mode or event counter mode) or the counter has reached the value 0x0000 (in watchdog mode).
- Receive message lost interrupt (CAN_GIM.RMLIM, CAN_GIS.RMLIS, CAN_GIF.RMLIF): A message is received for a mailbox that currently contains unread data. At least 1 bit in the CAN_RMLn register is set. If the bit in CAN_GIS and CAN_GIF registers is cleared and there is at least 1 bit in CAN_RML1 still set, then the bit in the CAN_GIS and CAN_GIF registers is not set again. The internal interrupt source signal is only active if a new bit in CAN_RML1 is set.
- Abort acknowledge interrupt (CAN_GIM.AAIM, CAN_GIS.AAIS, CAN_GIF.AAIF): At least 1 CAN_AA1.MB bit in the CAN_AA1 registers is set. If the bit in the CAN_GIS and CAN_GIF registers is cleared and there is at least 1 bit in CAN_AA1 still set, then the bit in the CAN_GIS and CAN_GIF registers is not set again. The internal interrupt source signal is only active if a new bit in CAN_AA1 is set. The CAN_AA1.MB bits maintain state even after the corresponding mailbox n is disabled (CAN_MC1 = 0).
- Access to unimplemented address interrupt (CAN_GIM.UIAIM, CAN_GIS.UIAIS, CAN_GIF.UIAIF): There was a CPU access to an address which is not implemented in the controller module.
- Wake-up interrupt (CAN_GIM.WUIM, CAN_GIS.WUIS, CAN_GIF.WUIF): The CAN module has left the sleep mode because of detected activity on the CAN bus line.
- Bus-off interrupt (CAN_GIM.BOIM, CAN_GIS.BOIS, CAN_GIF.BOIF): The CAN module has entered the bus-off state. This interrupt source is active if the status of the CAN core changes from normal operation mode to the bus-off mode. If the bit in the CAN_GIS and CAN_GIF registers is cleared and the bus-off mode is still active, then this bit is not set again. If the module leaves the bus-off mode, the bit in the CAN_GIS and CAN_GIF registers remains set, if not explicitly cleared.
- Error-passive interrupt (CAN_GIM.EPIM, CAN_GIS.EPIS, CAN_GIF.EPIF): The CAN module has entered the error-passive state. This interrupt source is active if the status of the CAN module changes from the error-active mode to the error-passive mode. If the bit in the CAN_GIS and CAN_GIF registers is cleared and the error-passive mode is still active, then this bit is not set again. If the module leaves the error-passive mode, the bit in the CAN_GIS and CAN_GIF registers remains set, if not explicitly cleared.
- Error warning receive interrupt (CAN_GIM.EWRIM, CAN_GIS.EWRIS, CAN_GIF.EWRIF): The CAN receive error counter (CAN_CEC.RXECNT) has reached the warning limit. If the bit in the CAN_GIS and CAN_GIF registers) is cleared and the error warning mode is still active, this bit is not set again. If the module leaves the error warning mode, the bit in the CAN_GIS and CAN_GIF registers remains set, if not explicitly cleared.

- Error warning transmit interrupt (CAN_GIM.EWTIM, CAN_GIS.EWTIS, CAN_GIF.EWTIF): The CAN transmit error counter (CAN_CEC.TXECNT) has reached the warning limit. If the bit in the CAN_GIS and CAN_GIF registers is cleared and the error warning mode is still active, this bit is not set again. If the module leaves the error warning mode, the bit in the CAN_GIS and CAN_GIF registers remains set, if not explicitly cleared.

Event Counter

For diagnostic functions, it is possible to use the universal counter as an event counter. The counter can be programmed in the CAN_UCCNF[3:0] field to increment on one of these conditions:

- 0x6 – CAN error frame. Counter increments if there is an error frame on the CAN bus line.
- 0x7 – CAN overload frame. Counter increments if there is an overload frame on the CAN bus line.
- 0x8 – Lost arbitration. Counter increments every time arbitration on the CAN line is lost during transmission.
- 0x9 – Transmission aborted. Counter increments every time arbitration is lost or a transmit request is canceled (CAN_AA1 sets).
- 0xA – Transmission succeeded. Counter increments every time a message sends without detected errors (CAN_TA1 sets).
- 0xB – Receive message rejected. Counter increments every time a message is received without detected errors but not stored in a mailbox because there is no matching identifier found.
- 0xC – Receive message lost. Counter increments every time a message is received without detected errors but not stored in a mailbox because the mailbox contains unread data (CAN_RML1 sets).
- 0xD – Message received. Counter increments every time a message is received without detected errors, whether the received message is rejected or stored in a mailbox.
- 0xE – Message stored. Counter increments every time a message is received without detected errors, has an identifier that matches an enabled receive mailbox, and is stored in the receive mailbox (CAN_RMP1 sets).
- 0xF – Valid message. Counter increments every time a valid transmit or receive message is detected on the CAN bus line.

CAN Warnings and Errors

The processor controls CAN warnings and errors using the error counter (CAN_CEC) register, the error status (CAN_ESR) register, and the error counter warning level (CAN_EWR) register. The following sections describe error handling.

Programmable Warning Limits

Programs can set the warning level for CAN_GIS.EWTIS and CAN_GIS.EWRIS separately by writing to the CAN_EWR.EWLREC and CAN_EWR.EWLTEC fields. After power-on reset, the CAN_EWR register is set to the default warning level of 96 for both error counters. After a soft reset, the contents of this register remain unchanged.

Error Handling

Error management is a part of the CAN standard. Several different kinds of bus errors can occur during transmissions:

- Bit error – Only the transmitting node detects this error. Whenever a node transmits, it continuously monitors its receive pin (CAN_RX) and compares the received bit with the transmitted bit. During the arbitration phase, the node postpones the transmission if the received and transmitted bits do not match. However, after the arbitration phase, a bit error is signaled any time the value on CAN_RX does not equal what is transmitted on the CAN_TX pin. (The arbitration phase completes when the CAN_MB[nn]_ID1.RTR bit is sent successfully).
- Form error. Occurs when a fixed-form bit position in the CAN frame contains one or more illegal bits. Occurs when a dominant bit is detected at a delimiter or end-of-frame bit position.
- Acknowledge error. Occurs whenever a message is sent and no receivers drive an acknowledge bit.
- CRC error. Occurs whenever a receiver calculates the CRC on the data it received and finds it different than the CRC that transmitted on the bus itself.
- Stuff error. The CAN specification requires the transmitter to insert an extra stuff bit of opposite value after 5 bits have transmitted with the same value. The receiver disregards the value of the stuff bits. However, it takes advantage of the signal edge to resynchronize itself. A stuff error occurs on receiving nodes whenever the sixth consecutive bit value is the same as the previous 5 bits.

Once the CAN module detects any of the errors, it updates the CAN_ESR and CAN_CEC registers. In addition to the standard errors, the CAN_ESR.SAO flag signals when the CAN_RX pin sticks at dominant level, indicating a possibility of shorted wires.

Error Frames

It is important that all nodes on the CAN bus ignore data frames that any single node failed to receive. Every node sends an error frame as soon as it has detected an error as shown in the ***CAN Error Example*** figure.

A device that has detected an error still completes the ongoing bit. It initiates an error frame by sending six dominant and eight recessive bits to the bus. Since this activity is a violation of the bit stuffing rule, all nodes are signaled to discard the ongoing frame. (All receivers that did not detect the transmission error in the first instance now detect a stuff bit error.)

The transmitter can detect a normal bit error sooner. It aborts the transmission of the ongoing frame and tries resending it later.

When all nodes on the bus have detected the error, they also send 6 dominant and 8 recessive bits to the bus. The resulting error frame consists of two different fields. The first field is the superposition of error flags contributed from the different stations, which are a sequence of 6–12 dominant bits. The second field is the error delimiter and consists of 8 recessive bits indicating the end of frame.

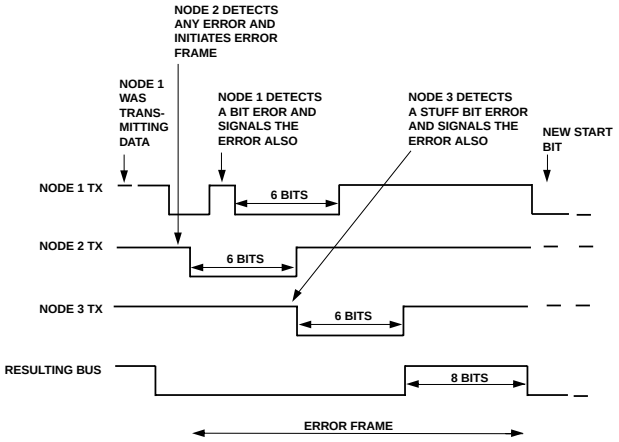

Figure 21-12: CAN Error Example

For CRC errors, the error frame initiates at the end of the frame, rather than immediately after the failing bit.

After having received 8 recessive bits, every node knows that the error condition is resolved and, if messages are pending, starts transmission. The transmitter that had to abort its operation must win the new arbitration again; otherwise its message is delayed as determined by priority.

Because the transmission of an error frame destroys the frame under transmission, a faulty node erroneously detecting an error can block the bus. So, there are two node states which determine a nodes right to signal an error—error-active and error-passive.

- *Error-active* nodes have an error detection rate below a certain limit. These nodes drive an active error flag of 6 dominant bits.
- *Error-passive* nodes have a higher error detection rate and can have a local problem and therefore have a limited right to signal errors. These nodes drive a passive error flag consisting of 6 recessive bits. Therefore, an error-passive transmitting node is still able to inform the other nodes about the aborting of a self-transmitted frame. But, it is no longer able to destroy correctly received frames of other nodes.

Error Levels

The CAN specification requires each node in the system to operate in one of three levels. The ***CAN Error Level Description*** table describes the levels. This functionality prevents nodes with high error rates from blocking the entire network, as local hardware can cause the errors. The CAN module provides an error counter for transmit (TEC) and an error counter for receive (REC). The `CAN_CEC` register contains each of these 8-bit counters.

After initialization, both the TEC and the REC counters are 0. Each time a bus error occurs, one of the counters increments by either 1 or 8, depending on the error situation. (Refer to version 2.0 of the CAN specification). Successful transmit or receive operations decrement the respective counter by 1.

If either of the error counters exceeds 127, the CAN module goes into an error-passive state and the `CAN_STAT.EP` bit is set. Once this state occurs, the module is not allowed to send any more active error frames. However, the module can still transmit messages and signal passive error frames in case the transmission fails due to bit errors.

If one of the counters exceeds 255 (that is, when an 8-bit counter overflows), the CAN module disconnects from the bus and it goes into bus-off mode. In this mode, the CAN_STAT.EBO bit is set. Software intervention is needed for recovery from this state, unless the CAN_CTL.ABO bit is enabled. The bit puts the module into active mode after the bus-off recovery sequence.

Table 21-5: CAN Error Level Description

Level	Condition	Description
Error active	Transmit and receive error counters <128	This level is the initial condition level. As long as errors stay below 128, the node drives active error flags during error frames.
Error passive	Transmit or receive error counter-value from 128 through 255, inclusive	Errors have accumulated to a level that requires the node to drive passive error flags during error frames.
Bus off	Transmit or receive error counters greater than 255	CAN module goes into bus-off mode

In addition to the three levels in the table, the CAN module also generates separate transmit and receive warnings (CAN specification enhancement). By default, when one of the error counters exceeds 96, it signals and reports a warning in the CAN_STAT register. The CAN receive warning flag (CAN_STAT.WR) bit is set when CAN_CEC.RXECNT exceeds 96. The CAN transmit warning flag (CAN_STAT.WT) bit is set when CAN_CEC.TXECNT exceeds 96. The error warning level can be programmed using the error warning register (CAN_EWR).

Additionally, interrupts can occur for all of these levels by unmasking them in the global CAN interrupt mask register (CAN_GIM). These interrupts include: the bus-off interrupt (CAN_GIM.BOIM), the error-passive interrupt (CAN_GIM.EPIM), the error warning receive interrupt (CAN_GIM.EWRIM), and the error warning transmit interrupt (CAN_GIM.EWTIM).

During the bus-off recovery sequence, internal logic sets the configuration mode request CAN_CTL.CCR bit. The CAN core module does not automatically come out of the bus-off mode. The CAN_CTL.CCR bit cannot be reset until the bus-off recovery sequence completes.

NOTE: Set the CAN_CTL.ABO bit to override this behavior. After exiting the bus-off or configuration modes, the CAN error counters are reset.

CAN Debug and Test Modes

The CAN module contains test mode features that aid in the debugging of the CAN software and system.

NOTE: When using these features, the CAN module does not always comply to the CAN specification. Enable or disable all test modes only when the module is in configuration mode (CAN_STAT.CCA=1) or suspend mode (CAN_STAT.CSA=1).

The CAN_DBG.CDE bit provides access to all of the debug features. Set this bit to enable the test mode. Write to the bit first, before writing to the CAN_DBG register. When the CAN_DBG.CDE bit is cleared, all debug features are disabled.

When the CAN_DBG.CDE bit is set, it enables writes to the other bits of the CAN_DBG register. It also enables these features, which are not compliant to the CAN standard:

- Bit timing registers can be changed anytime, not only during configuration mode. The group includes the CAN_CLK and CAN_TIMING registers.
- Write-access is allowed to the normally read-only CAN_CEC register.

The following list describes other bits in the debug register.

- The CAN module uses the CAN_DBG.MRB bit to enable the readback mode. In this mode, a message transmitted on the CAN bus (or through an internal loopback mode) is received back directly to the internal receive buffer. After a correct transmission, the internal logic treats this transfer as a normal receive message. This feature allows testing of most of the CAN features without an external device.
- The CAN_DBG.MAA bit allows the CAN module to generate its own acknowledge during the ACK slot of the CAN frame. No external devices or connections are necessary to readback a transmit message. In this mode, the sent message is automatically stored in the internal receive buffer. In auto-acknowledge mode, the module itself transmits the acknowledge. This acknowledge can be programmed to appear on the CAN_TX pin, if CAN_DBG.DIL=1 and CAN_DBG.DTO= 0. If the acknowledge is only used internally, then set these test mode bits to CAN_DBG.DIL= 0 and CAN_DBG.DTO=1.
- The CAN module uses the CAN_DBG.DIL bit to internally enable the transmit output to be routed back to the receive input.
- The CAN module uses the CAN_DBG.DTO bit to disable the CAN_TX output pin. When this bit is set, the CAN_TX pin continuously drives recessive bits.
- The CAN module uses the CAN_DBG.DRI bit to disable the CAN_RX input. When set, the internal logic receives recessive bits or receives the internally generated transmit value in the case of the internal loop enabled (CAN_DBG.DIL= 0). In either case, the value on the CAN_RX input pin is ignored.
- The CAN module uses the CAN_DBG.DEC bit to disable the transmit and receive error counters in the CAN_CEC register. When this bit is set, the CAN_CEC holds its current contents and is not allowed to increment or decrement the error counters. This mode does not conform to the CAN specification.

NOTE: Write to the error counter-registers in debug mode only. Write-access during reception can lead to undefined values. The maximum value which can be written into the error counters is 255. Therefore, the error counter-value of 256, which forces the module into the bus off state, cannot be written into the error counter-registers.

Table 21-6: Common CAN Test Mode Bit Combinations

MRB	MAA	DIL	DTO	DRI	CDE	Functional Description
X	X	X	X	X	0	Normal mode, not debug mode
0	X	X	X	X	X	No readback of transmit message
1	0	1	0	0	1	Normal transmission on CAN bus line.

Table 21-6: Common CAN Test Mode Bit Combinations (Continued)

MRB	MAA	DIL	DTO	DRI	CDE	Functional Description
						Read back. External acknowledge from external device required.
1	1	1	0	0	1	Normal transmission on CAN bus line. Read back. No external acknowledge required. Transmit message and acknowledge are transmitted on CAN bus line. CAN_RX input is enabled.
1	1	0	0	0	1	Normal transmission on CAN bus line. Read back. No external acknowledge required. Transmit message and acknowledge transmit on CAN bus line. CAN_RX input and internal loop are enabled (internal OR of TX and RX)
1	1	0	0	1	1	Normal transmission on CAN bus line. Read back. No external acknowledge required. Transmit message and acknowledge are transmitted on CAN bus line. CAN_RX input is ignored. Internal loop is enabled.
1	1	0	1	1	1	No transmission on CAN bus line. Read back. No external acknowledge required. Nether transmit message nor acknowledge are transmitted on CAN_TX. CAN_RX input is ignored. Internal loop is enabled.

ADSP-CM40x CAN Register Descriptions

Controller Area Network (CAN) contains the following registers.

Table 21-7: ADSP-CM40x CAN Register List

Name	Description
CAN_AA1	Abort Acknowledge 1 Register
CAN_AA2	Abort Acknowledge 2 Register
CAN_AM[nn]H	Acceptance Mask (H) Register

Table 21-7: ADSP-CM40x CAN Register List (Continued)

Name	Description
CAN_AM[nn]L	Acceptance Mask (L) Register
CAN_CEC	Error Counter Register
CAN_CLK	Clock Register
CAN_CTL	CAN Master Control Register
CAN_DBG	Debug Register
CAN_ESR	Error Status Register
CAN_EWR	Error Counter Warning Level Register
CAN_GIF	Global CAN Interrupt Flag Register
CAN_GIM	Global CAN Interrupt Mask Register
CAN_GIS	Global CAN Interrupt Status Register
CAN_INT	Interrupt Pending Register
CAN_MBIM1	Mailbox Interrupt Mask 1 Register
CAN_MBIM2	Mailbox Interrupt Mask 2 Register
CAN_MBRIF1	Mailbox Receive Interrupt Flag 1 Register
CAN_MBRIF2	Mailbox Receive Interrupt Flag 2 Register
CAN_MBTD	Temporary Mailbox Disable Register
CAN_MBTIF1	Mailbox Transmit Interrupt Flag 1 Register
CAN_MBTIF2	Mailbox Transmit Interrupt Flag 2 Register
CAN_MB[nn]_DATA0	Mailbox Word 0 Register
CAN_MB[nn]_DATA1	Mailbox Word 1 Register
CAN_MB[nn]_DATA2	Mailbox Word 2 Register
CAN_MB[nn]_DATA3	Mailbox Word 3 Register
CAN_MB[nn]_ID0	Mailbox ID 0 Register
CAN_MB[nn]_ID1	Mailbox ID 1 Register
CAN_MB[nn]_LENGTH	Mailbox Length Register
CAN_MB[nn]_TIMESTAMP	Mailbox Timestamp Register
CAN_MC1	Mailbox Configuration 1 Register
CAN_MC2	Mailbox Configuration 2 Register
CAN_MD1	Mailbox Direction 1 Register
CAN_MD2	Mailbox Direction 2 Register
CAN_OPSS1	Overwrite Protection/Single Shot Transmission 1 Register

Table 21-7: ADSP-CM40x CAN Register List (Continued)

Name	Description
CAN_OPSS2	Overwrite Protection/Single Shot Transmission 2 Register
CAN_RFH1	Remote Frame Handling 1 Register
CAN_RFH2	Remote Frame Handling 2 Register
CAN_RML1	Receive Message Lost 1 Register
CAN_RML2	Receive Message Lost 2 Register
CAN_RMP1	Receive Message Pending 1 Register
CAN_RMP2	Receive Message Pending 2 Register
CAN_STAT	Status Register
CAN_TA1	Transmission Acknowledge 1 Register
CAN_TA2	Transmission Acknowledge 2 Register
CAN_TIMING	Timing Register
CAN_TRR1	Transmission Request Reset 1 Register
CAN_TRR2	Transmission Request Reset 2 Register
CAN_TRS1	Transmission Request Set 1 Register
CAN_TRS2	Transmission Request Set 2 Register
CAN_UCCNF	Universal Counter Configuration Mode Register
CAN_UCCNT	Universal Counter Register
CAN_UCRC	Universal Counter Reload/Capture Register

Abort Acknowledge 1 Register

The CAN_AA1 register indicates transmission abort (due to lost arbitration or a CAN error) for mailboxes 8 through 15. Each bit in this register indicates transmission abort for the corresponding mailbox when set (=1). Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

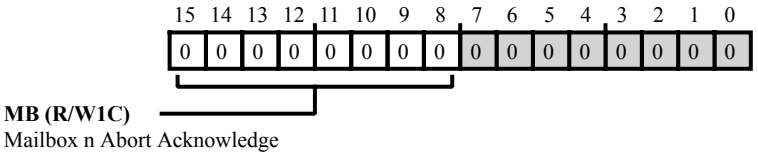

Figure 21-13: CAN_AA1 Register Diagram

Table 21-8: CAN_AA1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W1C)	MB	Mailbox n Abort Acknowledge.

Abort Acknowledge 2 Register

The CAN_AA2 register indicates transmission abort (due to lost arbitration or a CAN error) for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register indicates transmission abort for the corresponding mailbox when set (=1).

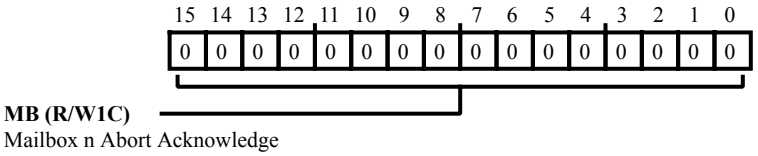

Figure 21-14: CAN_AA2 Register Diagram

Table 21-9: CAN_AA2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Abort Acknowledge.

Acceptance Mask (H) Register

The CAN_AM[nn]H register and CAN_AM[nn]L register manage acceptance mask operation. For information about acceptance mask operation, see the Receive Operation section.

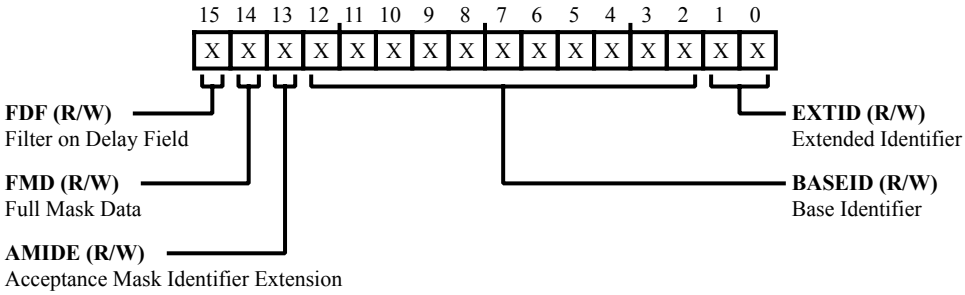

Figure 21-15: CAN_AM[nn]H Register Diagram

Table 21-10: CAN_AM[nn]H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	FDF	Filter on Delay Field. The CAN_AM[nn]H.FDF bit selects the operation of the CAN_AM[nn]H register and CAN_AM[nn]L register when the CAN_CTL.DNM bit is enabled. If the CAN_AM[nn]H.FDF bit is set, the corresponding CAN_AM[nn]L.EXTID bits hold the data field mask. If the CAN_AM[nn]H.FDF bit is cleared, the corresponding CAN_AM[nn]L.EXTID bits hold the high bits of the extended identifier mask.
14 (R/W)	FMD	Full Mask Data. The CAN_AM[nn]H.FMD bit works with the CAN_AM[nn]H.FDF bit to determine data field filtering. For information about data field filtering, see the Receive Operation section.
13 (R/W)	AMIDE	Acceptance Mask Identifier Extension. The CAN_AM[nn]H.AMIDE bit enables the comparison of the received message ID to the value in the CAN_AM[nn]H.EXTID and CAN_AM[nn]L.EXTID bits.
12:2 (R/W)	BASEID	Base Identifier. The CAN_AM[nn]H.BASEID bits hold the base ID for acceptance mask operations.
1:0 (R/W)	EXTID	Extended Identifier. The CAN_AM[nn]H.EXTID bits hold the extended ID (upper two bits) for acceptance mask operations.

Acceptance Mask (L) Register

The CAN_AM[nn]L register and CAN_AM[nn]H register manage acceptance mask operation. For information about acceptance mask operation, see the Receive Operation section.

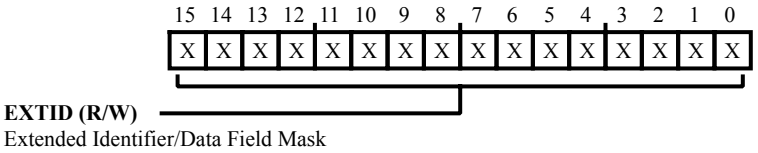

Figure 21-16: CAN_AM[nn]L Register Diagram

Table 21-11: CAN_AM[nn]L Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	EXTID	Extended Identifier/Data Field Mask. The CAN_AM[nn]L.EXTID bits hold the extended ID (lower 16 bits) for data field mask in acceptance mask operations.

Error Counter Register

The CAN_CEC register , CAN_ESR register, and CAN_EWR register control CAN warnings and errors. For detailed information about error and warning operations, see the Event Control section.

The CAN_CEC register holds an error counter for transmit (CAN_CEC.TXECNT) and an error counter for receive (CAN_CEC.RXECNT). After initialization, both counters are 0. Each time a bus error occurs, one of the counters is incremented by either 1 or 8, depending on the error situation (documented in Version 2.0 of CAN Specification). Successful transmit and receive operations decrement the respective counter by 1.

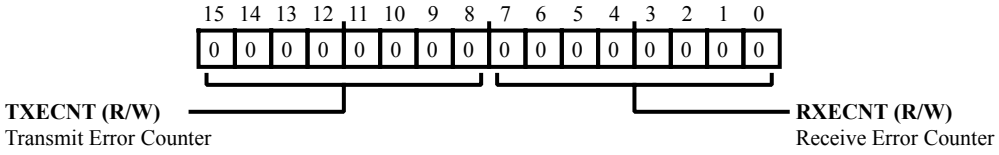

Figure 21-17: CAN_CEC Register Diagram

Table 21-12: CAN_CEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	TXECNT	Transmit Error Counter. The CAN_CEC.TXECNT bits hold the transmit error counter, which is incremented for errors (by either 1 or 8) and is decremented (by 1) for successful transmit operations.
7:0 (R/W)	RXECNT	Receive Error Counter. The CAN_CEC.RXECNT bits hold the receive error counter, which is incremented for errors (by either 1 or 8) and is decremented (by 1) for successful receive operations.

Clock Register

The CAN_CLK register select the bit rate prescaler for calculating the time quantum (TQ), which is used to derive the CAN clock from the system clock (SYSCLK). For more information about bit timing and clock operation, see the CAN Operating Modes section.

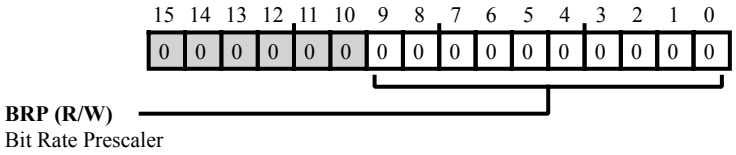

Figure 21-18: CAN_CLK Register Diagram

Table 21-13: CAN_CLK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	BRP	Bit Rate Prescaler. The CAN_CLK.BRP bits select the bit rate prescaler value, which is used to calculate the time quantum for CAN bit timing. The formula using CAN_CLK.BRP to calculate the time quantum is: TQ = (BRP+1) / SYSCLK Note that it is recommended that the CAN_CLK.BRP value be greater than or equal to 4. For more information about bit timing, see the Operating Modes section.

CAN Master Control Register

The CAN_CTL register controls CAN mode requests, including soft reset.

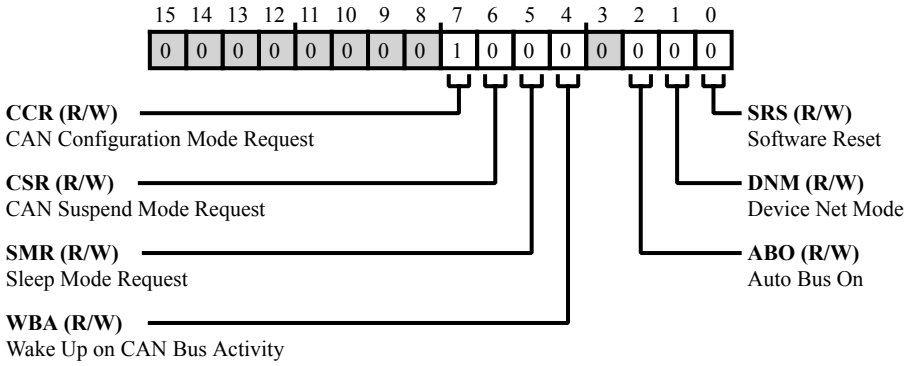

Figure 21-19: CAN_CTL Register Diagram

Table 21-14: CAN_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	CCR	CAN Configuration Mode Request. The CAN_CTL.CCR bit requests that the CAN enter configuration mode. Note that the CAN should always be put in configuration mode before modifying the CAN_CLK or CAN_TIMING registers.	
		0	No Request (Exit Configuration Mode)
		1	Request Configuration Mode
6 (R/W)	CSR	CAN Suspend Mode Request. The CAN_CTL.CSR bit requests that the CAN enter suspend mode. The CAN enters suspend mode after the current operation of the CAN bus is finished.	
		0	No Request (Exit Suspend Mode)
		1	Request Suspend Mode
5 (R/W)	SMR	Sleep Mode Request. The CAN_CTL.SMR bit requests that the CAN enter sleep mode. The CAN enters sleep mode after the current operation of the CAN bus is finished.	
		0	No Request (Exit Sleep Mode)
		1	Request Sleep Mode

Table 21-14: CAN_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	WBA	Wake Up on CAN Bus Activity. The CAN_CTL.WBA bit enables wake on CAN bus activity. When enabled, a dominant bit on the CAN_RX pin ends sleep mode (in addition the default wake up condition of a write to the CAN_INT register).	
		0	Disable Wake on Bus Activity
		1	Enable Wake on Bus Activity
2 (R/W)	ABO	Auto Bus On. The CAN_CTL.ABO bit selects whether (if enabled) the CAN enters active mode after the BusOff recovery sequence or (if disabled) the CAN enters configuration mode after the BusOff recovery sequence.	
		0	Disable Auto Bus On
		1	Enable Auto Bus On
1 (R/W)	DNM	Device Net Mode. The CAN_CTL.DNM bit enables mailbox filtering on a data field. The filtering is done on the standard ID of the message and data fields. For more information, see the CAN_AM[nn]H.FDF bit description.	
		0	Disable Device Net Mode
		1	Enable Device Net Mode
0 (R/W)	SRS	Software Reset. The CAN_CTL.SRS bit resets the CAN, bringing all control registers to a defined state. Soft reset is entered immediately after software has set the CAN_CTL.SRS bit.	
		0	No Action
		1	Reset CAN

Debug Register

The CAN_DBG register controls CAN debug modes, including CAN_TX and CAN_RX pin enable/disable.

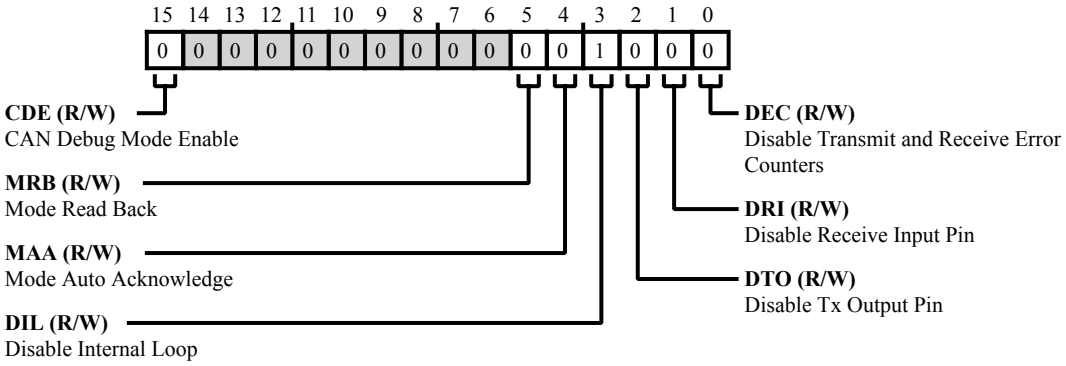

Figure 21-20: CAN_DBG Register Diagram

Table 21-15: CAN_DBG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	CDE	CAN Debug Mode Enable. The CAN_DBG.CDE bit enables debug mode. This bit must be written first before subsequent writes to the CAN_DBG register. When the CAN_DBG.CDE bit is cleared, all CAN debug features are disabled.	
		0	Disable Debug Mode
		1	Enable Debug Mode
5 (R/W)	MRB	Mode Read Back. The CAN_DBG.MRB bit enables read back mode. When enabled, a message transmitted on the CAN bus or through an internal loop back mode is received back directly to the internal receive buffer.	
		0	Disable Read Back Mode
		1	Enable Read Back Mode
4 (R/W)	MAA	Mode Auto Acknowledge. The CAN_DBG.MAA bit enables mode auto acknowledge, allowing the CAN to generate its own acknowledge during the ACK slot of the CAN frame. The CAN_DBG.MAA acknowledge appears on the CAN_TX pin if CAN_DBG.DIL =1 and CAN_DBG.DTO =0. If the acknowledge is only going to be used internally, these test mode bits should be set to CAN_DBG.DIL = 0 and CAN_DBG.DTO =1.	
		0	Disable Auto Acknowledge Mode
		1	Enable Auto Acknowledge Mode

Table 21-15: CAN_DBG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	DIL	Disable Internal Loop. The CAN_DBG.DIL bit disables internal loop mode, which routes the transmit output to the receive input.	
		0	Enable Internal Loop
		1	Disable Internal Loop
2 (R/W)	DTO	Disable Tx Output Pin. The CAN_DBG.DTO bit disables the CAN_TX pin.	
		0	Enable Tx Output Pin
		1	Disable Tx Output Pin, Drive Recessive
1 (R/W)	DRI	Disable Receive Input Pin. The CAN_DBG.DRI bit disables the CAN_RX pin.	
		0	Enable Rx Input Pin
		1	Disable Rx Input Pin, Drive Recessive Internally
0 (R/W)	DEC	Disable Transmit and Receive Error Counters. The CAN_DBG.DEC bit disables the transmit and receive error counters in the CAN_CEC register. When set, the CAN_CEC holds its current contents and is not allowed to increment or decrement the error counters. Note that this mode does not conform to the CAN specification.	
		0	Enable CEC Tx and Rx Error Counters
		1	Disable CEC Tx and Rx Error Counters

Error Status Register

The CAN_ESR register, CAN_CEC register, and CAN_EWR register control CAN warnings and errors. All bits in the CAN_ESR are W1C. Note that the CAN updates the CAN_CEC register when error status is detected in the CAN_ESR register. For detailed information about error and warning operations, see the Operating Modes section.

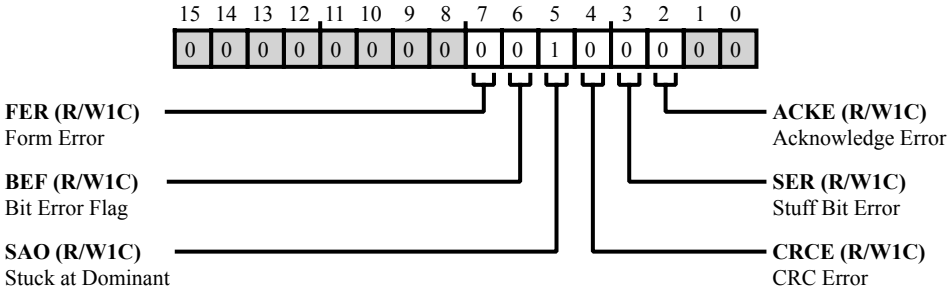

Figure 21-21: CAN_ESR Register Diagram

Table 21-16: CAN_ESR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	FER	Form Error. The CAN_ESR.FER bit indicates when a form error occurs, indicating that a fixed-form bit position in the CAN frame contains one or more illegal bits. This occurs when a dominant bit is detected at a delimiter or end-of-frame bit position.	
		0	No Status
		1	Form Error
6 (R/W1C)	BEF	Bit Error Flag. The CAN_ESR.BEF bit bit indicates (detected by the transmitting node only) the value on the CAN_RX pin does not equal what is being transmitted on the CAN_TX pin. When a node is transmitting, it continuously monitors its receive pin (CAN_RX) and compares the received data with the transmitted data. The node postpones the transmission (during the arbitration phase) if the received and transmitted data do not match. After the arbitration phase (CAN_MB[nn]_ID1.RTR bit sent successfully), a bit error is signaled when the value on the CAN_RX pin does not equal what is being transmitted on the CAN_TX pin.	
		0	No Status
		1	Bit Error Flag

Table 21-16: CAN_ESR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1C)	SAO	Stuck at Dominant. The CAN_ESR.SAO bit indicates when the CAN_RX pin sticks at dominant level, indicating that shorted wires are likely.	
		0	No Status
		1	Stuck At Dominant
4 (R/W1C)	CRCE	CRC Error. The CAN_ESR.CRCE bit indicates when a CRC error occurs. This error may occur when a receiver calculates the CRC on the data it received and finds the value different than the CRC that was transmitted on the bus.	
		0	No Status
		1	CRC Error
3 (R/W1C)	SER	Stuff Bit Error. The CAN_ESR.SER bit indicates when a stuff bit error (stuffed 6th consecutive bit value is the same as the previous five bits) occurs. The CAN specification requires that the transmitter insert an extra stuff bit of opposite value after 5 bits have been transmitted with the same value. The receiver disregards the value of these stuff bits. The receiver takes advantage of the signal edge to re-synchronize itself. A stuff bit error occurs on receiving nodes when the 6th consecutive bit value is the same as the previous five bits.	
		0	No Status
		1	Stuff Bit Error Receive
2 (R/W1C)	ACKE	Acknowledge Error. The CAN_ESR.ACKE bit indicates when an acknowledge error occurs, indicating that a message is sent and no receivers drive an acknowledge bit.	
		0	No Status
		1	Acknowledge Error

Error Counter Warning Level Register

The CAN_EWR register, CAN_CEC register, and CAN_ESR register control CAN warnings and errors. For detailed information about error and warning operations, see the Operating Modes section.

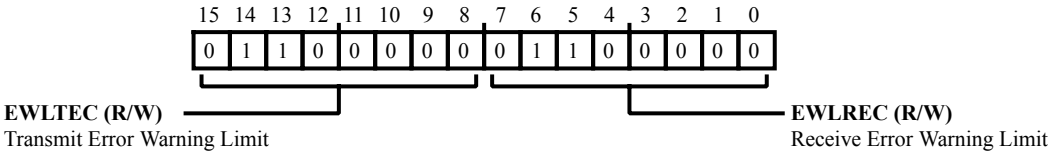

Figure 21-22: CAN_EWR Register Diagram

Table 21-17: CAN_EWR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	EWLTEC	Transmit Error Warning Limit. The CAN_EWR.EWLTEC bits select the transmit error warning limit, which is used as a condition for the CAN_GIS.EWTIS interrupt.
7:0 (R/W)	EWLREC	Receive Error Warning Limit. The CAN_EWR.EWLREC bits select the receive error warning limit, which is used as a condition for the CAN_GIS.EWRIS interrupt.

Global CAN Interrupt Flag Register

The CAN_GIF register, CAN_GIF register, and CAN_GIM register control CAN interrupts. For detailed information about interrupt operations, see the Event Control section.

The CAN_GIF register holds the interrupt flag. The CAN_INT.GIRQ bit is only asserted if a bit in the CAN_GIF is set. The CAN_INT.GIRQ bit remains set as long as at least one bit in the CAN_GIF register is set.All bits in this register are W1C.

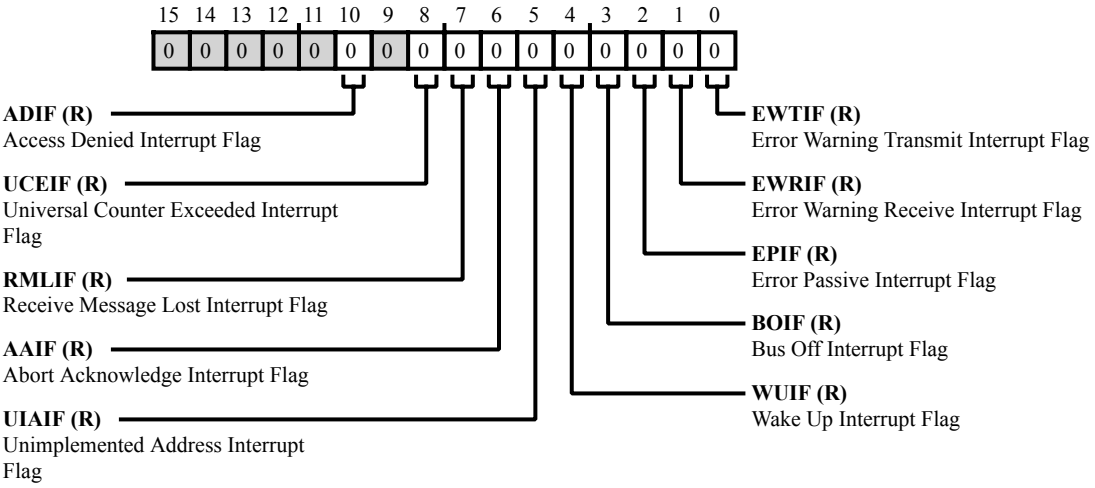

Figure 21-23: CAN_GIF Register Diagram

Table 21-18: CAN_GIF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/NW)	ADIF	Access Denied Interrupt Flag. The CAN_GIF.ADIF bit indicates the access denied interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
8 (R/NW)	UCEIF	Universal Counter Exceeded Interrupt Flag. The CAN_GIF.UCEIF bit indicates the universal counter exceeded interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)

Table 21-18: CAN_GIF Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	RMLIF	Receive Message Lost Interrupt Flag. The CAN_GIF.RMLIF bit indicates the receive message lost interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
6 (R/NW)	AAIF	Abort Acknowledge Interrupt Flag. The CAN_GIF.AAIF bit indicates the abort acknowledge interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
5 (R/NW)	UIAIF	Unimplemented Address Interrupt Flag. The CAN_GIF.UIAIF bit indicates the unimplemented address interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
4 (R/NW)	WUIF	Wake Up Interrupt Flag. The CAN_GIF.WUIF bit indicates the wake up interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
3 (R/NW)	BOIF	Bus Off Interrupt Flag. The CAN_GIF.BOIF bit indicates the bus off interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
2 (R/NW)	EPIF	Error Passive Interrupt Flag. The CAN_GIF.EPIF bit indicates the error passive mode interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
1 (R/NW)	EWRIF	Error Warning Receive Interrupt Flag. The CAN_GIF.EWRIF bit indicates the error warning receive interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)

Table 21-18: CAN_GIF Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/NW)	EWTIF	Error Warning Transmit Interrupt Flag. The `CAN_GIF.EWTIF` bit indicates the error warning transmit interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)

Global CAN Interrupt Mask Register

The CAN_GIM register, CAN_GIF register, and CAN_GIF register control CAN interrupts. For detailed information about interrupt operations, see the Event Control section.

The CAN_GIM register holds the interrupt mask. The interrupt mask bits only affect the content of the CAN_GIF register. If the mask bit is not set (enabled/unmasked), the corresponding flag bit is not set when the event occurs.

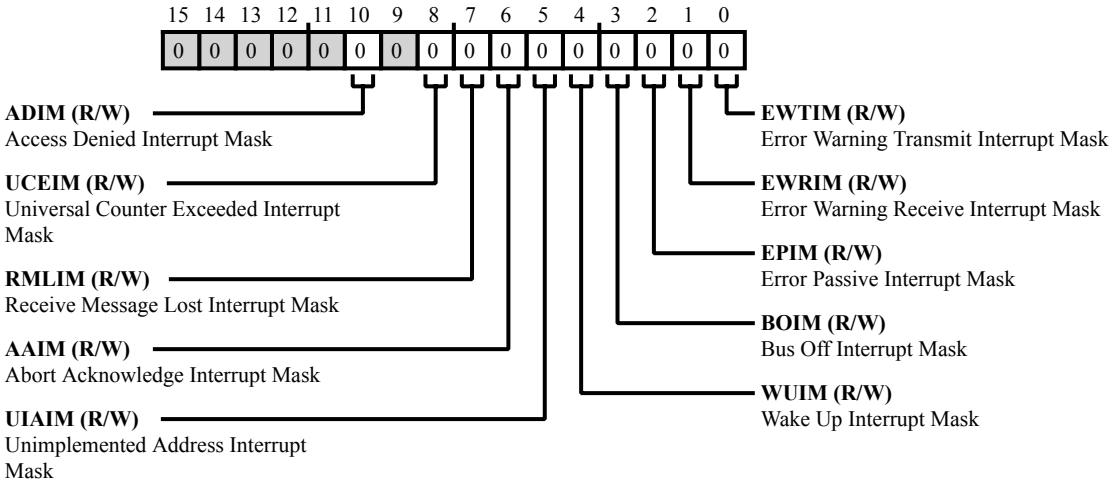

Figure 21-24: CAN_GIM Register Diagram

Table 21-19: CAN_GIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W)	ADIM	Access Denied Interrupt Mask. The CAN_GIM.ADIM bit enables (unmasks) the access denied interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
8 (R/W)	UCEIM	Universal Counter Exceeded Interrupt Mask. The CAN_GIM.UCEIM bit enables (unmasks) the universal counter exceeded interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
7 (R/W)	RMLIM	Receive Message Lost Interrupt Mask. The CAN_GIM.RMLIM bit enables (unmasks) the receive message lost interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)

Table 21-19: CAN_GIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	AAIM	Abort Acknowledge Interrupt Mask. The CAN_GIM.AAIM bit enables (unmasks) the abort acknowledge interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
5 (R/W)	UIAIM	Unimplemented Address Interrupt Mask. The CAN_GIM.UIAIM bit enables (unmasks) the unimplemented address interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
4 (R/W)	WUIM	Wake Up Interrupt Mask. The CAN_GIM.WUIM bit enables (unmasks) the wake up interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
3 (R/W)	BOIM	Bus Off Interrupt Mask. The CAN_GIM.BOIM bit enables (unmasks) the bus off interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
2 (R/W)	EPIM	Error Passive Interrupt Mask. The CAN_GIM.EPIM bit enables (unmasks) the error passive mode interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
1 (R/W)	EWRIM	Error Warning Receive Interrupt Mask. The CAN_GIM.EWRIM bit enables (unmasks) the error warning receive interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
0 (R/W)	EWTIM	Error Warning Transmit Interrupt Mask. The CAN_GIM.EWTIM bit enables (unmasks) the error warning transmit interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)

Global CAN Interrupt Status Register

The CAN_GIS register, CAN_GIF register, and CAN_GIM register control CAN interrupts. For detailed information about interrupt operations, see the Event Control section.

The CAN_GIS register holds the interrupt status. All bits in this register are W1C.

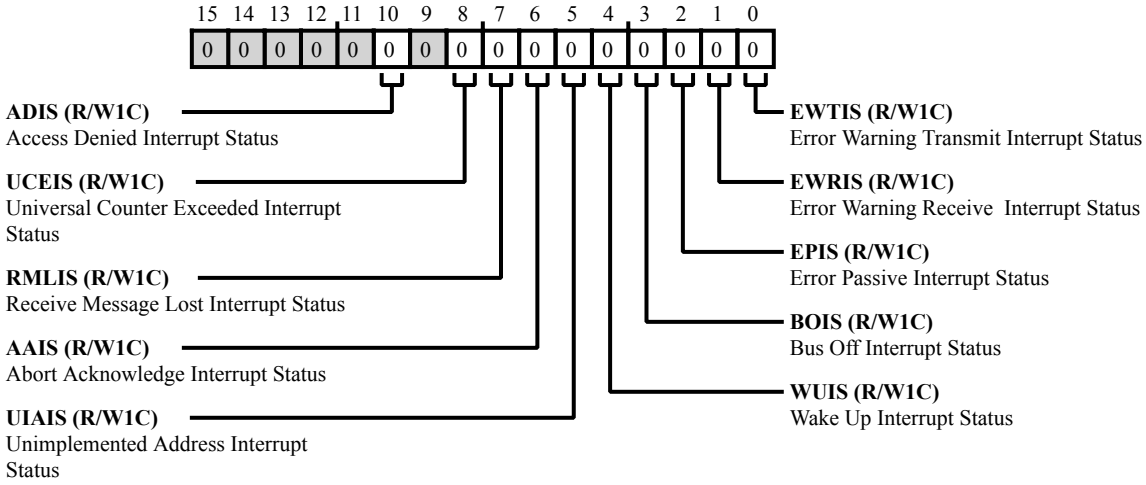

Figure 21-25: CAN_GIS Register Diagram

Table 21-20: CAN_GIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W1C)	ADIS	Access Denied Interrupt Status. The CAN_GIS.ADIS bit indicates when at least one access to the mailbox RAM occurred during a data update by internal logic.	
		0	No Interrupt Pending
		1	Interrupt Pending
8 (R/W1C)	UCEIS	Universal Counter Exceeded Interrupt Status. The CAN_GIS.UCEIS bit indicates when there has been an overflow of the universal counter (in time stamp mode or event counter mode) or the counter has reached the value 0x0000 (in watchdog mode).	
		0	No Interrupt Pending
		1	Interrupt Pending

Table 21-20: CAN_GIS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	RMLIS	Receive Message Lost Interrupt Status. The CAN_GIS.RMLIS bit indicates when a message is received for a mailbox that currently contains unread data. At least one bit in the receive message lost register (CAN_RML1 or CAN_RML2) is set. If the bit in CAN_GIS (and CAN_GIF) is reset and there is at least one bit in CAN_RML1 or CAN_RML2 still set, the bit in CAN_GIF (and CAN_GIF) is not set again. The internal interrupt source signal is only active if a new bit in CAN_RML1 or CAN_RML2 is set.	
		0	No Interrupt Pending
		1	Interrupt Pending
6 (R/W1C)	AAIS	Abort Acknowledge Interrupt Status. The CAN_GIS.AAIS bit indicates when At least one abort acknowledge bit is set in the CAN_AA1 or the CAN_AA2 registers. If the bit in CAN_GIS (and CAN_GIF) is reset and there is at least one bit in CAN_AA1 or CAN_AA2 still set, the bit in CAN_GIS (and CAN_GIF) is not set again. The internal interrupt source signal is only active if a new bit in CAN_AA1 or CAN_AA2 is set. The abort acknowledge bits maintain state even after the corresponding mailbox n is disabled.	
		0	No Interrupt Pending
		1	Interrupt Pending
5 (R/W1C)	UIAIS	Unimplemented Address Interrupt Status. The CAN_GIS.UIAIS bit indicates when there was a processor core access to an address that is not implemented in the CAN.	
		0	No Interrupt Pending
		1	Interrupt Pending
4 (R/W1C)	WUIS	Wake Up Interrupt Status. The CAN_GIS.WUIS bit indicates when the CAN has left the sleep mode because of detected activity on the CAN bus line.	
		0	No Interrupt Pending
		1	Interrupt Pending
3 (R/W1C)	BOIS	Bus Off Interrupt Status. The CAN_GIS.BOIS bit indicates when the CAN has entered the bus-off state. This interrupt source is active if the status of the CAN changes from normal operation mode to the bus-off mode. If the bit in CAN_GIS (and CAN_GIF) is reset and the bus-off mode is still active, this bit is not set again. If the module leaves the bus-off mode, the bit in CAN_GIS (and CAN_GIF) remains set.	
		0	No Interrupt Pending
		1	Interrupt Pending

Table 21-20: CAN_GIS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	EPIS	Error Passive Interrupt Status. The CAN_GIS.EPIS bit indicates when the CAN has entered the error passive state. This interrupt source is active if the status of the CAN changes from the error active mode to the error passive mode. If the bit in CAN_GIS (and CAN_GIF) is reset and the error passive mode is still active, this bit is not set again. If the CAN leaves the error passive mode, the bit in CAN_GIS (and CAN_GIF) remains set.	
		0	No Interrupt Pending
		1	Interrupt Pending
1 (R/W1C)	EWRIS	Error Warning Receive Interrupt Status. The CAN_GIS.EWRIS bit indicates when the CAN_CEC.RXECNT has reached the warning limit. If the bit in CAN_GIS (and CAN_GIF) is reset and the error warning mode is still active, this bit is not set again. If the CAN leaves the error warning mode, the bit in CAN_GIS (and CAN_GIF) remains set.	
		0	No Interrupt Pending
		1	Interrupt Pending
0 (R/W1C)	EWTIS	Error Warning Transmit Interrupt Status. The CAN_GIS.EWTIS bit indicates when the CAN_CEC.TXECNT has reached the warning limit. If the bit in CAN_GIS (and CAN_GIF) is reset and the error warning mode is still active, this bit is not set again. If the CAN leaves the error warning mode, the bit in CAN_GIS (and CAN_GIF) remains set.	
		0	No Interrupt Pending
		1	Interrupt Pending

Interrupt Pending Register

The CAN_INT register indicates the status of pending CAN interrupts and indicates the state of the CAN_RX and CAN_TX pins. Though this register is read-only, a write is allowed to exit the built-in sleep mode of the module on processors supporting this feature.

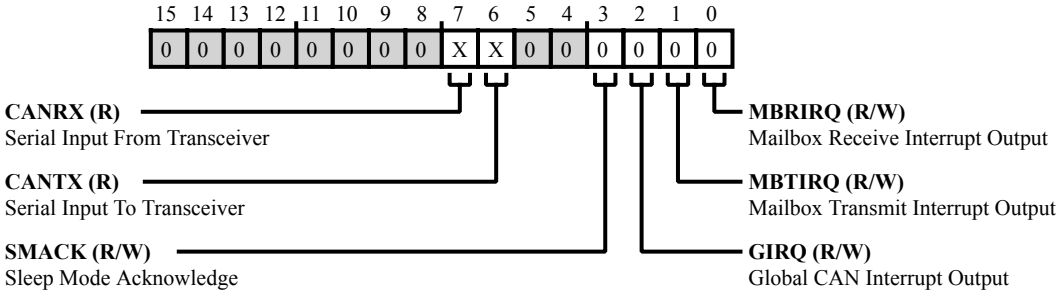

Figure 21-26: CAN_INT Register Diagram

Table 21-21: CAN_INT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	CANRX	Serial Input From Transceiver. The CAN_INT.CANRX bit indicates the logic value that the CAN detects on the CAN_RX pin. Note that the reset/default value for CAN_INT.CANRX is dependent on pin values.	
		0	Dominant Value (Low Active)
		1	Recessive Value (High Active)
6 (R/NW)	CANTX	Serial Input To Transceiver. The CAN_INT.CANTX bit indicates the logic value that the CAN detects on the CAN_TX pin. Note that the reset/default value for CAN_INT.CANTX is dependent on pin values.	
		0	Dominant Value (Low Active)
		1	Recessive Value (High Active)
3 (R/W)	SMACK	Sleep Mode Acknowledge. The CAN_INT.SMACK bit indicates when the CAN has entered sleep mode.	
		0	Not in Sleep Mode
		1	Sleep Mode

Table 21-21: CAN_INT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	GIRQ	Global CAN Interrupt Output. The CAN_INT.GIRQ bit indicates when at least one bit is set in the CAN_GIF register, indicating at least one unmasked CAN is flagged (latched). The CAN_INT.GIRQ bit remains set as long as at least one bit is set in the CAN_GIF register.	
		0	No CAN Global Interrupt Flag Set
		1	CAN Global Interrupt Flag (1 or More) Set
1 (R/W)	MBTIRQ	Mailbox Transmit Interrupt Output. The CAN_INT.MBTIRQ bit indicates when any bits are set in the CAN_MBTIF1 register or CAN_MBTIF2 register, indicating transmit.	
		0	No CAN Transmit Flags Set
		1	CAN Transmit Flags Set (1 or More)
0 (R/W)	MBRIRQ	Mailbox Receive Interrupt Output. The CAN_INT.MBRIRQ bit indicates when any bits are set in the CAN_MBRIF1 register or CAN_MBRIF2 register, indicating receive.	
		0	No CAN Receive Flags Set
		1	CAN Receive Flags Set (1 or More)

Mailbox Interrupt Mask 1 Register

The CAN_MBIM1 register enables transmit and receive interrupts for mailboxes 0 through 15. Each bit in this register requests enables the transmit or receive interrupt for the corresponding mailbox when set (=1).

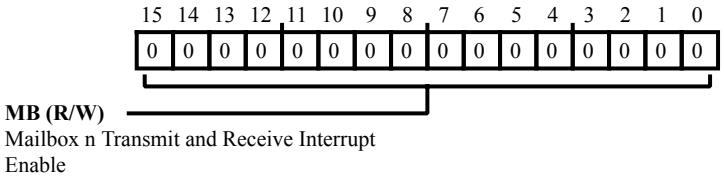

Figure 21-27: CAN_MBIM1 Register Diagram

Table 21-22: CAN_MBIM1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Transmit and Receive Interrupt Enable.

Mailbox Interrupt Mask 2 Register

The CAN_MBIM2 register enables transmit and receive interrupts for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register requests enables the transmit or receive interrupt for the corresponding mailbox when set (=1).

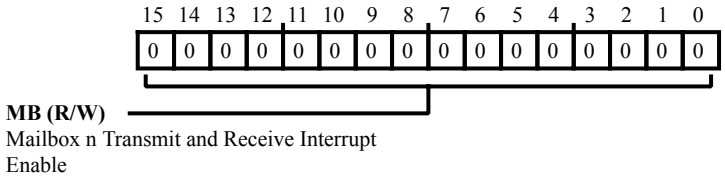

Figure 21-28: CAN_MBIM2 Register Diagram

Table 21-23: CAN_MBIM2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Transmit and Receive Interrupt Enable.

Mailbox Receive Interrupt Flag 1 Register

The CAN_MBRIF1 register indicates when a receive interrupt is pending---due to successful reception (corresponding CAN_RMP1 bit set) and the interrupt is enabled (corresponding CAN_MBIM1 bit set)---for mailboxes 0 through 15. Each bit in this register indicates the receive interrupt pending status for the corresponding mailbox when set (=1). When any bit in CAN_MBRIF1 is set, the CAN receive interrupt request is raised (CAN_INT.MBRIRQ bit set). To clear the interrupt request, all of the set bits in CAN_RMP1 must be cleared by software, then the associated bits set in CAN_MBRIF1 must be cleared (W1C).

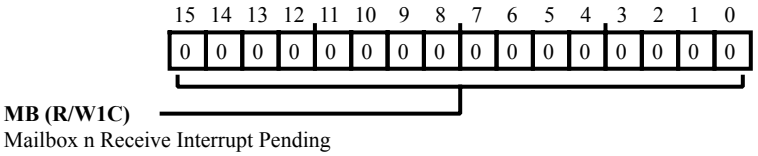

Figure 21-29: CAN_MBRIF1 Register Diagram

Table 21-24: CAN_MBRIF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Receive Interrupt Pending.

Mailbox Receive Interrupt Flag 2 Register

The CAN_MBRIF2 register indicates when a receive interrupt is pending---due to successful reception (corresponding CAN_RMP2 bit set) and the interrupt is enabled (corresponding CAN_MBIM2 bit set)---for mailboxes 16 (bit 0) through 23 (bit 7). Each bit in this register indicates the receive interrupt pending status for the corresponding mailbox when set (=1). When any bit in CAN_MBRIF2 is set, the CAN receive interrupt request is raised (CAN_INT.MBRIRQ bit set). To clear the interrupt request, all of the set bits in CAN_RMP2 must be cleared by software, then the associated bits set in CAN_MBRIF2 must be cleared (W1C). Bits 8 through 15 are reserved and read-only, as the corresponding mailboxes (24 through 31) are transmit-only mailboxes.

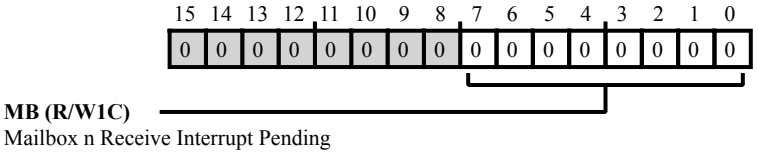

Figure 21-30: CAN_MBRIF2 Register Diagram

Table 21-25: CAN_MBRIF2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W1C)	MB	Mailbox n Receive Interrupt Pending.

Temporary Mailbox Disable Register

The CAN_MBTD register supports temporarily and selectively disabling CAN mailboxes. For more information about this feature, see the Operating Modes section.

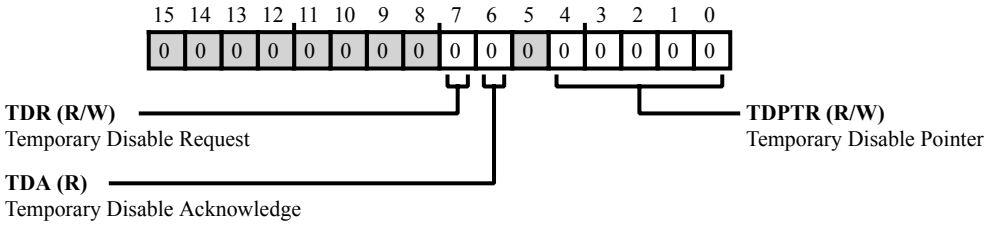

Figure 21-31: CAN_MBTD Register Diagram

Table 21-26: CAN_MBTD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	TDR	Temporary Disable Request. The CAN_MBTD.TDR bit hold the pointer to mailbox, which is disabled when the CAN_MBTD.TDR bit is set.	
		0	No Request
		1	Request Temporary Mailbox Disable
6 (R/NW)	TDA	Temporary Disable Acknowledge. The CAN_MBTD.TDA bit indicates when the mailbox (to which the CAN_MBTD.TDPTR bit point) is disabled. When this bit is set for a mailbox, only the data field of that mailbox may be updated. Accesses that mailboxs control bits and the identifier are denied.	
		0	No Acknowledge
		1	Acknowledge Temporary Mailbox Disable
4:0 (R/W)	TDPTR	Temporary Disable Pointer. The CAN_MBTD.TDPTR bits hold the pointer to mailbox, which is disabled when the CAN_MBTD.TDR bit is set.	

Mailbox Transmit Interrupt Flag 1 Register

The CAN_MBTIF1 register indicates when a transmit interrupt is pending---due to successful transmission (corresponding CAN_TA1 bit set) and the interrupt is enabled (corresponding CAN_MBIM1 bit set)---for mailboxes 8 through 15. Each bit in this register indicates the transmit interrupt pending status for the corresponding mailbox when set (=1). When any bit in CAN_MBTIF1 is set, the CAN transmit interrupt request is raised (CAN_INT.MBTIRQ bit set). To clear the interrupt request, all of the set bits in CAN_MBTIF1 must be cleared by software (W1C). Also, software must clear the associated bits set in CAN_TA1 or set the associated bits in CAN_TRS1 bit to clear the interrupt source asserting the bits in CAN_MBTIF1. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

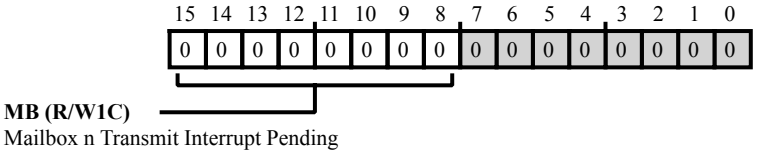

Figure 21-32: CAN_MBTIF1 Register Diagram

Table 21-27: CAN_MBTIF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W1C)	MB	Mailbox n Transmit Interrupt Pending.

Mailbox Transmit Interrupt Flag 2 Register

The CAN_MBTIF2 register indicates when a transmit interrupt is pending---due to successful transmission (corresponding CAN_TA2 bit set) and the interrupt is enabled (corresponding CAN_MBIM2 bit set)---for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register indicates the transmit interrupt pending status for the corresponding mailbox when set (=1). When any bit in CAN_MBTIF2 is set, the CAN transmit interrupt request is raised (CAN_INT.MBTIRQ bit set). To clear the interrupt request, all of the set bits in CAN_MBTIF2 must be cleared by software (W1C). Also, software must clear the associated bits set in CAN_TA2 or set the associated bits in CAN_TRS2 bit to clear the interrupt source asserting the bits in CAN_MBTIF2.

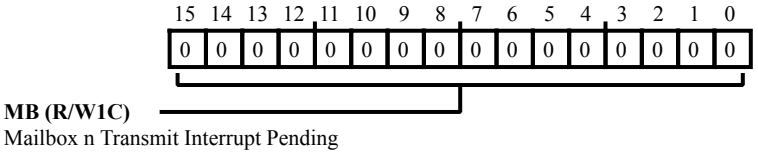

Figure 21-33: CAN_MBTIF2 Register Diagram

Table 21-28: CAN_MBTIF2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Transmit Interrupt Pending.

Mailbox Word 0 Register

The CAN_MB[nn]_DATA0 register holds mailbox data bytes.

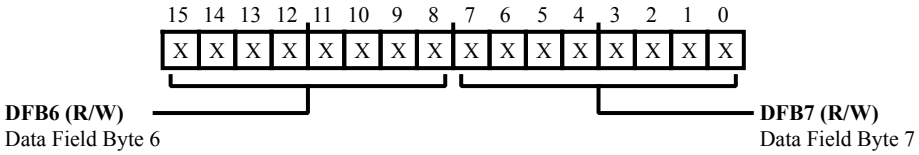

Figure 21-34: CAN_MB[nn]_DATA0 Register Diagram

Table 21-29: CAN_MB[nn]_DATA0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	DFB6	Data Field Byte 6. The CAN_MB[nn]_DATA0.DFB6 bits hold mailbox data.
7:0 (R/W)	DFB7	Data Field Byte 7. The CAN_MB[nn]_DATA0.DFB7 bits hold mailbox data.

Mailbox Word 1 Register

The CAN_MB[nn]_DATA1 register holds mailbox data bytes.

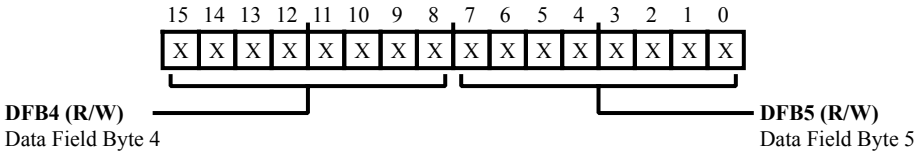

Figure 21-35: CAN_MB[nn]_DATA1 Register Diagram

Table 21-30: CAN_MB[nn]_DATA1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	DFB4	Data Field Byte 4. The CAN_MB[nn]_DATA1.DFB4 bits hold mailbox data.
7:0 (R/W)	DFB5	Data Field Byte 5. The CAN_MB[nn]_DATA1.DFB5 bits hold mailbox data.

Mailbox Word 2 Register

The CAN_MB[nn]_DATA2 register holds mailbox data bytes.

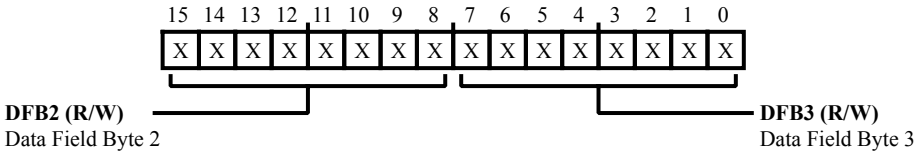

Figure 21-36: CAN_MB[nn]_DATA2 Register Diagram

Table 21-31: CAN_MB[nn]_DATA2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	DFB2	Data Field Byte 2. The CAN_MB[nn]_DATA2.DFB2 bits hold mailbox data.
7:0 (R/W)	DFB3	Data Field Byte 3. The CAN_MB[nn]_DATA2.DFB3 bits hold mailbox data.

Mailbox Word 3 Register

The CAN_MB[nn]_DATA3 register holds mailbox data bytes.

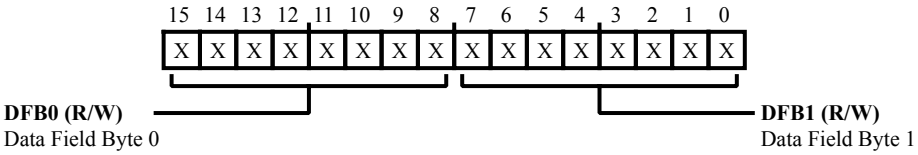

Figure 21-37: CAN_MB[nn]_DATA3 Register Diagram

Table 21-32: CAN_MB[nn]_DATA3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	DFB0	Data Field Byte 0. The CAN_MB[nn]_DATA3.DFB0 bits hold mailbox data.
7:0 (R/W)	DFB1	Data Field Byte 1. The CAN_MB[nn]_DATA3.DFB1 bits hold mailbox data.

Mailbox ID 0 Register

The CAN_MB[nn]_ID0 register contains the lower 16 bits of the 18-bit extended identifier.

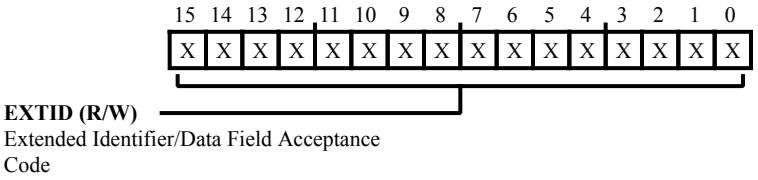

Figure 21-38: CAN_MB[nn]_ID0 Register Diagram

Table 21-33: CAN_MB[nn]_ID0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	EXTID	Extended Identifier/Data Field Acceptance Code. The CAN_MB[nn]_ID0.EXTID bits hold the lower 16 bits of the 18-bit extended ID.

Mailbox ID 1 Register

The CAN_MB[nn]_ID1 register contains the identifier bits of mailbox. The 11-bit BASE_ID is mapped to The CAN_MB[nn]_ID1.BASEID field. It also enables the extended identification and contains upper two bits of 18-bit extended identifier.

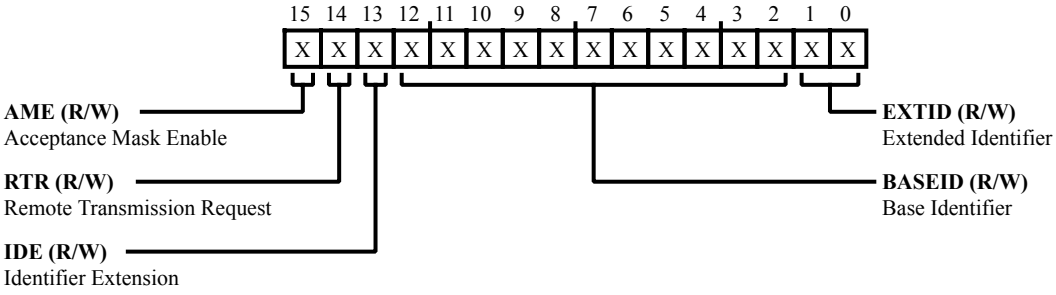

Figure 21-39: CAN_MB[nn]_ID1 Register Diagram

Table 21-34: CAN_MB[nn]_ID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	AME	Acceptance Mask Enable. The CAN_MB[nn]_ID1.AME bit enables acceptance mask operations if the mailbox is configured as receiver. When enabled (=1), only those bits that have the corresponding mask bit cleared are compared to the received message ID. A bit position that is set in the mask register does not need to match. This bit should be set to 0 when the mailbox is configured in transmit mode.
14 (R/W)	RTR	Remote Transmission Request. The CAN_MB[nn]_ID1.RTR bit selects whether the frame contains data (data frame) or contains a request for data associated with the message identifier in the frame being sent (remote frame).
13 (R/W)	IDE	Identifier Extension. The CAN_MB[nn]_ID1.IDE bit enables the comparison of the received message ID to the value in the CAN_MB[nn]_ID1.EXTID and CAN_MB[nn]_ID0.EXTID bits. When configured as transmitter, it sends the extended identifier in addition to the base identifier.
12:2 (R/W)	BASEID	Base Identifier. The CAN_MB[nn]_ID1.BASEID bits hold the base identifier for acceptance mask operations.
1:0 (R/W)	EXTID	Extended Identifier. The CAN_MB[nn]_ID1.EXTID bits hold the upper two bits of 18-bit extended identifier.

Mailbox Length Register

The CAN_MB[nn]_LENGTH register holds the data length code for the received remote frame. For more information about remote frames, see the Remote Frame Handling section.

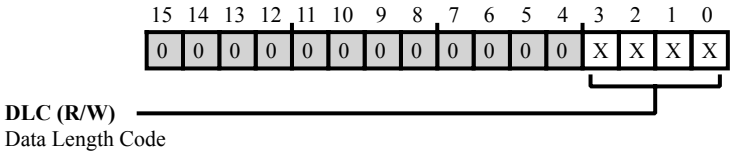

Figure 21-40: CAN_MB[nn]_LENGTH Register Diagram

Table 21-35: CAN_MB[nn]_LENGTH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	DLC	Data Length Code. The CAN_MB[nn]_LENGTH.DLC bits hold the DLC value of the received remote frame. The received value overwrites any previous value.

Mailbox Timestamp Register

The CAN_MB[nn]_TIMESTAMP register holds an indication of the time of reception or transmission for each message, when the universal counter is in time stamp mode (CAN_UCCNF.UCCNF =0x1). In this mode, the CAN writes the value of the counter (CAN_UCCNT) to the CAN_MB[nn]_TIMESTAMP register when a received message is stored or a message is transmitted. For more information about timestamps, see the Time Stamps section.

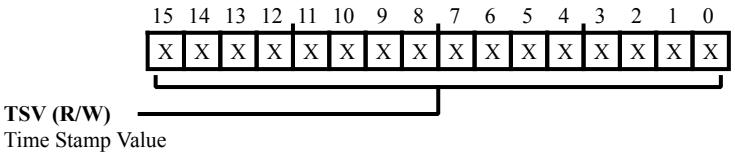

Figure 21-41: CAN_MB[nn]_TIMESTAMP Register Diagram

Table 21-36: CAN_MB[nn]_TIMESTAMP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	TSV	Time Stamp Value. The CAN_MB[nn]_TIMESTAMP.TSV bits hold the message timestamp value.

Mailbox Configuration 1 Register

The CAN_MC1 register enables mailboxes 0 through 15. Each bit in this register enables or disables the corresponding mailbox. For all bits, set the bit (=1) to enable the mailbox, and clear the bit (=0) to disable the mailbox.

Enabling and disabling mailboxes has an impact on transmit requests. Setting the CAN_TRS1 bit associated with a disabled mailbox may result in erroneous behavior. Similarly, disabling a mailbox before the associated CAN_TRS1 bit is reset by the internal logic can cause unpredictable results.

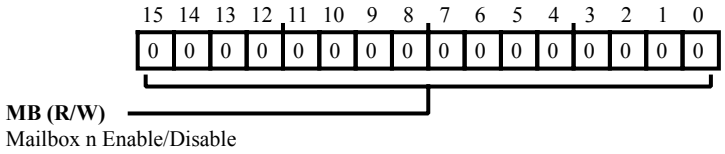

Figure 21-42: CAN_MC1 Register Diagram

Table 21-37: CAN_MC1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Enable/Disable.

Mailbox Configuration 2 Register

The CAN_MC2 register enables mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register enables or disables the corresponding mailbox. For all bits, set the bit (=1) to enable the mailbox, and clear the bit (=0) to disable the mailbox.

Enabling and disabling mailboxes has an impact on transmit requests. Setting the CAN_TRS2 bit associated with a disabled mailbox may result in erroneous behavior. Similarly, disabling a mailbox before the associated CAN_TRS2 bit is reset by the internal logic can cause unpredictable results.

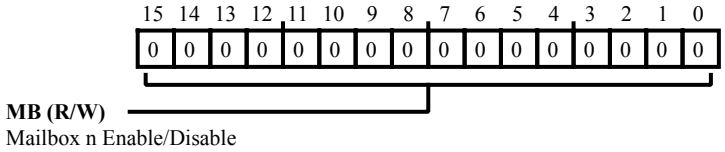

Figure 21-43: CAN_MC2 Register Diagram

Table 21-38: CAN_MC2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Enable/Disable.

Mailbox Direction 1 Register

The CAN_MD1 register selects the data transfer direction for mailboxes 0 through 15. Each bit in this register selects receive mode or transmit mode for the corresponding mailbox. For all bits, set the bit (=1) for receive mode from the mailbox, and clear the bit (=0) for transmit mode to the mailbox. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

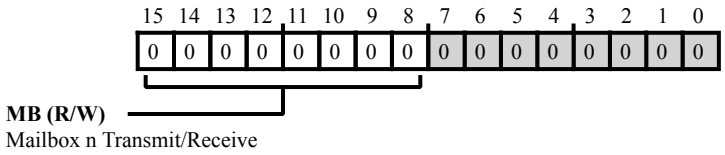

Figure 21-44: CAN_MD1 Register Diagram

Table 21-39: CAN_MD1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	MB	Mailbox n Transmit/Receive.

Mailbox Direction 2 Register

The CAN_MD2 register selects the data transfer direction for mailboxes 16 (bit 0) through 23 (bit 7). Each bit in this register selects receive mode or transmit mode for the corresponding mailbox. For all bits, set the bit (=1) for receive mode from the mailbox, and clear the bit (=0) for transmit mode to the mailbox. Bits 8 through 15 are read-only, as the corresponding mailboxes (24 through 31) are transmit-only mailboxes.

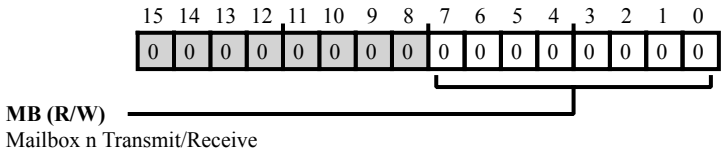

Figure 21-45: CAN_MD2 Register Diagram

Table 21-40: CAN_MD2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	MB	Mailbox n Transmit/Receive.

Overwrite Protection/Single Shot Transmission 1 Register

The CAN_OPSS1 register enables overwrite protection for mailboxes 0 through 15. Each bit in this register enables overwrite protection for the corresponding mailbox when set (=1). Note that enabling this bit affects transmit and receive operations for mailboxes. For more information about remote overwrite protection, see the detailed feature description in the CAN Functional Description section. For more information about how this feature affects transmit and receive operations, see the CAN Operating Modes sections, describing transmit and receive operations.

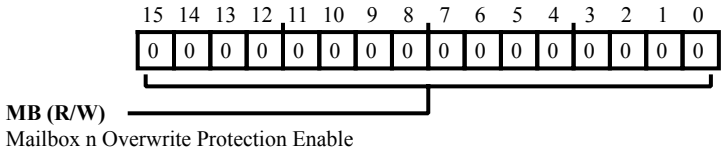

Figure 21-46: CAN_OPSS1 Register Diagram

Table 21-41: CAN_OPSS1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Overwrite Protection Enable.

Overwrite Protection/Single Shot Transmission 2 Register

The CAN_OPSS2 register enables overwrite protection for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register enables overwrite protection for the corresponding mailbox when set (=1). Note that enabling this bit affects transmit and receive operations for mailboxes. For more information about remote overwrite protection, see the detailed feature description in the CAN Functional Description section. For more information about how this feature affects transmit and receive operations, see the CAN Operating Modes sections, describing transmit and receive operations.

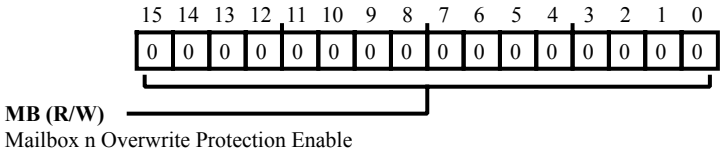

Figure 21-47: CAN_OPSS2 Register Diagram

Table 21-42: CAN_OPSS2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Overwrite Protection Enable.

Remote Frame Handling 1 Register

The CAN_RFH1 register enables remote frame handling for mailboxes 8 through 15. Each bit in this register enables remote frame handling for the corresponding mailbox when set (=1). Note that enabling this bit affects transmit and receive operations for mailboxes. For more information about remote frame handling, see the CAN Operating Modes sections, describing transmit and receive operations. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

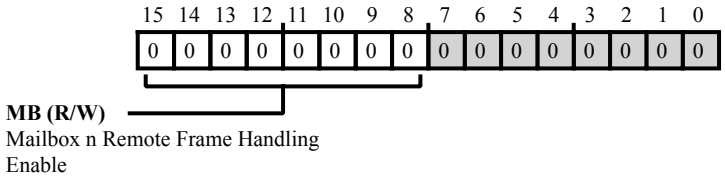

Figure 21-48: CAN_RFH1 Register Diagram

Table 21-43: CAN_RFH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	MB	Mailbox n Remote Frame Handling Enable.

Remote Frame Handling 2 Register

The CAN_RFH2 register enables remote frame handling for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register enables remote frame handling for the corresponding mailbox when set (=1). Note that enabling this bit affects transmit and receive operations for mailboxes. For more information about remote frame handling, see the CAN Operating Modes sections, describing transmit and receive operations.

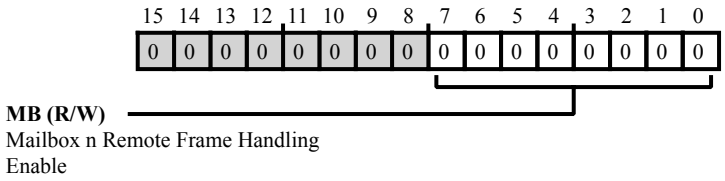

Figure 21-49: CAN_RFH2 Register Diagram

Table 21-44: CAN_RFH2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	MB	Mailbox n Remote Frame Handling Enable.

Receive Message Lost 1 Register

The CAN_RML1 register indicates when a message is lost---due to a message coming while there is pending data (corresponding CAN_RMP1 bit set) and overwrite protection is disabled (CAN_OPSS1 bit cleared)---for mailboxes 0 through 15. Each bit in this register indicates the message lost status for the corresponding mailbox when set (=1).

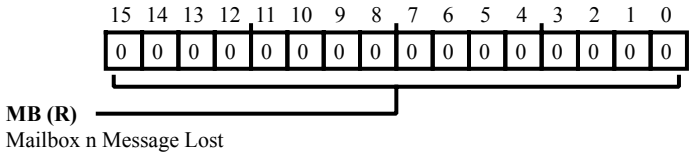

Figure 21-50: CAN_RML1 Register Diagram

Table 21-45: CAN_RML1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	MB	Mailbox n Message Lost.

Receive Message Lost 2 Register

The CAN_RML2 register indicates when a message is lost---due to a message coming while there is pending data (corresponding CAN_RMP2 bit set) and overwrite protection is disabled (CAN_OPSS2 bit cleared)---for mailboxes 16 (bit 0) through 23 (bit 7). Each bit in this register indicates the message lost status for the corresponding mailbox when set (=1). Bits 8 through 15 are reserved, as the corresponding mailboxes (24 through 31) are transmit-only mailboxes.

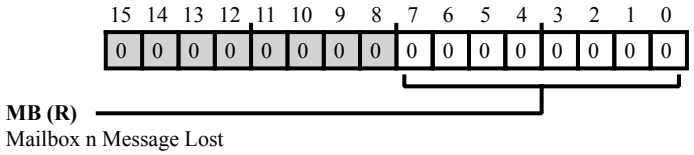

Figure 21-51: CAN_RML2 Register Diagram

Table 21-46: CAN_RML2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	MB	Mailbox n Message Lost.

Receive Message Pending 1 Register

The CAN_RMP1 register indicates when a message is pending (unread data) for mailboxes 0 through 15. Each bit in this register indicates the message pending status for the corresponding mailbox when set (=1).

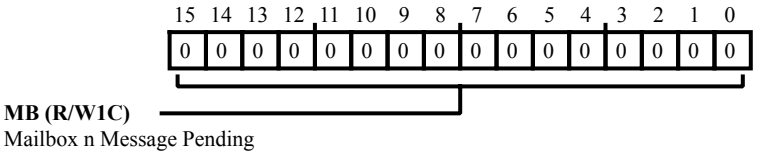

Figure 21-52: CAN_RMP1 Register Diagram

Table 21-47: CAN_RMP1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Message Pending.

Receive Message Pending 2 Register

The CAN_RMP2 register indicates when a message is pending (unread data) for mailboxes 16 (bit 0) through 23 (bit 7). Each bit in this register indicates the message pending status for the corresponding mailbox when set (=1). Bits 8 through 15 are reserved, as the corresponding mailboxes (24 through 31) are transmit-only mailboxes.

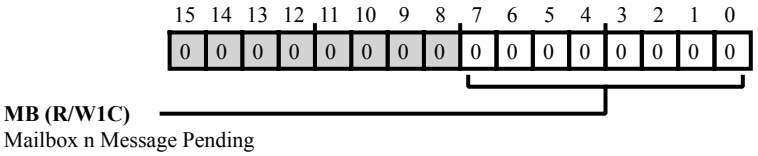

Figure 21-53: CAN_RMP2 Register Diagram

Table 21-48: CAN_RMP2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W1C)	MB	Mailbox n Message Pending.

Status Register

The CAN_STAT register indicates status for CAN modes and error conditions.

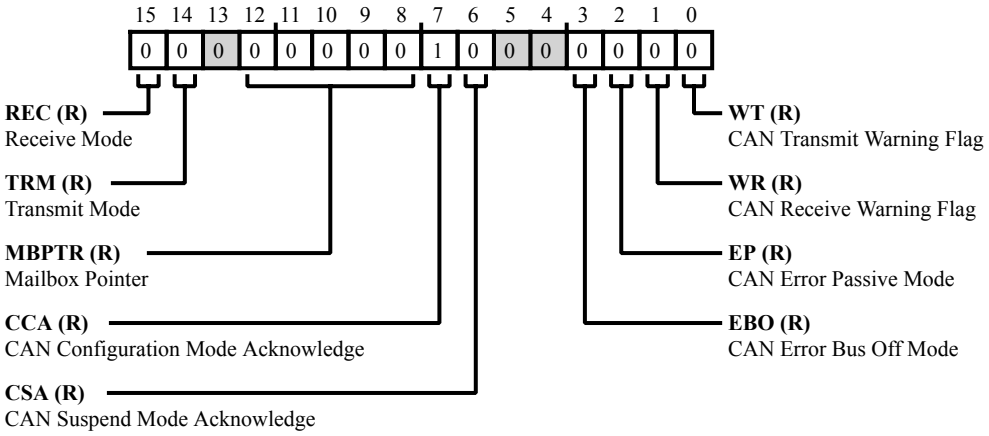

Figure 21-54: CAN_STAT Register Diagram

Table 21-49: CAN_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/NW)	REC	Receive Mode. The CAN_STAT.REC bit indicates whether the CAN is in receive mode.	
		0	Not in Receive Mode
		1	Receive Mode
14 (R/NW)	TRM	Transmit Mode. The CAN_STAT.TRM bit indicates whether the CAN is in transmit mode.	
		0	Not in Transmit Mode
		1	Transmit Mode
12:8 (R/NW)	MBPTR	Mailbox Pointer. The CAN_STAT.MBPTR bits represent the mailbox number of the current transmit message. After a successful transmission, these bits remain unchanged.	
		0-31	Processing Mailbox 0 to31 Message
7 (R/NW)	CCA	CAN Configuration Mode Acknowledge. The CAN_STAT.CCA bit indicates whether the CAN is in configuration mode.	
		0	Not in Configuration Mode
		1	Configuration mode

Table 21-49: CAN_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/NW)	CSA	CAN Suspend Mode Acknowledge. The CAN_STAT.CSA bit indicates whether the CAN is in suspend mode.	
		0	Not in Suspend Mode
		1	Suspend mode
3 (R/NW)	EBO	CAN Error Bus Off Mode. The CAN_STAT.EBO bit indicates whether the CAN is in error bus off mode.	
		0	TXECNT Below 256
		1	TXECNT Above Bus Off Limit
2 (R/NW)	EP	CAN Error Passive Mode. The CAN_STAT.EP bit indicates whether the CAN is in error passive mode.	
		0	TXECNT and RXECNT Below 128
		1	TXECNT or RXECNT Above EP Level
1 (R/NW)	WR	CAN Receive Warning Flag. The CAN_STAT.WR bit indicates whether the CAN has detected a receive warning flag condition.	
		0	RXECNT Below Limit
		1	RXECNT at Limit
0 (R/NW)	WT	CAN Transmit Warning Flag. The CAN_STAT.WT bit indicates whether the CAN detected a transmit warning flag condition.	
		0	TXECNT Below Limit
		1	TXECNT at Limit

Transmission Acknowledge 1 Register

The CAN_TA1 register indicates transmission success for mailboxes 8 through 15. Each bit in this register indicates transmission success for the corresponding mailbox when set (=1). Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

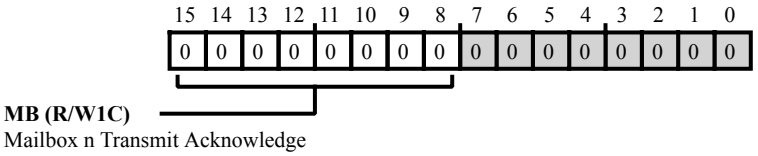

Figure 21-55: CAN_TA1 Register Diagram

Table 21-50: CAN_TA1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W1C)	MB	Mailbox n Transmit Acknowledge.

Transmission Acknowledge 2 Register

The CAN_TA2 register indicates transmission success for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register indicates transmission success for the corresponding mailbox when set (=1).

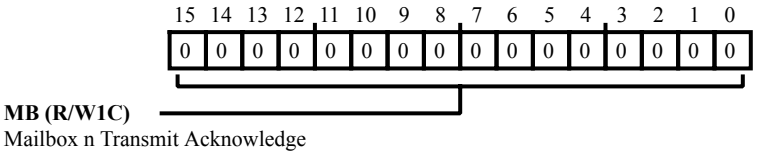

Figure 21-56: CAN_TA2 Register Diagram

Table 21-51: CAN_TA2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Transmit Acknowledge.

Timing Register

The CAN_TIMING register select the time segments, sampling, and synchronization for CAN bit timing. For more information about bit timing and clock operation, see the CAN Operating Modes section.

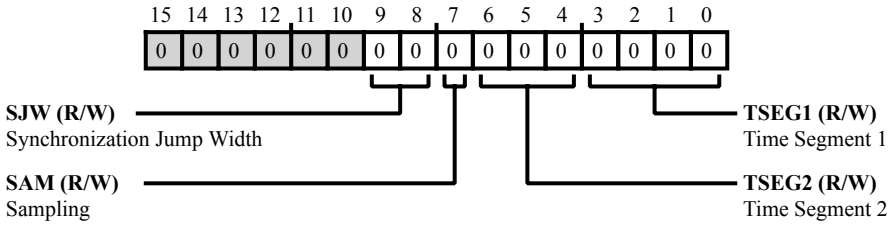

Figure 21-57: CAN_TIMING Register Diagram

Table 21-52: CAN_TIMING Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:8 (R/W)	SJW	Synchronization Jump Width. The CAN_TIMING.SJW bits select the maximum number of time quanta, ranging from 1 to 4(SJW + 1). This selection allows for a re-synchronization attempt when the CAN detects a recessive-to-dominant edge outside the synchronization segment. The re-synchronization automatically moves the sampling point such that the CAN bit is still handled properly. Note that the CAN_TIMING.SJW value should not exceed CAN_TIMING.TSEG2 or CAN_TIMING.TSEG1.
7 (R/W)	SAM	Sampling. The CAN_TIMING.SAM bit selects whether the CAN performs normal sampling (once at the sampling point described by the CAN_TIMING register) or performs over sampling. If CAN_TIMING.SAM is set, the CAN over samples the input signal at three times at the SYSCLK rate. The resulting value is generated by a majority decision of the three sample values. Note that the CAN_TIMING.SAM bit should always be cleared if the CAN_CLK.BRP value is less than 4.
6:4 (R/W)	TSEG2	Time Segment 2. The CAN_TIMING.TSEG2 bits and CAN_TIMING.TSEG1 bits control how many time quanta of which the CAN bits consist, resulting in the CAN bit rate. For more information about bit timing and clock operation, see the CAN Operating Modes section. Note that the CAN_TIMING.TSEG1 value should always be greater than or equal to the CAN_TIMING.TSEG2 value.
3:0 (R/W)	TSEG1	Time Segment 1. The CAN_TIMING.TSEG1 bits and CAN_TIMING.TSEG2 bits control how many time quanta of which the CAN bits consist, resulting in the CAN bit rate. For more information about bit timing and clock operation, see the CAN Operating Modes section. Note that the CAN_TIMING.TSEG1 value should always be greater than or equal to the CAN_TIMING.TSEG2 value.

Transmission Request Reset 1 Register

The CAN_TRR1 register requests transmit abort for mailboxes 8 through 15. Bits in this register request transmit abort for the corresponding mailbox when set (=1). When a transmission completes, the corresponding bits in the transmit request set register (CAN_TRS1) and in the CAN_TRR1 are cleared. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

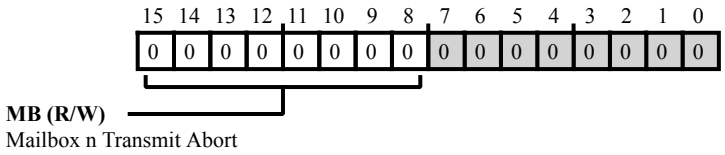

Figure 21-58: CAN_TRR1 Register Diagram

Table 21-53: CAN_TRR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	MB	Mailbox n Transmit Abort.

Transmission Request Reset 2 Register

The CAN_TRR2 register requests transmit abort for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register requests transmit abort for the corresponding mailbox when set (=1). When a transmission completes, the corresponding bits in the transmit request set register (CAN_TRS2) and in the CAN_TRR2 are cleared.

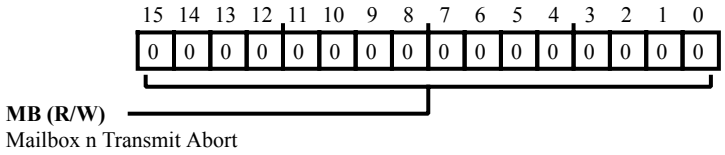

Figure 21-59: CAN_TRR2 Register Diagram

Table 21-54: CAN_TRR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Transmit Abort.

Transmission Request Set 1 Register

The CAN_TRS1 register requests transmit for mailboxes 8 through 15. Bits in this register request transmit for the corresponding mailbox when set (=1). After writing the data and the identifier into the mailbox area, the message is sent after mailbox n is enabled (with the corresponding bit in CAN_MC1 = 1}, and (subsequently) the corresponding transmit request bit is set (in CAN_TRS1). When a transmission completes, the corresponding bits in CAN_TRS1 and in the transmit request reset register (CAN_TRR1) are cleared. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

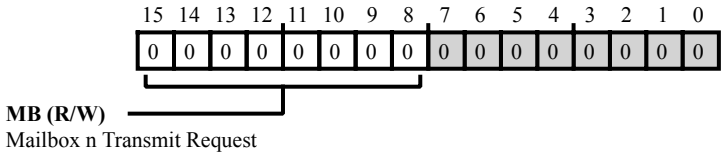

Figure 21-60: CAN_TRS1 Register Diagram

Table 21-55: CAN_TRS1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	MB	Mailbox n Transmit Request.

Transmission Request Set 2 Register

The CAN_TRS2 register requests transmit for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register requests transmit for the corresponding mailbox when set (=1). After writing the data and the identifier into the mailbox area, the message is sent after mailbox n is enabled (with the corresponding bit in CAN_MC2 = 1}, and (subsequently) the corresponding transmit request bit is set (in CAN_TRS2). When a transmission completes, the corresponding bits in CAN_TRS2 and in the transmit request reset register (CAN_TRR2) are cleared.

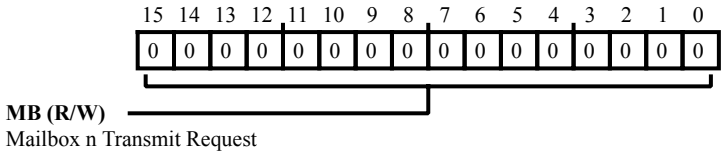

Figure 21-61: CAN_TRS2 Register Diagram

Table 21-56: CAN_TRS2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Transmit Request.

Universal Counter Configuration Mode Register

The CAN_UCCNF register controls the operation of the universal counter, including counter enable and counter mode selection.

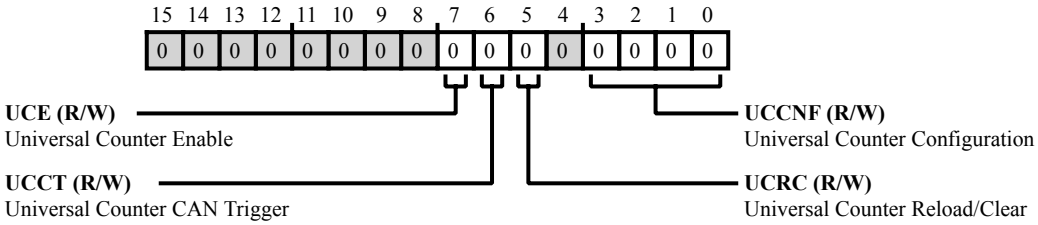

Figure 21-62: CAN_UCCNF Register Diagram

Table 21-57: CAN_UCCNF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	UCE	Universal Counter Enable. The CAN_UCCNF.UCE bit enables universal counter operation in the mode selected by the CAN_UCCNF.UCCNF bits.	
		0	Disable Counter
		1	Enable Counter
6 (R/W)	UCCT	Universal Counter CAN Trigger. The CAN_UCCNF.UCCT bit enables the universal counter trigger, directing the CAN to re-load the counter on mailbox 4 reception in watchdog mode and clear the counter on mailbox 4 reception in time stamp mode. This bit has no effect in all other modes.	
		0	Disable Trigger
		1	Enable Trigger
5 (R/W)	UCRC	Universal Counter Reload/Clear. The CAN_UCCNF.UCRC bit re-loads or clears the universal counter, depending on the counter mode. In watchdog mode, setting this bit directs the CAN to re-load the counter. In all other modes, setting this bit directs the CAN to clear the counter.	
		0	No Action
		1	Re-load or Clear the Counter
3:0 (R/W)	UCCNF	Universal Counter Configuration. The CAN_UCCNF.UCCNF bits select the universal counter operating mode. For more information about these modes, see the Operating Modes section.	
		0	Reserved
		1	Time Stamp Mode
		2	Watchdog Mode

Table 21-57: CAN_UCCNF Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		3	Auto-transmit Mode
		4	Reserved
		5	Reserved
		6	Count Error Frames
		7	Count Overload Frames
		8	Count Arbitration Lost
		9	Count Aborted Transmissions
		10	Count Successful Transmissions
		11	Count Rejected Receive Messages
		12	Count Receive Message Lost
		13	Count Successful Receptions
		14	Count Stored Receptions
		15	Count Valid Messages

Universal Counter Register

The CAN_UCCNT register holds the current universal count. This register is re-loaded from the CAN_UCRC register when the decrements to zero in auto-transmit mode.

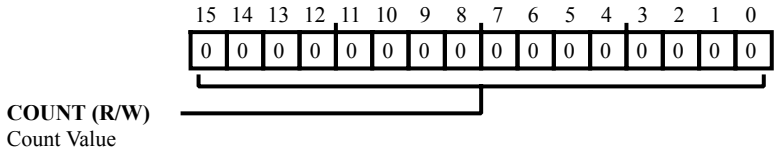

Figure 21-63: CAN_UCCNT Register Diagram

Table 21-58: CAN_UCCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	COUNT	Count Value. The CAN_UCCNT.COUNT bits hold the current universal count value.

Universal Counter Reload/Capture Register

The CAN_UCRC register holds the period value (universal count), which is used in auto-transmit mode as the period for sending the message in mailbox 11 (broadcast heartbeat) to all CAN nodes. Accordingly, messages sent this way usually have high priority.

The period value is written to the CAN_UCRC register. When auto-transmit mode is enabled (CAN_UCCNF.UCCNF = 0x3), the CAN loads the counter with the value in CAN_UCRC. The counter decrements to 0 at the CAN bit clock rate, then is reloaded. Each time the counter decrements to 0, the CAN sets the CAN_TRS1.MB bit for mailbox 11 and sends the corresponding message from mailbox 11.

Note that for auto-transmit mode, mailbox 11 must be configured as a transmit mailbox and must contain valid data (identifier, control bits, and data). This setup must occur before the counter first expires after this mode is enabled.

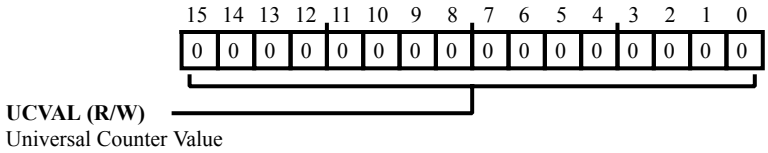

Figure 21-64: CAN_UCRC Register Diagram

Table 21-59: CAN_UCRC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	UCVAL	Universal Counter Value. The CAN_UCRC.UCVAL bits hold the value for the universal count period, which is used in auto-transmit mode.

22 Universal Serial Bus (USB)

The USB OTG controller provides a low-cost connectivity solution for consumer mobile devices such as cell phones, digital still cameras, and MP3 players. It allows these devices to transfer data using a point-to-point USB connection without the need for a personal computer host.

The USB controller can operate in a traditional USB peripheral-only mode as well as the host mode presented in the On-The-Go (OTG) supplement. to the USB 2.0 SpecificationThe USB module supports:

- Host mode transfers at full-speed (12 Mbp/sec) rate
- Host mode transfers at low-speed (1.5 Mbp/sec) rates. The connection to low-speed devices is only possible through a full-speed hub.
- Peripheral mode transfers at full-speed (12 Mbp/sec) rate

The USB controller uses a peripheral bus slave interface to access its control and status registers as well as read and write to the endpoint packet buffers. Data transfers to and from the USB controller through the 3 transmit and 3 receive endpoint FIFOs (EP1 – EP3), providing a total of 6 data endpoints.

USB Features

The USB controller provides the following features:

- Low speed and full speed rates supported
- One bidirectional control endpoint
- Three transmit and three receive unidirectional endpoints
- 2624 byte of FIFO space for packet buffering
- Seven DMA master channels
- Two top-level maskable general-purpose interrupts
- Low-power wake-up on activity
- VBUS control interrupts for external analog VBUS control
- Session request protocol (SRP) and host negotiation protocol (HNP) capability

- Host transaction scheduling in hardware
- Soft connect or disconnect feature
- Full-speed physical layer UTMI+ level 3 interface for on-chip PHY
- Backwards compatible with existing USB 1.1 hosts

Only device requirements or system bandwidth limit the number of active endpoints at one time because each endpoint operates independently from the next. Software determines the type of transfer for each endpoint individually and also the manner in which it is transferred between the USB controller and memory (DMA or interrupt-based). The USB uses endpoint zero solely for receive and transmit control transfer. These transfers are used for device configuration and information gathering.

USB Functional Description

The following sections describe the function of the USB OTG interface.

USB Architectural Concepts

The USB controller operates in either of two USB operation modes (peripheral or host mode) at a given time.

In peripheral mode, the USB controller encodes, decodes, checks, and directs all USB packets sent and received, responding appropriately to host requests. Data is transferred from the processor core memory into the Tx FIFOs of the device onto USB as IN packets. In the other direction, USB OUT packets are received into the Rx FIFOs (having been sent from the host) and transferred to system memory for processing or storage. In peripheral mode, the USB controller acts as a slave device to another USB host; either a personal computer or another OTG host controller.

When operating in host mode, the USB controller uses simple hosting capabilities to master point-to-point connections with another USB peripheral, initiating transfers on the bus for the peripheral to respond. USB IN packets are received into the Rx FIFOs for transfer into the processor core memory. Data written into Tx FIFOs is transmitted onto the bus as USB OUT packets. In this mode, the USB controller encodes, decodes, and checks USB packets sent and received. The controller automatically schedules isochronous and interrupt transfers from the endpoint buffers. It performs one transaction every *n* frames, where n represents the polling interval programmed for the endpoint.

Any of the endpoints can be programmed to be written to or read from using the DMA master channels. This configuration provides the most efficient means of transferring data between the controller and on-chip memory.

USB endpoints 0 through 3 have DMA interrupt lines (`USB_DMA_IRQ`) providing a total of seven DMA request lines.

The USB provides two top-level maskable interrupts, each of which can be sourced from any or all of transmit endpoint status, receive endpoint status or global USB status. See Interrupt Signals for details.

The RAM interface of the USB controller supports a single block of synchronous single-port RAM used to buffer the USB packets.

2624 bytes of SRAM are available.

The UTMI+ level 3 PHY interface provides a means of connecting a selection of full-speed PHYs to the controller, from device-only PHYs through full OTG-compliant PHYs.

For details of the PHY interface, See UTMI Interface.

ATTENTION: Check the processor data sheet for requirements regarding minimum system clock frequency needed for proper USB operation.

The USB controller is configured as either a USB OTG A device or B device depending on the type of plug inserted into its USB receptacle. The state of the `USB_ID` (connector ID) pin determines this configuration.

The USB controller uses an asynchronous wake-up circuit to detect when another B device is asserting its D+ pull-up. This activity initiates the SRP (session request protocol) when all other clocks are off.

Use of the controller for OTG functionality requires the capability to:

- Drive VBUS (as a default A device powering the bus)
- Discharge VBUS (speeding up the time for VBUS to fall below the *SessionEnd* threshold as a B device checking initial conditions)
- Charge VBUS to 2.1 V (when initiating SRP as a B device).

The UTMI interface drives these controls, but the controller also provides a separate interrupt register, `USB_VBUS_CTL`, which represents the drive VBUS, discharge VBUS, and charge VBUS signaling. See the register section for more information on these controls.

Multi-Point Support

The USB controller has the facility, when operating in host mode, to act as the host to a range of USB peripheral devices.

Full-speed or low-speed devices connect to the USB controller through a USB hub.

The USB controller, as part of its support for multiple devices, permits individual allocation of the functions of the target to the different Rx and Tx endpoints implemented. Furthermore, the USB controller can make this allocation dynamically, allowing the devices from the targeted peripheral list to be used in different combinations. The numbers of Tx and Rx endpoints implemented in the controller limit the combinations of peripheral devices that can be used together. Devices can only be added where the required endpoints remain available.

On-Chip Bus Interfaces

The USB controller uses two 32-bit wide independent bus interfaces, a master and a slave, to communicate with a processor-based subsystem. The slave interface allows the processor core to access the control and status registers (including DMA master registers) and the endpoint FIFOs. The integrated DMA uses the master interface to drive data into or out of the endpoint FIFOs with minimal processor core interaction. For more information, see USB Block Diagram.

FIFO Configuration

Each bidirectional endpoint (provided as two unidirectional endpoints) has its own endpoint number (0 for control, 1 on up for data transfer). Although two endpoints could use the same number, the endpoints can support different transfer types. Each of these bidirectional endpoints has a fixed region of the SRAM in the USB controller to which it has access. This feature dictates to some extent the types of transfers that can be used for that particular endpoint. This restriction follows from the maximum size of USB packets, which varies with each transfer type. The ***FIFO Sizes and Transfer Types*** table lists the endpoint FIFO configuration, with an indication of the transfer types possible for that particular buffer size.

Table 22-1: FIFO Sizes and Transfer Types

Bidirectional Endpoint (Rx and Tx)	FIFO Size (each direction)	USB Transfer Types
0	64 bytes	Size fixed for control transfers
1–2	128 bytes	Bulk, Interrupt, Isochronous
3	1024 bytes	Bulk, Interrupt, Isochronous

Each endpoint FIFO can buffer one or two packets (in double-buffered mode). Double-buffering is recommended for most applications to improve efficiency by reducing the frequency of servicing for each endpoint.

Double-buffering bulk transactions means that data transfers over the USB are not slowed when packets are loaded or unloaded from the FIFO in the time it takes to transfer a packet. Double-buffering isochronous transactions allows more time to load or unload the FIFO. It also allows the usage of the SOF interrupt to service the endpoint rather than the endpoint interrupt. This functionality has the following advantages:

- Easy detection of lost packets
- Regular interrupt timing (making it easier to source or sink the data)
- If the USB controller uses more than one isochronous endpoint, one interrupt can service all the endpoints.

The USB controller uses the transmit or receive FIFO address registers to specify the address of each endpoint FIFO.

Clocking

The USB controller uses the system clock SYSCLK to generate an internal clock (CLK) used to clock the USB registers.

For proper operation, refer to device datasheet for minimum system clock SYSCLK value.

The transceiver clock (XCLK) is a 60-MHz clock sourced from the UTMI PHY. The PHY interface logic and USB engine use the clock. This controller does not require an external USB clock, instead this clock is derived from the processor PLL (`CGU_DIV.SYSCLK`). The clock SYSCLK must be 60 MHz for USB operation. For more information, see the Clock Generation Unit (CGU) chapter.

For the ADSP-CM40x, the *DCLK* signal of the CGU is equivalent to the *USBCLK* signal.

NOTE: For best performance (best signal integrity), follow the guidelines in the data sheet for selecting an input clock frequency.

When the controller is in the SUSPEND state and when no session is active, the clock and much of the USB controller is stopped to reduce power consumption. The clock becomes operational again when RESUME signaling is detected on the USB lines.

UTMI Interface

The interface to the on-chip PHY uses the industry-standard UTMI+ (universal transceiver macro interface) level 3.

This interface provides full-speed device and OTG functionality and supports communication to a hub.

The PHY is a mixed-signal block and includes the following:

- Full-speed drivers and receivers (single-ended and differential)
- Full-speed CDR
- Full-speed shift registers, NRZI encode or decode and bit-stuff encode or decode
- Data line pull-up and pull-down resistors
- VBUS and `USB_ID` level detection
- Host disconnect detection

Although the UTMI specification indicates that VBUS charging, driving and discharging happen inside the PHY, for process-restricting and power reasons, implement these functions off-chip in a separate USB charge-pump chip.

ADSP-CM40x USB Register List

The Universal Serial Bus controller (USB) is a multi-point high-speed dual-role USB 2.0-compliant controller. The USB controller can operate in a traditional USB peripheral-only mode as well as the host mode presented in the On-The-Go (OTG) supplement to the USB 2.0 Specification, Rev 1.0a; June 24, 2003; USB-IF. A set of registers governs USB controller operations. For more information on USB controller functionality, see the USB controller register descriptions.

Table 22-2: ADSP-CM40x USB Register List

Name	Description
`USB_DEV_CTL`	Device Control Register
`USB_DMA[n]_ADDR`	DMA Channel n Address Register
`USB_DMA[n]_CNT`	DMA Channel n Count Register
`USB_DMA[n]_CTL`	DMA Channel n Control Register
`USB_DMA_IRQ`	DMA Interrupt Register
`USB_EP0I_CFGDATA[N]`	EP0 Configuration Information Register

Table 22-2: ADSP-CM40x USB Register List (Continued)

Name	Description
USB_EP0I_CNT[N]	EP0 Number of Received Bytes Register
USB_EP0I_CSR[N]_H	EP0 Configuration and Status (Host) Register
USB_EP0I_CSR[N]_P	EP0 Configuration and Status (Peripheral) Register
USB_EP0I_NAKLIMIT[N]	EP0 NAK Limit Register
USB_EP0I_TYPE[N]	EP0 Connection Type Register
USB_EP0_CFGDATA[n]	EP0 Configuration Information Register
USB_EP0_CNT[n]	EP0 Number of Received Bytes Register
USB_EP0_CSR[n]_H	EP0 Configuration and Status (Host) Register
USB_EP0_CSR[n]_P	EP0 Configuration and Status (Peripheral) Register
USB_EP0_NAKLIMIT[n]	EP0 NAK Limit Register
USB_EP0_TYPE[n]	EP0 Connection Type Register
USB_EPINFO	Endpoint Information Register
USB_EPI[N]_RXCNT	EPn Number of Bytes Received Register
USB_EPI[N]_RXCSR_H	EPn Receive Configuration and Status (Host) Register
USB_EPI[N]_RXCSR_P	EPn Receive Configuration and Status (Peripheral) Register
USB_EPI[N]_RXINTERVAL	EPn Receive Polling Interval Register
USB_EPI[N]_RXMAXP	EPn Receive Maximum Packet Length Register
USB_EPI[N]_RXTYPE	EPn Receive Type Register
USB_EPI[N]_TXCSR_H	EPn Transmit Configuration and Status (Host) Register
USB_EPI[N]_TXCSR_P	EPn Transmit Configuration and Status (Peripheral) Register
USB_EPI[N]_TXINTERVAL	EPn Transmit Polling Interval Register
USB_EPI[N]_TXMAXP	EPn Transmit Maximum Packet Length Register
USB_EPI[N]_TXTYPE	EPn Transmit Type Register
USB_EPI_FIFOSZ	FIFO Size
USB_EP[n]_FIFOSZ	FIFO Size
USB_EP[n]_RXCNT	EPn Number of Bytes Received Register
USB_EP[n]_RXCSR_H	EPn Receive Configuration and Status (Host) Register
USB_EP[n]_RXCSR_P	EPn Receive Configuration and Status (Peripheral) Register
USB_EP[n]_RXINTERVAL	EPn Receive Polling Interval Register
USB_EP[n]_RXMAXP	EPn Receive Maximum Packet Length Register
USB_EP[n]_RXTYPE	EPn Receive Type Register

Table 22-2: ADSP-CM40x USB Register List (Continued)

Name	Description
USB_EP[n]_TXCSR_H	EPn Transmit Configuration and Status (Host) Register
USB_EP[n]_TXCSR_P	EPn Transmit Configuration and Status (Peripheral) Register
USB_EP[n]_TXINTERVAL	EPn Transmit Polling Interval Register
USB_EP[n]_TXMAXP	EPn Transmit Maximum Packet Length Register
USB_EP[n]_TXTYPE	EPn Transmit Type Register
USB_FADDR	Function Address Register
USB_FIFOB[n]	FIFO Byte (8-Bit) Register
USB_FIFOH[n]	FIFO Half-Word (16-Bit) Register
USB_FIFO[n]	FIFO Word (32-Bit) Register
USB_FRAME	Frame Number Register
USB_FS_EOF1	Full-Speed EOF 1 Register
USB_IDCTL	ID Control
USB_IEN	Common Interrupts Enable Register
USB_INDEX	Index Register
USB_INTRRX	Receive Interrupt Register
USB_INTRRXE	Receive Interrupt Enable Register
USB_INTRTX	Transmit Interrupt Register
USB_INTRTXE	Transmit Interrupt Enable Register
USB_IRQ	Common Interrupts Register
USB_LINKINFO	Link Information Register
USB_LPM_ATTR	LPM Attribute Register
USB_LPM_CTL	LPM Control Register
USB_LPM_FADDR	LPM Function Address Register
USB_LPM_IEN	LPM Interrupt Enable Register
USB_LPM_IRQ	LPM Interrupt Status Register
USB_LS_EOF1	Low-Speed EOF 1 Register
USB_MP[n]_RXFUNCADDR	MPn Receive Function Address Register
USB_MP[n]_RXHUBADDR	MPn Receive Hub Address Register
USB_MP[n]_RXHUBPORT	MPn Receive Hub Port Register
USB_MP[n]_TXFUNCADDR	MPn Transmit Function Address Register
USB_MP[n]_TXHUBADDR	MPn Transmit Hub Address Register

Table 22-2: ADSP-CM40x USB Register List (Continued)

Name	Description
USB_MP[n]_TXHUBPORT	MPn Transmit Hub Port Register
USB_PHY_CTL	FS PHY Control
USB_PHY_STAT	FS PHY Status
USB_POWER	Power and Device Control Register
USB_RAMINFO	RAM Information Register
USB_RQPKTCNT[n]	EPn Request Packet Count Register
USB_RXDPKTBUFDIS	RX Double Packet Buffer Disable for Endpoints 1 to 3
USB_SOFT_RST	Software Reset Register
USB_TESTMODE	Testmode Register
USB_TXDPKTBUFDIS	TX Double Packet Buffer Disable for Endpoints 1 to 3
USB_VBUS_CTL	VBUS Control Register
USB_VPLEN	VBUS Pulse Length Register

ADSP-CM40x USB Interrupt List

Table 22-3: ADSP-CM40x USB Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
106	USB0_STAT	USB0 Status/FIFO Data Ready	Level	
107	USB0_DATA	USB0 DMA Status/Transfer Complete	Level	

ADSP-CM40x USB Trigger List

Table 22-4: ADSP-CM40x USB Trigger List Masters

Trigger ID	Name	Description	Sensitivity
52	USB0_DATA	USB0 DMA Status/Transfer Complete	Level

Table 22-5: ADSP-CM40x USB Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

USB Block Diagram

The *USB OTG Controller Block Diagram* shows the functional blocks within the USB. For more information about the blocks, see the USB Functional Description.

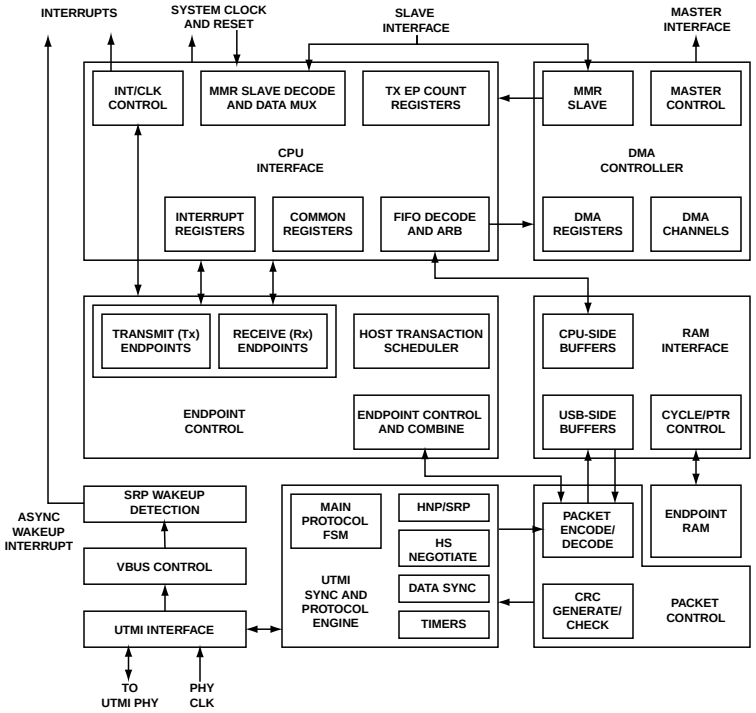

Figure 22-1: USB OTG Controller Block Diagram

USB Definitions

A list of common USB terms and their definitions as used in this specification and based on the USB controller follows:

'A' Device

The USB device with a mini-A plug inserted into its receptacle. The A device always supplies power to VBUS.

'B' Device

The USB device with a standard-B or mini-B plug inserted into its receptacle. The B device starts a session as the peripheral.

Bidirectional endpoint

An endpoint that can concurrently support both receive and transfer packets.

Control endpoint

An endpoint used only for transfer of USB control packets for setup and configuration. In all USB devices, the control endpoint refers to the bidirectional endpoint 0.

Dual role device

A USB device that can operate either as the USB host in an OTG session or as a traditional USB peripheral.

Endpoint

A single physical communication channel for USB, implemented as a FIFO and control logic for that endpoint. Each endpoint has an associated USB transfer type, maximum packet size, bandwidth requirement, endpoint number, and (often) a fixed transfer direction.

Frame

A regular, fixed 1 ms timeslot that can contain several transactions. The transfer type determines the permissible transactions for a given endpoint.

HNP

Host negotiation protocol. Part of the USB OTG supplement that allows the host function to be transferred between two connected dual role devices.

Packet

The lowest level of data exchange on USB. The transfer type and buffer size of the USB peripheral determine the size.

PHY

The PHY is a transceiver circuit that implements the physical layer of USB. For full speed USB OTG, this circuit includes line drivers and receivers, pull-up, or pull-down resistors as well as device ID and VBUS level detection.

Session

A period during which USB transfers take place within an OTG connection. The A device (drives VBUS) or B device (initiates SRP) can initiate this period. VBUS is powered during a session.

SRP

Session request protocol. Part of the USB OTG supplement that allows a B device to turn on VBUS and initiate a USB session.

Transaction

Collection of one or more packets in sequence

Transfer

Collection of one or more transfers in sequence

Unidirectional endpoint

Endpoint with its direction fixed in a single direction (for example, it can only receive packets from the USB) in both host and peripheral modes.

USB References

The following references provide further information regarding the USB.

- *On-The-Go Supplement to the USB 2.0 Specification*, Rev 1.0a, June 24, 2003, USB-IF
- *Universal Serial Bus Specification 2.0*

USB Operating Modes

The USB OTG interface can operate in peripheral mode or host mode.

When the USB controller operates in peripheral mode, the controller can be attached to a conventional host (such as a personal computer) or another OTG device operating in host mode. The second device can be high speed or full speed. When linked to another peripheral device, the USB controller can also act as the host. If the other device is also a dual role controller, the two devices can switch roles, as needed.

The role the USB controller takes depends on the way the devices are cabled together. Each USB cable has an A and a B device end. If the A end of the cable is plugged into the device containing the USB controller, the USB controller takes the role of the host device. It goes into host mode (in this case, the `USB_DEV_CTL.HOSTMODE` bit is set to 1). If the B of the cable is plugged in, the USB controller goes instead into peripheral mode (and the `USB_DEV_CTL.HOSTMODE` bit remains at 0).

When both devices contain dual role controllers, signaling can be used to switch the roles of the two devices, without switching the cable connecting the two devices. See Host Negotiation Protocol for details on the conditions under which the USB controller can switch between peripheral and host mode.

NOTE: The multi-point capability of the USB controller is associated with a range of registers recording the allocation of device functions to individual endpoints and device function characteristics. These characteristics include endpoint number, operating speed, and transaction type on an endpoint-by-endpoint basis. These registers are principally associated with the use of the USB controller as the host to a number of devices. However, set the registers when the core is used as the host for a single target device.

To enable the USB:

1. Configure the USB PLL multiplier settings in the USB PLL control register. Check the processor data sheet for the requirements for input clock frequency.

Peripheral Mode

USB OTG interface operations for the peripheral mode differ from host mode in a number of ways. The following sections describe peripheral mode operations.

Endpoint Setup

In peripheral mode, the USB uses a few endpoint-specific configuration bits when setting up an endpoint for transfer for all types of peripheral transfer. The configuration determines how the processor core interacts with the endpoint FIFO.

One key parameter required before a transfer can occur through an endpoint is the maximum USB packet size that the endpoint can support. The software sets this value. It depends on various system constraints. These constraints include the size of hardware FIFO available and system latencies as well as the USB transfer type and class used. The `USB_EP[n]_TXMAXP` or `USB_EP[n]_RXMAXP` registers define the maximum amount of data that can be transferred to the selected endpoint in a single frame. The value must match the programmed maximum individual packet size (*MaxPktSize*) of the standard endpoint descriptor for the endpoint.

For transmit endpoints, program the maximum packet size using the `USB_EP[n]_TXMAXP`. For receive endpoints, the USB uses the `USB_EP[n]_RXMAXP` register. The maximum packet size must not exceed the actual hardware endpoint FIFO size.

The sizes of the transmit or receive FIFOs for endpoints 1 to 3 are fixed. The FIFOs operate in single buffered mode when the maximum packet size is equal or greater than half the size of the FIFO. The FIFOs operate in double buffered mode when the maximum packet size is less than half the size of the FIFO.

Because the USB controller uses a 32-bit interface, choose an even number for the value of *MaxPktSize*. This selection simplifies transferring data between FIFOs and the processor core.

Configure more setup parameters using the `USB_EP[n]_TXCSR_H` or `USB_EP[n]_RXCSR_H` register (depending on whether the endpoint in question is for receive or transmit). The USB uses the `USB_EP[n]_RXCSR_H.DMAREQEN` bit in this register to enable the assertion of the appropriate DMA request whenever the endpoint is able to receive or transmit another packet. The USB uses the `USB_EP[n]_RXCSR_H.AUTOCLR` and `USB_EP[n]_RXCSR_H.AUTOREQ` bits to set the FIFO ready triggers (`USB_EP[n]_RXCSR_H.RXPKTRDY` and `USB_EP[n]_TXCSR_H.TXPKTRDY`) automatically whenever a packet is transferred to streamline DMA operation for transfers that span multiple packets. Note, however, that the USB cannot use `USB_EP[n]_RXCSR_H.AUTOCLR` and `USB_EP[n]_RXCSR_H.AUTOREQ` bits with high-bandwidth endpoints. Refer to the “Register Descriptions” section for more details on the endpoint control and status registers.

IN Transactions as a Peripheral

When the USB controller operates in peripheral mode, the transmit FIFOs handle data for IN transactions. The maximum size of data packet that can be placed in a FIFO for a transmit endpoint is programmable. When applicable, the value written to the `USB_EP[n]_TXMAXP` register for that endpoint determines the size (maximum payload multiplied by the number of transactions per micro-frame).

The maximum packet size set for any endpoint must not exceed the FIFO size. (See FIFO Configuration.)

ATTENTION: Do not write to the USB_EP[n]_TXMAXP register while there is data in the FIFO, as unexpected results can occur.

The following sections describe the two types of packet buffering used for IN transactions.

Single packet buffering. Set the USB_EP[n]_TXCSR_P.TXPKTRDY bit as each packet for transmission is loaded into the transmit FIFO. If the USB_EP[n]_TXCSR_P.AUTOSET bit is set, the USB_EP[n]_TXCSR_P.TXPKTRDY bit is automatically set when a maximum-sized packet is loaded into the FIFO. For packet sizes less than the maximum, and where auto-set cannot be used (high-bandwidth isochronous or interrupt transactions), always set the USB_EP[n]_TXCSR_P.TXPKTRDY bit manually (for example by the processor core).

When the USB_EP[n]_TXCSR_P.TXPKTRDY bit is set, either manually or automatically, the USB_EP[n]_TXCSR_P.NEFIFO bit is also set and the packet is ready to be sent. When the packet is successfully sent, both the USB_EP[n]_TXCSR_P.TXPKTRDY and USB_EP[n]_TXCSR_P.NEFIFO bits are cleared. The USB controller generates the appropriate transmit endpoint interrupt (if enabled). The next packet can then be loaded into the FIFO.

Double packet buffering. Set the USB_EP[n]_TXCSR_P.TXPKTRDY bit as each packet for tranmission is loaded into the transmit FIFO. If the USB_EP[n]_TXCSR_P.AUTOSET bit is set, the USB_EP[n]_TXCSR_P.TXPKTRDY bit is automatically set when a maximum-sized packet is loaded into the FIFO. For packet sizes less than the maximum, always set the USB_EP[n]_TXCSR_P.TXPKTRDY bit manually (for example, set by the processor core).

When the USB_EP[n]_TXCSR_P.TXPKTRDY bit is set, either manually or automatically, the USB_EP[n]_TXCSR_P.NEFIFO bit also is set. The USB_EP[n]_TXCSR_P.TXPKTRDY bit is then immediately cleared (and an interrupt generated, if enabled). A second packet can now be loaded into the transmit FIFO and the USB_EP[n]_TXCSR_P.TXPKTRDY bit is set again (either manually or automatically if the packet is the maximum size). Both packets are now ready for transmission.

When the first packet is successfully sent, the USB_EP[n]_TXCSR_P.TXPKTRDY bit is cleared. The USB controller generates the appropriate transmit endpoint interrupt (if enabled) to signal that another packet can now be loaded into the transmit FIFO. The state of the USB_EP[n]_TXCSR_P.NEFIFO bit indicates how many packets can be loaded. If the USB_EP[n]_TXCSR_P.NEFIFO bit is set, then there is another packet in the FIFO and only one more packet can be loaded. If the USB_EP[n]_TXCSR_P.NEFIFO bit is cleared, then there are no packets in the FIFO and two more packets can be loaded.

OUT Transactions as a Peripheral

When the USB controller operates in peripheral mode, the receive FIFOs handle data for OUT transactions.

The value written to the USB_EP[n]_RXMAXP register for an endpoint determines the maximum amount of data received by a receive endpoint in any frame. The value is programmable. The maximum packet size must not exceed the FIFO size.

If the size of the receive endpoint FIFO is less than twice the maximum packet size for this endpoint, only one data packet can be buffered in the FIFO. Single buffering is selected. (The size is set in the USB_EP[n]_RXMAXP register). When a packet is received and placed in the receive FIFO, the USB_EP[n]_RXCSR_P.RXPKTRDY bit and the USB_EP[n]_RXCSR_P.FIFOFULL bit are set. The USB controller generates the appropriate receive endpoint

interrupt (if enabled) to signal that a packet can now be unloaded from the FIFO. After the packet is unloaded, clear the USB_EP[n]_RXCSR_P.RXPKTRDY bit to allow reception of more packets. If the USB_EP[n]_RXCSR_P.AUTOCLR bit is set and a maximum-sized packet is unloaded from the FIFO, the USB_EP[n]_RXCSR_P.RXPKTRDY bit is cleared automatically. The USB_EP[n]_RXCSR_P.FIFOFULL bit is also cleared. For packet sizes less than the maximum, clear the USB_EP[n]_RXCSR_P.RXPKTRDY bit manually (for example by the processor core).

If double packet buffering is enabled, then two data packets can be buffered. When the first packet for reception is loaded into the receive FIFO, the USB_EP[n]_RXCSR_P.RXPKTRDY bit is set. The USB controller generates the appropriate receive endpoint interrupt (if enabled) to signal that a packet can now be unloaded from the FIFO. The USB_EP[n]_RXCSR_P.FIFOFULL bit is not set. This bit is only set if a second packet is received and loaded into the receive FIFO.

After the first packet is unloaded, clear the USB_EP[n]_RXCSR_P.RXPKTRDY bit to allow reception of further packets. If the USB_EP[n]_RXCSR_P.AUTOCLR bit is set and a maximum-sized packet is unloaded from the FIFO, the USB_EP[n]_RXCSR_P.RXPKTRDY bit is cleared automatically. For packet sizes less than the maximum, clear the USB_EP[n]_RXCSR_P.RXPKTRDY bit manually (for example by the processor core).

If the USB_EP[n]_RXCSR_P.FIFOFULL bit is set to 1 when USB_EP[n]_RXCSR_P.RXPKTRDY is cleared, the USB controller first clears the USB_EP[n]_RXCSR_P.FIFOFULL bit. The controller then sets the USB_EP[n]_RXCSR_P.RXPKTRDY bit again, indicating that there is another packet waiting in the FIFO for unloading.

Peripheral Transfer Work Flows

The USB transfer types (control, bulk, isochronous, and interrupt transfers) have different system requirements as well as individual USB transfer-specific features. Software handles each type differently. There is no uniform way of doing transfers across all transfer types using the USB controller.

The following sections provide some guidelines for peripheral mode transfer flows for each of the transfer types, in both IN (transmit) and OUT (receive) directions. For bulk endpoints, the optimal transfer flow depends on whether the final size of the transfer is known or unknown. The USB driver class in use determines whether the transfer size is known or not. Some drivers define the complete transfer size, and others operate on a packet-by-packet basis using a short packet to denote the end of a transfer. (A short packet is less than the value configured in the USB_EP[n]_TXMAXP register or less than the value configured in the USB_EP[n]_RXMAXP register).

Each of the work flows uses the following common steps.

1. Configure the endpoint control and status registers and the USB_EP[n]_TXMAXP or USB_EP[n]_RXMAXP value.
2. Configure the appropriate data transfer mechanism (DMA or interrupt setup).
3. Data transfer occurs.

The work flows do not describe the actions of the USB controller immediately preceding the endpoint setup. (For example, the reception of an IN/OUT token from the host, token validity checking, or NAK generation, among

others). Note also that there is no error-handling contained in the work flows (for example, checking the USB_EP[n]_RXCSR_P.FIFOFULL bit before writing data).

The proceeding sections use terms packets, frames, and transfers with their strict USB definitions (see USB Definitions).

Control Transactions as a Peripheral

Endpoint 0 is the main control endpoint of the USB controller. As such, the routines required to service endpoint 0 are more complicated than the routines required to service other endpoints.

The software is required to handle all the standard device requests that the USB controller sends or receives through endpoint 0. The *Universal Serial Bus Specification*, Revision 2.0, Chapter 9 describes the requirements. The protocol for these device requests involves different numbers and types of transactions per transfer. To accommodate this functionality, the processor must take a state machine approach to command decoding and handling.

The standard device requests a USB peripheral receives fits into three categories:

- Zero data requests (in which the command includes all the information)
- Write requests (in which extra data follows the command)
- Read requests (in which the device sends data back to the host)

The following sections describe the sequence of actions that the software must perform to process these different types of device request.

Write Requests

The host sends an 8-byte command followed by a write request that contains an extra packet (or packets) of data. An example of a write standard device request is SET_DESCRIPTOR.

As with all requests, the sequence of events begins when the software receives an endpoint 0 interrupt. The USB_EP[n]_RXCSR_P.RXPKTRDY bit is also set. The host then reads and decodes the 8-byte command from the endpoint 0 FIFO.

As with a zero data request, write to the USB_EP0_CSR[n]_P register to set the USB_EP0_CSR[n]_P.SPKTRDY bit. (The event indicates that the host read the command from the FIFO). But, in this case, do not set the USB_EP0_CSR[n]_P.DATAEND bit (indicating that more data is expected).

When a second endpoint 0 interrupt is received, the USB_EP0_CSR[n]_P register is read to check the endpoint status. The USB_EP0_CSR[n]_P.RXPKTRDY bit is set to indicate that a data packet is received. Read the USB_EP0_CNT[n] register to determine the size of this data packet. The data packet can then be read from the endpoint 0 FIFO.

If the length of the data associated with the request is greater than the maximum packet size for endpoint 0, the host sends more data packets. (The WLENGTH field in the command indicates the length of the data). In this case, the USB_EP0_CSR[n]_P.SPKTRDY bit is set, but do not set the USB_EP0_CSR[n]_P.DATAEND bit.

When all the expected data packets have been received, software writes to the USB_EP0_CSR[n]_P register to set the USB_EP0_CSR[n]_P.SPKTRDY bit and to set the USB_EP0_CSR[n]_P.DATAEND bit (indicating that no more data is expected).

When the host moves to the status stage of the request, software generates another endpoint 0 interrupt to indicate that the request has completed. No further action is required from the software; the interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or cannot be executed, then when the host decodes it, software must write to the USB_EP0_CSR[n]_P register. This operation sets the USB_EP0_CSR[n]_P.SPKTRDY bit and the USB_EP0_CSR[n]_P.SENDSTALL bit. When the host sends more data, the USB controller sends a stall to tell the host that the request was not executed. Software generates an endpoint 0 interrupt and the USB_EP0_CSR[n]_P.SENTSTALL bit is set.

If the host sends more data after the USB_EP0_CSR[n]_P.DATAEND has been set, then the USB controller sends a stall. Software generates an endpoint 0 interrupt and the USB_EP0_CSR[n]_P.SENTSTALL bit is set.

Read Requests

The function sends the 8-byte command followed by read requests containing a packet (or packets) of data to the host. Examples of standard device requests for read are:

- GET_CONFIGURATION
- GET_INTERFACE
- GET_DESCRIPTOR
- GET_STATUS
- SYNCH_FRAME

As with all requests, the sequence of events begins when the software receives an endpoint 0 interrupt. The USB_EP[n]_RXCSR_P.RXPKTRDY bit is also set. The host then reads and decodes the 8-byte command from the endpoint 0 FIFO. Write the USB_EP0_CSR[n]_P.SPKTRDY bit (indicating that the command has been read from the FIFO).

The data to transmit to the host is written to the endpoint 0 FIFO. If the size of the transmit data is greater than the maximum packet size for endpoint 0, only the maximum packet size is written to the FIFO. The USB_EP0_CSR[n]_P.TXPKTRDY bit is then set (indicating that there is a packet in the FIFO to be sent). When the packet has been sent to the host, software generates another endpoint 0 interrupt. The next data packet can be written to the FIFO.

When the last data packet has been written to the FIFO, software writes to the USB_EP0_CSR[n]_P register to set the USB_EP0_CSR[n]_P.TXPKTRDY bit and to set the USB_EP0_CSR[n]_P.DATAEND bit. (This activity indicates that there is no more data after this packet).

When the host moves to the status stage of the request, software generates another endpoint 0 interrupt to indicate that the request has completed. No further action is required from the software; the interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when the host decodes it, software must write to the USB_EP0_CSR[n]_P register. This operation sets the USB_EP0_CSR[n]_P.SPKTRDY bit and the USB_EP0_CSR[n]_P.SENDSTALL bit. When the host requests data, the USB controller sends a stall to tell the host that the request was not executed. Software generates an endpoint 0 interrupt and the USB_EP0_CSR[n]_P.SENTSTALL bit is set.

If the host requests more data after USB_EP0_CSR[n]_P.DATAEND is set, then the USB controller sends a stall. Software generates an endpoint 0 interrupt and the USB_EP0_CSR[n]_P.SENTSTALL bit is set.

Zero Data Requests

Zero data requests have all their information included in the 8-byte command and do not require transfer of extra data.

Examples of standard device requests for zero data are:

- SET_FEATURE
- CLEAR_FEATURE
- SET_ADDRESS
- SET_CONFIGURATION
- SET_INTERFACE

As with all requests, the sequence of events begins when the software receives an endpoint 0 interrupt. The USB_EP0_CSR[n]_P.RXPKTRDY bit is also set. The host must then read and decode the 8-byte command from the endpoint 0 FIFO, and take appropriate action. For example, if the command is SET_FEATURE, the host writes 7-bit address value contained in the command to the USB_FADDR register.

Software must set the USB_EP0_CSR[n]_P.SPKTRDY bit (indicating that the command has been read from the FIFO) and the USB_EP0_CSR[n]_P.DATAEND bit (indicating that no further data is expected for this request).

When the host moves to the status stage of the request, the USB controller generates a second endpoint 0 interrupt, indicating that the request has completed. No further action is required from the software; the second interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when the host decodes it, the USB_EP0_CSR[n]_P.SPKTRDY bit is set which sets the USB_EP0_CSR[n]_P.SENDSTALL bit. When the host moves to the status stage of the request, the USB controller sends a stall to tell the host that the request was not executed. The USB controller generates a second endpoint 0 interrupt and sets the USB_EP0_CSR[n]_P.SENTSTALL bit.

If the host sends more data after the USB_EP0_CSR[n]_P.DATAEND bit is set, then the USB controller sends a stall. It generates an endpoint 0 interrupt and sets the USB_EP0_CSR[n]_P.SENTSTALL bit.

Endpoint 0 States

When the USB operates as a peripheral, the endpoint 0 control has three modes (IDLE, Tx, and Rx). The modes correspond to the phases of the control transfer and the state that endpoint 0 enters during phases of the transfer. (See Endpoint 0 Service Routine as Peripheral.)

IDLE is the default mode on power-up or reset. The processor sets `RxPktRdy` bit when endpoint 0 is in IDLE state, indicating a new device request. Once the processor unloads the device request from the FIFO, the USB decodes the descriptor. It determines whether there is a data phase and, if so, the direction of the data phase of the control transfer (to set the FIFO direction).

Depending on the direction of the data phase, endpoint 0 goes into either Tx state or Rx state. If there is no data phase, endpoint 0 remains in IDLE state to accept the next device request.

The processor must take different actions at the different phases of the possible transfers (for example, loading the FIFO, setting `TxPktRdy`). The ***Endpoint 0 Control States*** figure shows the actions for the phase. The USB changes the FIFO direction depending on the direction of the data phase, independently of the processor.

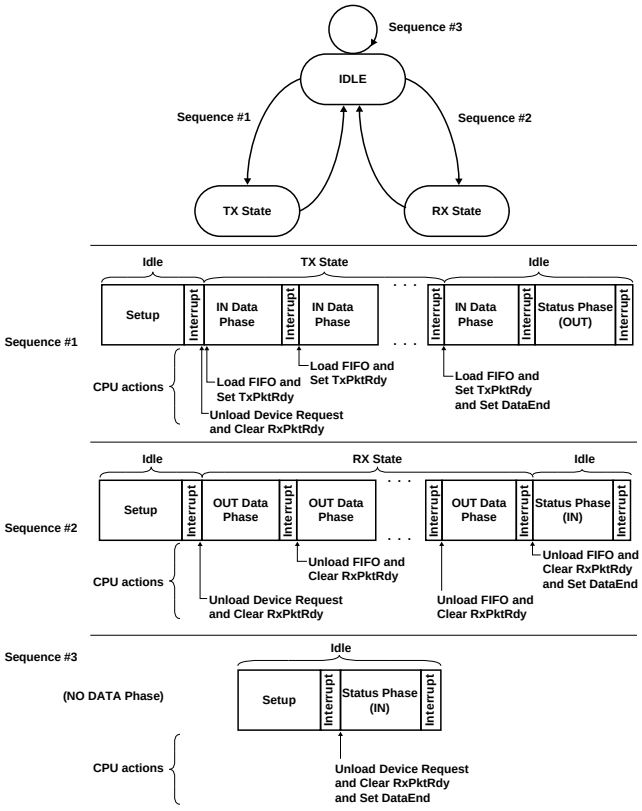

Figure 22-2: Endpoint 0 Control States

Endpoint 0 Service Routine as Peripheral

The USB controller generates an endpoint 0 interrupt when:

- The USB controller sets the USB_EP0_CSR[n]_P.RXPKTRDY bit after a valid token has been received and data has been written to the FIFO.
- The USB controller clears the USB_EP0_CSR[n]_P.TXPKTRDY bit after the data packet in the FIFO has been successfully transmitted to the host.
- The USB controller sets the USB_EP0_CSR[n]_P.SENTSTALL bit after a control transaction is ended due to a protocol violation.
- The USB controller sets the USB_EP0_CSR[n]_P.SETUPEND bit because a control transfer has ended before USB_EP0_CSR[n]_P.DATAEND is set.

Whenever the endpoint 0 service routine is entered, the firmware must first check whether the current control transfer has been ended. The transfer can end due to either a stall condition or a premature end-of-control transfer. If the control transfer ends due to a stall condition, the USB controller sets the USB_EP0_CSR[n]_P.SENTSTALL bit. If the control transfer ends due to a premature end-of-control transfer, the USB controller sets USB_EP0_CSR[n]_P.SETUPEND. In either case, the firmware must abort processing the current control transfer and set the state to IDLE.

Once the firmware has determined that an illegal bus state did not generate the interrupt, the next action depends on the endpoint state.

If endpoint 0 is in IDLE state, the only valid reason the USB controller can generate an interrupt is due to the core receiving data from the USB bus. The service routine must check for this state by testing the USB_EP0_CSR[n]_P.RXPKTRDY bit. If the USB controller sets the bit, then the core has received a SETUP packet. The processor unloads this packet from the FIFO and decodes it to determine the next action. Depending on the command contained within the SETUP packet, endpoint 0 enters one of the following three states.

- If the command is a single packet transaction (SET_ADDRESS, SET_INTERFACE and the others) without a data phase, the endpoint remains in the IDLE state.
- If the command has an OUT data phase (SET_DESCRIPTOR and others), the endpoint enters the Rx state.
- If the command has an IN data phase (GET_DESCRIPTOR and others), the endpoint enters the Tx state.

If the endpoint is in Tx state, the interrupt indicates that the core has received an IN token and data from the FIFO has been sent. The firmware must respond to this event by:

- Placing more data in the FIFO when the host still expects more data, or
- Setting the USB_EP0_CSR[n]_P.DATAEND bit to indicate that the data phase is complete

Once the data phase of the transaction completes, endpoint 0 returns to the IDLE state to await the next control transaction.

If the endpoint is in the Rx state, the interrupt indicates that a data packet has been received. The firmware must respond by unloading the received data from the FIFO. The firmware must then determine whether it has received all of the expected data. If it has, the firmware must set the USB_EP0_CSR[n]_P.DATAEND bit and return endpoint 0 to IDLE state. If more data is expected, the firmware must set the USB_EP0_CSR[n]_P.SPKTRDY bit to indicate that it has read the data in the FIFO and leave the endpoint in the Rx state.

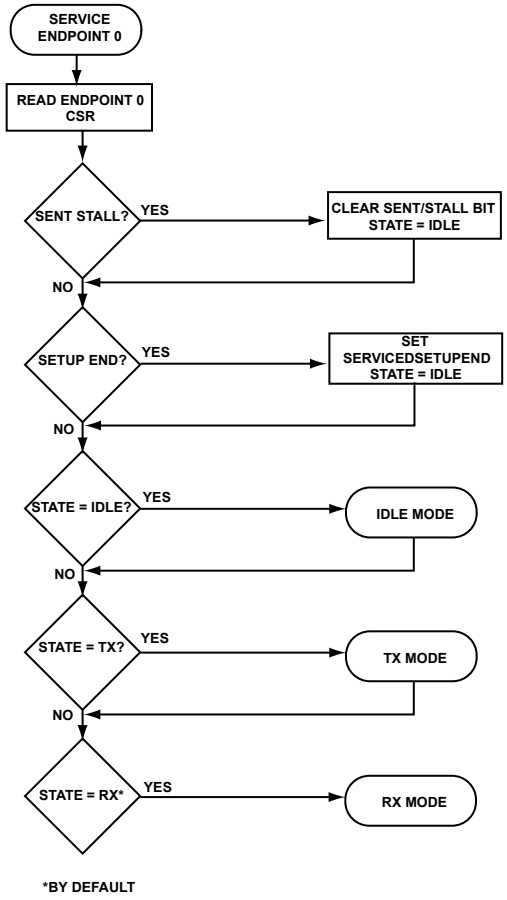

Figure 22-3: Endpoint 0 Service Routine

Idle Mode

The endpoint 0 control must select IDLE mode at power-on or reset. The endpoint 0 control returns to this mode when the Rx and Tx modes terminate.

As shown in the ***Endpoint 0 Idle Mode (Setup Phase)*** figure, the SETUP phase of control transfer is handled in IDLE mode.

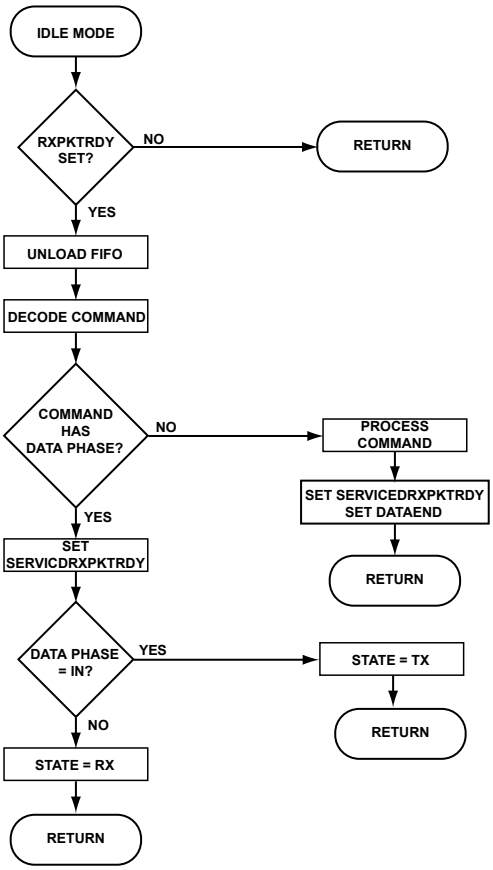

Figure 22-4: Endpoint 0 Idle Mode (Setup Phase)

Tx Mode

Refer to the ***Endpoint 0 Tx Mode*** figure. When the endpoint is in Tx state, all arriving IN tokens must be treated as part of a data phase until the required amount of data has been sent to the host. If either a SETUP or an OUT token is received while the endpoint is in the Tx state, a `USB_EP0_CSR[n]_P.SETUPEND` condition occurs. The core expects only IN tokens.

Three events can cause the Tx mode to terminate before the expected amount of data has been sent:

- The host sends an invalid token which sets the `USB_EP0_CSR[n]_P.SETUPEND` bit.
- The firmware sends a packet containing less than the maximum packet size for endpoint 0.
- The firmware sends an empty data packet.

Until the transaction is terminated, when the firmware receives an interrupt which indicates that a packet has been sent from the FIFO, it simply loads the FIFO. An interrupt is generated when the USB controller clears `USB_EP0_CSR[n]_P.TXPKTRDY`.

When the firmware forces the termination of a transfer (by sending a short or empty data packet), it must set the `USB_EP0_CSR[n]_P.DATAEND` bit. This event indicates to the core that the data phase is complete and that the core will receive an acknowledge packet next.

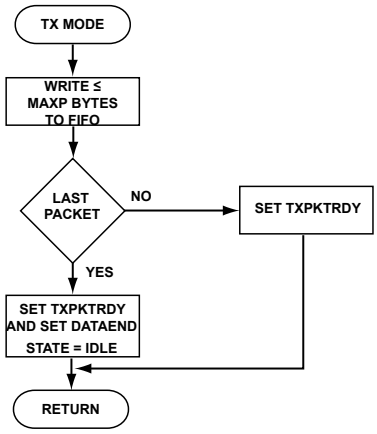

Figure 22-5: Endpoint 0 Tx Mode

Rx Mode

Refer to the ***Endpoint 0 Rx Mode*** figure. In Rx mode, all arriving data must be treated as part of a data phase until the expected amount of data is received. If either a SETUP or an IN token is received while the endpoint is in Rx state, a `USB_EP0_CSR[n]_P.SETUPEND` condition occurs since the core expects only OUT tokens.

Three events can cause the Rx mode to terminate before the expected amount of data is received:

- The host sends an invalid token which sets the `USB_EP0_CSR[n]_P.SETUPEND` bit.
- The host sends a packet which contains less than the maximum packet size for endpoint 0.
- The host sends an empty data packet.

The transaction terminates when the firmware receives an interrupt which indicates that new data has arrived (`USB_EP0_CSR[n]_P.RXPKTRDY` bit is set). Until the transaction terminates, firmware must unload the FIFO and clear `USB_EP0_CSR[n]_P.RXPKTRDY` by setting the `USB_EP0_CSR[n]_P.SPKTRDY` bit.

When the firmware detects the termination of a transfer (by receiving either the expected amount of data or an empty data packet), it must set the `USB_EP0_CSR[n]_P.DATAEND` bit. This event indicates to the core that the data phase is complete and that the core will receive an acknowledge packet next.

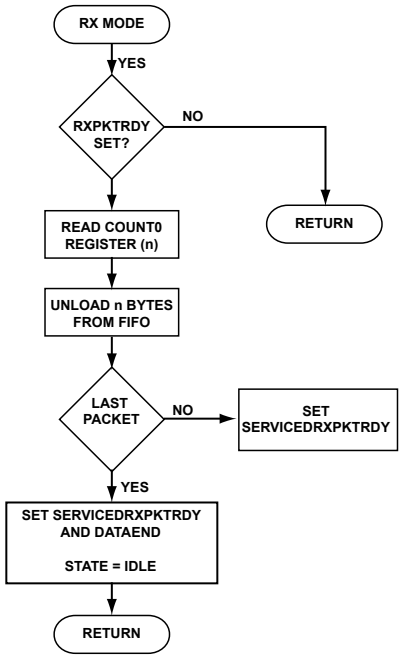

Figure 22-6: Endpoint 0 Rx Mode

Peripheral Mode, Bulk IN, Transfer Size Known

For this process, the maximum size of an individual packet (*MaxPktSize*) in bytes and the complete transfer size (*TxferSize*) in bytes, must be known.

1. Load *MaxPktSize* into the `USB_EP[n]_TXMAXP` register.
2. Set the following bits: `USB_EP[n]_TXCSR_P.DMAREQEN` = 1, `USB_EP[n]_TXCSR_P.AUTOSET` = 1, `USB_EP[n]_TXCSR_P.ISO` = 0, `USB_EP[n]_TXCSR_P.FRCDATATGL`= 0.
3. Load the *TxferSize* value into the `USB_DMA[n]_CNT` register.
4. Configure the DMA controller to write the data into the corresponding Tx FIFO address.
5. On each `USB_DMA[n]_CNT` transition, the DMA controller writes a new packet into the FIFO. The `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is automatically set when each new packet is written.

   ADDITIONAL INFORMATION: Repeat Step 5 for each full packet of the transfer. Even if the final packet is a short packet, the USB controller automatically detects the packet because the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is set.

Peripheral Mode, Bulk IN, Transfer Size Unknown

For this process, assume the maximum individual packet size (*MaxPktSize*) in bytes is an even number of bytes.

1. Load *MaxPktSize* into the `USB_EP[n]_TXMAXP` register.

2. Set the following bits: USB_EP[n]_TXCSR_P.DMAREQEN = 1, USB_EP[n]_TXCSR_P.AUTOSET = 1, USB_EP[n]_TXCSR_P.ISO = 0, USB_EP[n]_TXCSR_P.FRCDATATGL= 0.
3. Configure the DMA controller to write *MaxPktSize*/2 half words into the corresponding Tx FIFO address on each USB_DMA[n]_CNT.
4. Set up an ISR, sensitive to the DMA work-block-complete interrupt, that writes a remaining short packet into the Tx FIFO using processor core DMA. Then, set the USB_EP[n]_TXCSR_P.TXPKTRDY bit or toggle the USB_EP[n]_TXCSR_P.TXPKTRDY bit to send a zero-length packet.
5. On each USB_DMA[n]_CNT transition, the DMA controller writes a new packet into the FIFO. The USB_EP[n]_TXCSR_P.TXPKTRDY bit is set automatically when each new packet is written.

   ADDITIONAL INFORMATION: Repeat step 5 for each full packet of the transfer. The ISR from step 4 manages the final short or zero-length packet.

Peripheral Mode, ISO IN, Small MaxPktSize

For this process, the maximum individual packet size (*MaxPktSize*) in bytes is less than 128 bytes and is an even number of bytes. Assume that double buffering is enabled, and the auto-set feature is not used (because packets are often less than *MaxPktSize*).

1. Load *MaxPktSize* into the USB_EP[n]_TXMAXP register.
2. Set the following bits: USB_EP[n]_TXCSR_P.ISO = 1.
3. Preload the first two packets into the endpoint Tx FIFO and set the USB_EP[n]_TXCSR_P.TXPKTRDY bit.
4. Set up an ISR, sensitive to the USB_IRQ.SOF interrupt, which writes a new packet into the Tx FIFO and sets the USB_EP[n]_TXCSR_P.TXPKTRDY bit.
5. Set the USB_IEN.SOF bit = 1 to generate an interrupt on each start-of-frame.

   ADDITIONAL INFORMATION: Repeat step 5 for each ISO packet.

Peripheral Mode, ISO IN, Large MaxPktSize

For this process, the maximum individual packet size (*MaxPktSize*) in bytes is greater than 128 bytes and is an even number of bytes. Assume that double buffering is enabled, and the auto-set feature is not used (because packets are often less than *MaxPktSize*).

1. Load *MaxPktSize* into the USB_EP[n]_TXMAXP register.
2. Set the USB_EP[n]_TXCSR_P.ISO bit = 1.
3. Set the USB_POWER.ISOUPDT bit = 1 to prevent the initial packet loaded into the FIFO from transmitting on the USB until the next 1ms frame.
4. Load the total number of bytes for the first two packets into the USB_DMA[n]_CNT register.

5. Configure the DMA controller to pre-load the two packets into the corresponding Tx FIFO address and set the`USB_EP[n]_TXCSR_P.TXPKTRDY` bit.
6. Set up an ISR, sensitive to the `USB_IRQ.SOF` interrupt, which writes a new packet into the Tx FIFO by configuring the DMA controller to load the packet.
7. Set the `USB_IEN.SOF` bit = 1 to generate an interrupt on each start-of-frame.

   ADDITIONAL INFORMATION: Repeat step 7 for each ISO packet.

Peripheral Mode, Bulk OUT, Transfer Size Known

For this process, the maximum individual packet size (*MaxPktSize*) in bytes and the complete transfer size (*TxferSize*) in bytes must be known.

1. Load *MaxPktSize* into `USB_EP[n]_RXMAXP`.
2. Set the following bits: `USB_EP[n]_RXCSR_P.DMAREQEN` = 1, `USB_EP[n]_RXCSR_P.AUTOCLR`= 1, `USB_EP[n]_RXCSR_P.ISO` = 0, `USB_EP[n]_RXCSR_P.CLRDATATGL` = 0, `USB_EP[n]_RXCSR_P.DMAREQMODE` = 0.
3. Configure the DMA controller to read the full *TxferSize*/2 half words from the corresponding Rx FIFO address.
4. On each `USB_DMA[n]_CNT` transition, the DMA controller reads another packet from the FIFO. The USB controller automatically clears the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit when each new packet is read.

   ADDITIONAL INFORMATION: Repeat step 5 for each full packet of the transfer. If *TxferSize* is not an exact multiple of *MaxPktSize*, the final `USB_DMA[n]_CNT` transition causes the DMA controller to read out only the short packet that remains.

Peripheral Mode, Bulk OUT, Transfer Size Unknown

For this process, the maximum individual packet size (*MaxPktSize*) in bytes must be known.

1. Load *MaxPktSize* into `USB_EP[n]_RXMAXP`.
2. Set the following bits: `USB_EP[n]_RXCSR_P.DMAREQEN`= 1, `USB_EP[n]_RXCSR_P.AUTOCLR` = 1, `USB_EP[n]_RXCSR_P.ISO` = 0, `USB_EP[n]_RXCSR_P.CLRDATATGL`= 0, `USB_EP[n]_RXCSR_P.DMAREQMODE` = 1.
3. Set the appropriate bit in the `USB_INTRRXE` register.
4. Configure the DMA controller to read *MaxPktSize*/2 half words from the corresponding Rx FIFO address on each `USB_DMA[n]_CNT` transition.
5. Set up an ISR, sensitive to the Rx interrupt, which reads the `USB_EP[n]_RXCNT` register and then transfers `USB_EP[n]_RXCNT` bytes (in half words) from the Rx FIFO to the processor core.

ADDITIONAL INFORMATION: Depending on the number of bytes in the FIFO, configure the DMA to read the data, or read it with the processor core.

ADDITIONAL INFORMATION: On each USB_DMA[n]_CNT transition, the DMA controller reads a packet from the FIFO. The USB controller automatically clears the USB_EP[n]_RXCSR_P.RXPKTRDY bit when each new packet is read.

ADDITIONAL INFORMATION: Repeat step 5 for each full packet of the transfer.

6. If a packet is received that is less than *MaxPktSize*, the Rx interrupt goes high, and the ISR from step 5 reads out the remaining short packet.

Peripheral Mode, ISO OUT, Small MaxPktSize

For this process, the maximum individual packet size (*MaxPktSize*) in bytes is less than 128 bytes. Assume that double buffering is enabled.

1. Load the *MaxPktSize* value into the USB_EP[n]_RXMAXP register.
2. Set the USB_EP[n]_RXCSR_P.ISO bit = 1.
3. Set up an ISR, sensitive to the USB_IRQ.SOF interrupt, that reads the USB_EP[n]_RXCSR_P.FIFOFULL bit and then reads the USB_EP0_CNT[n].RXCNT status register. The ISR removes one or two packets (equal to the USB_EP0_CNT[n].RXCNT number of bytes) from the FIFO. It then clears the USB_EP[n]_RXCSR_P.RXPKTRDY bit.
4. Set the USB_IEN.SOF bit = 1 to generate an interrupt on each start-of-frame.

   ADDITIONAL INFORMATION: Repeat step 4 for each ISO packet.

Peripheral Mode, ISO OUT, Large MaxPktSize

For this process, the maximum individual packet size (*MaxPktSize*) in bytes is greater than 128 bytes. Assume that double buffering is enabled.

1. Load *MaxPktSize* into the USB_EP[n]_RXMAXP register.
2. Set the USB_EP[n]_RXCSR_P.ISO bit = 1.
3. Set up an ISR (sensitive to the USB_IRQ.SOF interrupt), that reads the USB_EP[n]_RXCSR_P.FIFOFULL bit, and then reads the USB_EP[n]_RXCNT status register. The ISR configures the DMA controller to remove one or two packets (equal to the USB_EP[n]_RXCNT number of bytes) from the FIFO.
4. Set up an ISR, sensitive to the DMA work-block-complete interrupt to clear the USB_EP[n]_RXCSR_P.RXPKTRDY.
5. Set the USB_IEN.SOF bit = 1 to generate an interrupt on each start-of-frame.

   ADDITIONAL INFORMATION: Repeat step 5 for each ISO packet.

Peripheral Mode Suspend

When no activity has occurred on the USB for 3 ms, the USB controller enters suspend mode. If the suspend interrupt (USB_IRQ.SUSPEND) is enabled, an interrupt is generated now.

When resume signaling is detected, the USB controller exits suspend mode. If the USB_IRQ.RESUME interrupt is enabled, an interrupt is generated. The processor core can also force the USB controller to exit suspend mode by setting the USB_POWER.RESUME bit. This event initiates a remote wake-up. When this bit is set, the USB controller exits suspend mode and drives resume signaling onto the bus. The processor core must clear this bit after 10 ms (a maximum of 15 ms) to end resume-signaling.

NOTE: The USB_IRQ.RESUME interrupt is not generated when the processor core exits suspend mode. This interrupt is not generated when the software initiates remote wake-up.

Start-of-frame (SOF) Packets

When the USB controller operates in peripheral mode, it receives a start-of-frame packet from the host every millisecond when in full-speed mode.

When the USB controller receives a SOF packet, it writes the 11-bit frame number contained in the packet into the USB_FRAME register. An output pulse, lasting one USB clock bit period, is generated. A start-of-frame interrupt is also generated (if enabled by the USB_IRQ.SOF bit).

After the USB controller has started to receive SOF packets, the controller expects one every millisecond . If the USB controller does not receive an SOF packet after 1.00358 ms , it is assumed that the packet is lost. A start-of-frame pulse (together with a USB_IRQ.SOF interrupt) is still generated even though the USB_FRAME register is not updated. The USB controller continues to generate an SOF pulse every millisecond It resynchronizes these pulses to the received SOF packets when it successfully receives these packets again.

Soft Connect/Soft Disconnect

In peripheral mode, the USB controller sets or clears the USB_POWER.SOFTCONN bit to switch between normal mode and non-driving mode. When USB_POWER.SOFTCONN=1, the USB controller is in normal mode and the D+/D– lines of the USB bus are enabled. When the USB_POWER.SOFTCONN=0, the PHY is put into non-driving mode and D+ and D– are three-stated. The USB controller appears to have been disconnected from the USB bus.

After system reset, USB_POWER.SOFTCONN=0. From that point, the USB controller appears disconnected until the software has set USB_POWER.SOFTCONN =1. The application software can then choose when to set the PHY to its normal mode. Systems with a lengthy initialization procedure can use this functionality to ensure that initialization is complete and the system is ready to perform enumeration before connecting to the USB. Once the USB_POWER.SOFTCONN bit is set to 1, the software can also clear this bit to 0 to simulate a disconnect.

Error Handling As a Peripheral

The host can abort a control transfer due to a protocol error on the USB. The function controller software can also abort the transfer (for example, because it cannot process the command).

The USB controller automatically detects protocol errors and sends a stall packet to the host under the following conditions.

1. The host sends more data during the OUT data phase of a write request than specified in the command. This condition is detected when the host sends an OUT token after the USB_EP0_CSR[n]_P.DATAEND bit is set.
2. The host requests more data during the IN data phase of a read request than specified in the command. This condition is detected when the host sends an IN token after the USB_EP0_CSR[n]_P.DATAEND bit is set.
3. The host sends more than *MaxPktSize* data bytes in an OUT data packet.
4. The host sends a non-zero length DATA1 packet during the status phase of a read request.

When the USB controller has sent the stall packet, it sets the USB_EP0_CSR[n]_P.SENTSTALL bit and generates an interrupt. When the software receives an endpoint 0 interrupt with the USB_EP0_CSR[n]_P.SENTSTALL bit set, it aborts the current transfer, clears the USB_EP0_CSR[n]_P.SENTSTALL bit, and returns to the IDLE state.

If the host enters the status phase before all the data for the request transfers, or sends a new SETUP packet before completing the current transfer, then it prematurely ends the transfer. The USB_EP0_CSR[n]_P.SETUPEND bit is set and an endpoint 0 interrupt is generated. When the software receives an endpoint 0 interrupt with the USB_EP0_CSR[n]_P.SETUPEND bit set, it aborts the current transfer, sets the USB_EP0_CSR[n]_P.SSETUPEND bit, and returns to the IDLE state. If the USB_EP0_CSR[n]_P.RXPKTRDY bit is set, it indicates that the host has sent another SETUP packet and the software must then process this command.

If the software wants to abort the current transfer, because it cannot process the command or has some other internal error, then it must set the USB_EP0_CSR[n]_P.SENTSTALL bit. The USB controller then sends a stall packet to the host, sets the USB_EP0_CSR[n]_P.SENTSTALL bit, and generates an endpoint 0 interrupt.

Stalls Issued to Control Transfers

In peripheral mode, the USB controller automatically issues a stall handshake to a control transfer under the following conditions:

1. The host sends more data during an OUT data phase of a control transfer than specified in the device request during the SETUP phase. The USB controller detects this condition when the host sends an OUT token (instead of an IN token) after the processor core unloads the last OUT packet and sets the USB_EP0_CSR[n]_P.DATAEND bit.
2. The host requests more data during an IN data phase of a control transfer than specified in the device request during the SETUP phase. The USB controller detects this condition when the host sends an IN token (instead of an OUT token) after the processor core clears USB_EP[n]_TXCSR_P.TXPKTRDY and sets USB_EP0_CSR[n]_P.DATAEND. The processor sets USB_EP0_CSR[n]_P.DATAEND in response to the host-issued ACK for the last packet.
3. The host sends more than *MaxPktSize* data with an OUT data token.
4. The host sends the wrong PID (packet identifier) for the OUT status phase of a control transfer.
5. The host sends more than a zero length data packet for the OUT status phase.

Zero Length OUT Data Packets in Control Transfers

The USB controller uses a zero-length OUT data packet to indicate the end of a control transfer. In normal operation, such packets must only be received after the entire length of the device request transfers (for example, after the processor core has set the USB_EP0_CSR[n]_P.DATAEND bit). If the host sends a zero-length OUT data packet before the entire length of device request transfers, this packet signals the premature end of the transfer. In this case, the USB controller automatically flushes any IN token the processor core has loaded for the data phase from the FIFO and sets the USB_EP0_CSR[n]_P.SETUPEND bit.

Host Mode

USB OTG interface operations in host mode differ from peripheral mode in a number of ways. The following sections describe host mode operations.

Transaction Scheduling

When operating as a host, the USB controller maintains a frame counter.

If the target function is a full-speed device, the USB controller automatically sends an SOF packet at the start of each frame.

If the target function is a low-speed device, a K state is transmitted on the bus to act as a *keep-alive*. It stops the low-speed device from going into suspend mode.

After the SOF packet is transmitted, the USB controller cycles through all the endpoints looking for active transactions. An active transaction is defined as an Rx endpoint for which the USB_EP[n]_RXCSR_H.REQPKT bit is set or a Tx endpoint for which the USB_EP[n]_TXCSR_H.TXPKTRDY bit is set.

An active isochronous or interrupt transaction only starts if:

- It is found on the first transaction scheduler cycle of a frame, and
- The interval counter for that endpoint has counted down to zero

This functionality ensures that only one interrupt or isochronous transaction occurs per endpoint per *n* frames where *n* is the interval set in the USB_EP[n]_TXINTERVAL or USB_EP[n]_RXINTERVAL register for that endpoint.

An active bulk transaction starts immediately, provided there is sufficient time left in the frame to complete the transaction before the next SOF packet is due. If the transaction must be retried, then it is not retried until the transaction scheduler has checked all the other endpoints for active transactions first. (For example, the transaction is retried because a NAK was received or the target function did not respond). This check ensures that an endpoint that is sending many NAKs does not block other transactions on the bus. The USB controller permits specifying a limit (USB_EP[n]_TXINTERVAL or USB_EP[n]_RXINTERVAL registers) to the length of time in which NAKs can be received from a particular target before the endpoint is timed out.

Endpoint Setup and Data Transfer

When the HOST_MODE bit is set to 1, the USB controller operates as a host for point-to-point communications with another USB device. Or, when attached to a hub, the USB controller operates as a host for communication with a range of devices in a multi-point set-up.

The USB controller supports full-speed and low-speed USB functions, both for point-to-point communication and for operation through a hub.

Where necessary, the core automatically carries out the necessary transaction translation to allow usage of a low-speed or full-speed device with a USB 2.0 hub.

The USB controller supports control, bulk, isochronous, or interrupt transactions.

Transfers between the subsystem and endpoint FIFOs in host mode are similar to peripheral mode. See the descriptions of processor core-to-FIFO data transfer in Peripheral Mode.

Control Transaction as a Host

Host control transactions are conducted through endpoint 0. The software handles all the standard device requests that are sent or received through endpoint 0 (as described in *Universal Serial Bus Specification*, Revision 2.0, Chapter 9).

For a USB peripheral, there are three categories of standard device requests:

- ***Zero data requests.*** Comprised of a SETUP command followed by an IN status phase. The command includes all the information.
- ***Write requests.*** Comprised of a SETUP command, followed by an OUT data phase followed by an IN status phase. Extra data follows the command.
- ***Read requests*** Comprised of a SETUP command, followed by an IN data phase followed by an OUT status phase. The device is required to send data back to the host.

A timeout can be set to limit the length of time during which the USB controller retries a transaction that the target continually NAKs. This limit can be between 2 and 2^{15} frames or micro frames and is set through the USB_EP0_NAKLIMIT[n] register.

The following sections describe the steps taken in different phases of a control transaction and the actions of the core when issuing standard device requests.

Set up Phase as a Host

The processor core driving the host device performs the following actions for the SETUP phase of a control transaction.

1. Load the 8 bytes of the required device request command into the endpoint 0 FIFO.
2. Set the USB_EP0_CSR[n]_H.SETUPPKT bit and USB_EP0_CSR[n]_H.TXPKTRDY bit. These bits must be set together.

The USB controller then sends a SETUP token followed by the 8-byte command to endpoint 0 of the addressed device, retrying as necessary.

3. At the end of the attempt to send the data, the USB controller generates an endpoint 0 interrupt (for example, set USB_INTRTXE.EP0). The processor core then reads the USB_EP0_CSR[n]_H register to establish whether the USB_EP0_CSR[n]_H.RXSTALL, USB_EP0_CSR[n]_H.TOERR, or the USB_EP0_CSR[n]_H.NAKTO bits are set.

   If USB_EP0_CSR[n]_H.RXSTALL=1, the target did not accept the command (for example, because the target device does not support it) and issues a stall response.

   If USB_EP0_CSR[n]_H.TOERR=1, the USB controller tried to send the SETUP packet and the following data packet three times without getting a response.

   If USB_EP0_CSR[n]_H.NAKTO=1, the USB controller received a NAK response to each attempt to send the SETUP packet, for longer than the time set in the USB_EP0_NAKLIMIT[n] register. Direct the USB controller to either clear the USB_EP0_CSR[n]_H.NAKTO bit to continue trying this transaction (until it times out again) or to flush the FIFO to abort the transaction before clearing the USB_EP0_CSR[n]_H.NAKTO bit.

4. If none of USB_EP0_CSR[n]_H.RXSTALL, USB_EP0_CSR[n]_H.TOERR or USB_EP0_CSR[n]_H.NAKTO bits are set, the SETUP phase is correctly acknowledged. The processor core can proceed to the following IN data phase, OUT data phase or IN status phase specified for the particular standard device request.

IN Data Phase as a Host

The processor core driving the host device performs the following actions for the IN data phase of a control transaction.

1. Set the USB_EP0_CSR[n]_H.REQPKT bit.
2. Wait while the USB controller sends the IN token and then receives the required data back.
3. When the USB controller generates the endpoint 0 interrupt (for example, by setting the USB_INTRTXE.EP0 bit), read the USB_EP0_CSR[n]_H register. Determine whether the USB_EP0_CSR[n]_H.RXSTALL bit, the USB_EP0_CSR[n]_H.TOERR bit, the USB_EP0_CSR[n]_H.NAKTO bit, or the USB_EP0_CSR[n]_H.RXPKTRDY bit is set.

   If USB_EP0_CSR[n]_H.RXSTALL=1, the target has issued a stall response.

   If USB_EP0_CSR[n]_H.TOERR=1, the USB controller has tried to send the required IN token three times without getting a response.

   If USB_EP0_CSR[n]_H.NAKTO=1, the USB controller has received a NAK response to each attempt to send the IN token, for longer than the time set in the USB_EP0_NAKLIMIT[n] register. Direct the USB controller to either clear the USB_EP0_CSR[n]_H.NAKTO bit to continue trying this transaction (until it times out again) or clear USB_EP0_CSR[n]_H.REQPKT before clearing the USB_EP0_CSR[n]_H.NAKTO bit to abort the transaction.

4. If the USB_EP0_CSR[n]_H.RXPKTRDY bit is set, the processor core reads the data from the endpoint 0 FIFO, then clears USB_EP0_CSR[n]_H.RXPKTRDY.
5. If further data is expected, the processor core must repeat the previous steps.

   When all the data is successfully received, the processor core can proceed to the OUT status phase of the control transaction.

OUT Data as a Host (Control)

The processor core driving the host device performs the following actions for the OUT data phase of a control transaction.

1. Load the data to be sent into the endpoint 0 FIFO.
2. Set the USB_EP0_CSR[n]_H.TXPKTRDY bit.

   The USB controller sends an OUT token followed by the data from the FIFO to endpoint 0 of the addressed device, retrying as necessary.
3. At the end of the attempt to send the data, the USB controller generates an endpoint 0 interrupt (for example by setting the USB_INTRTX.EP0 bit). The processor core can then read the USB_EP0_CSR[n]_H to establish whether the USB_EP0_CSR[n]_H.RXSTALL bit, the USB_EP0_CSR[n]_H.TOERR bit, or the USB_EP0_CSR[n]_H.NAKTO bit is set.

   If USB_EP0_CSR[n]_H.RXSTALL=1, the target has issued a stall response.

   If USB_EP0_CSR[n]_H.TOERR=1 the USB controller has tried to send the OUT token and the following data packet three times without getting a response.

   If USB_EP0_CSR[n]_H.NAKTO=1, the USB controller has received a NAK response to each attempt to send the OUT token, for longer than the time set in the USB_EP0_NAKLIMIT[n] register. Direct the USB controller to either clear the USB_EP0_CSR[n]_H.NAKTO bit to continue trying this transaction (until it times out again) or to flush the FIFO to abort the transaction before clearing the USB_EP0_CSR[n]_H.NAKTO bit.

   If none of the USB_EP0_CSR[n]_H.RXSTALL, USB_EP0_CSR[n]_H.TOERR, or USB_EP0_CSR[n]_H.NAKTO bits are set, the OUT data is correctly acknowledged.
4. If further data must be sent, the processor core must repeat the previous steps.

   When all the data is successfully sent, the processor core proceeds to the IN status phase of the control transaction.

IN Status Phase as a Host (Following SETUP Phase or OUT Data Phase)

The processor core driving the host device performs the following actions for the IN status phase of a control transaction.

1. Set the USB_EP0_CSR[n]_H.STATUSPKT and USB_EP0_CSR[n]_H.REQPKT bits. These bits must be set together.

2. Wait while the USB controller both sends an IN token and receives a response from the USB peripheral.
3. When the USB controller generates the endpoint 0 interrupt (for example, it sets the USB_INTRTX.EP0 bit), read the USB_EP0_CSR[n]_H register to establish whether the USB_EP0_CSR[n]_H.RXSTALL, USB_EP0_CSR[n]_H.TOERR, USB_EP0_CSR[n]_H.NAKTO, or the USB_EP0_CSR[n]_H.RXPKTRDY bits are set.

   If USB_EP0_CSR[n]_H.RXSTALL=1, the target could not complete the command and so has issued a stall response.

   If USB_EP0_CSR[n]_H.TOERR=1, the USB controller has tried to send the required IN token three times without getting a response.

   If USB_EP0_CSR[n]_H.NAKTO=1, the USB controller has received a NAK response to each attempt to send the IN token, for longer than the time set in the USB_EP0_NAKLIMIT[n] register. Direct the USB controller to either clear the USB_EP0_CSR[n]_H.NAKTO bit to continue trying this transaction (until it times out again) or clear USB_EP0_CSR[n]_H.REQPKT before clearing the USB_EP0_CSR[n]_H.NAKTO bit to abort the transaction.
4. If the USB_EP0_CSR[n]_H.RXPKTRDY bit is set, the processor core typically clears it.

OUT Status Phase as a Host (Following IN Data Phase)

The processor core driving the host device performs the following actions for the OUT status phase of a control transaction.

1. Set USB_EP0_CSR[n]_H.STATUSPKT and USB_EP0_CSR[n]_H.TXPKTRDY bits. These bits must be set together.
2. Wait while the USB controller both sends the OUT token and a zero-length DATA1 packet.
3. At the end of the attempt to send the data, the USB controller generates an endpoint 0 interrupt. The processor core then reads the USB_EP0_CSR[n]_H register to discover when the USB_EP0_CSR[n]_H.RXSTALL, USB_EP0_CSR[n]_H.TOERR, or USB_EP0_CSR[n]_H.NAKTO bits are set.

   If USB_EP0_CSR[n]_H.RXSTALL=1, the target could not complete the command and so has issued a stall response.

   If USB_EP0_CSR[n]_H.TOERR=1, the USB controller has tried to send the STATUS packet and the following data packet three times without getting a response.

   If USB_EP0_CSR[n]_H.NAKTO=1, the USB controller has received a NAK response to each attempt to send the IN token, for longer than the time set in the USB_EP0_NAKLIMIT[n] register. Direct the USB controller to either clear the USB_EP0_CSR[n]_H.NAKTO bit to continue trying this transaction (until it times out again) or to flush the FIFO to abort the transaction before clearing the USB_EP0_CSR[n]_H.NAKTO bit.
4. If none of the USB_EP0_CSR[n]_H.RXSTALL, USB_EP0_CSR[n]_H.TOERR, or USB_EP0_CSR[n]_H.NAKTO bits are set, the status phase is correctly acknowledged.

Host IN Transactions

When the USB controller operates as a host, IN transactions are handled like OUT transactions are handled when the USB controller is operating as a peripheral. First, the USB controller sets USB_EP[n]_RXCSR_H.REQPKT bit to initiate the transaction. This bit indicates to the transaction scheduler that there is an active transaction on this endpoint. The transaction scheduler then sends an IN token to the target function.

When the packet is received and placed in the Rx FIFO, the USB_EP[n]_RXCSR_H.RXPKTRDY bit is set. The appropriate Rx endpoint interrupt is generated (if enabled) to signal that the processor can now unload a packet can now from the FIFO. When the processor unloads the packet, USB_EP[n]_RXCSR_H.RXPKTRDY is cleared. The USB controller uses the USB_EP[n]_RXCSR_H.AUTOCLR bit to clear the USB_EP[n]_RXCSR_H.RXPKTRDY bit automatically when the processor unloads a maximum sized packet from the FIFO. There is also an USB_EP[n]_RXCSR_H.AUTOREQ bit that automatically sets the USB_EP[n]_RXCSR_H.REQPKT bit when the USB_EP[n]_RXCSR_H.RXPKTRDY bit is cleared. The USB controller can use the USB_EP[n]_RXCSR_H.AUTOCLR and USB_EP[n]_RXCSR_H.AUTOREQ bits with an external DMA controller to perform complete bulk transfers without processor core intervention.

If the target function responds to a bulk or interrupt IN token with a NAK, the USB controller retries the transaction until the NAK limit set in the USB_EP0_NAKLIMIT[n] register is reached. If the target function responds with a stall, the USB controller does not retry the transaction, but sets the USB_EP[n]_RXCSR_H.RXSTALL bit to interrupt the processor core. If the target function does not respond to the IN token within the required time, the USB controller retries the transaction.(USB controller also retries the transaction if there was a CRC or bit-stuff error in the packet). If, after three attempts, the target function still has not responded, the USB controller clears the USB_EP[n]_RXCSR_H.REQPKT bit and interrupts the processor core with the DATAERROR_R bit in USB_RXCSR set.

Host OUT Transactions

When the USB controller operates as a host, OUT transactions are handled like IN transactions are handled when the USB controller operates as a peripheral.

The USB_EP[n]_TXCSR_H.TXPKTRDY bit must be set as the processor loads each packet into the Tx FIFO. The USB controller uses the USB_EP[n]_TXCSR_H.AUTOSET bit to cause the USB_EP[n]_TXCSR_H.TXPKTRDY bit to be automatically set when the processor loads a maximum sized packet into the FIFO. The USB controller can use the USB_EP[n]_TXCSR_H.AUTOSET bit with an external DMA controller to perform complete bulk transfers without processor core intervention.

If the target function responds to the OUT token with a NAK, the USB controller retries the transaction until the NAK limit set in the USB_EP0_NAKLIMIT[n] register is reached. If the target function responds with a stall, the USB controller does not retry the transaction, but sets the USB_EP[n]_TXCSR_H.RXSTALL bit to interrupt the processor core. If the target function does not respond to the OUT token within the required time, the USB controller retries the transaction.(USB controller also retries the transaction if there was a CRC or bit-stuff error in the packet). If, after three attempts, the target function still has not responded, the USB controller flushes the FIFO and sets the USB_EP[n]_TXCSR_H.TXTOERR bit to interrupt the processor core.

Multi-Point Support

The following sections describe the multi-point support of the USB controller.

- Allocating Devices to Endpoints
- Multi-Point Operation
- Multi-Point Bandwidth Considerations

Allocating Devices to Endpoints

The separate functions of the connected devices are allocated to the endpoints within the USB controller through a group of three registers. The registers are associated with each implemented Rx or Tx endpoint (including endpoint 0).

The registers are:

- USB_MP[n]_TXFUNCADDR/ USB_MP[n]_RXFUNCADDR
- USB_MP[n]_TXHUBADDR/ USB_MP[n]_RXHUBADDR
- USB_MP[n]_TXHUBPORT/ USB_MP[n]_RXHUBPORT

The location of these registers depends on which of the endpoints is being addressed.

Record the address of the target function that is accessed through the selected endpoint in the transmit and receive function address registers. Record this information separately for each Tx and Rx endpoint used. In particular, set both USB_MP[n]_TXFUNCADDR and USB_MP[n]_RXFUNCADDR for endpoint 0.

The USB controller uses the transmit and receive hub address and hub port registers when a full-speed or low-speed device is connected to it through a full-speed USB 2.0 hub. The hub carries out the required transaction translation between full-speed transmission and low-speed or full-speed transmission. In this situation, the USB_MP[n]_TXHUBADDR/ USB_MP[n]_RXHUBADDR and USB_MP[n]_TXHUBPORT/ USB_MP[n]_RXHUBPORT registers must record the address of the hub that carries out the transaction translation. It must also record the address of the port of that hub through which the associated Tx or Rx endpoint must access the device.

If endpoint 0 is connected to a hub, then set both the Tx and the Rx versions of these registers for this endpoint. The USB controller also uses hub address registers to record whether the hub offers multiple transaction translators or just a single transaction translator. This configuration has a significant effect on the overall bandwidth that can be achieved.

In addition to recording the address of the target function, record the endpoint number and operating speed of the target device and the type of transaction that is executed. For a Tx endpoint, set this information in the USB_EP[n]_TXTYPE register when the index register is set to select the required endpoint. For an Rx endpoint, set this information in the USB_EP[n]_RXTYPE register when the index register is set to select the required endpoint. In both cases, record the endpoint number in bits 3–0, select the transaction type through bits 5–4, and select the operating speed through bits 7–6.

Set only the speed for endpoint 0 because endpoint 0 only has the facilities to handle control transactions and therefore is always associated with a device endpoint 0. Use bits 7–6 of the Type 0 register to set the speed. The register is located at address 0x1A when the index register is set to 0.

Multi-Point Operation

After allocating functions to endpoints and recording the operating speed of the target device, multi-point operations can be configured. Most operations in a multi-point set-up are the same as for the equivalent actions where the core is attached to a single other device.

However, more steps are required when:

- The option of dynamically switching the allocation of functions to endpoints is taken (for example, to allow the support of a wider range of devices)
- The control packets normally associated with endpoint 0 are handled through a different endpoint.

If dynamic allocation is used, the program must monitor the current data toggle state associated with the endpoint and with each of the devices that are allocated to that endpoint. This knowledge allows the program to select the correct data toggle state when switching occurs between one device and the other. (This action is the programs responsibility. The core cannot determine what data toggle state is expected when a function switches in and out of use.)

The data toggle state can be switched from its current state by writing to the appropriate `USB_EP[n]_TXCSR_H` or `USB_EP[n]_RXCSR_H` register. This activity sets the data toggle write enable and data toggle bits that are included in the registers when the core is in host mode.

Data toggle write enable and data toggle bits are also included in the `USB_EP0_CSR[n]_H` register. However, control operations carried out through endpoint 0 of the core normally leave the data toggle in the expected state.

Where control packets are handled through an endpoint other than endpoint 0, programs must prompt for each setup token to be sent. Programs must set the `USB_EP[n]_TXCSR_H.SETUPPKT` bit when the core operates in host mode, along with the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit. If the `USB_EP[n]_TXCSR_H.SETUPPKT` bit is not set, an OUT token is sent.

Use endpoint 0 of the USB controller to handle control packets for all of the devices attached to the controller, and to switch the allocation of this endpoint, as appropriate. Sending the correct token is ensured, as is ensuring that the data toggle is correctly set for this endpoint.

Using a different endpoint for this function is possible, as described, but note the following:

- The control function must be allocated to an Rx/Tx endpoint pair (with the same endpoint number).
- The chosen endpoints must each be associated with FIFOs that can accommodate the packet size associated with EP0 transactions at the chosen operating speed. The size is a minimum of 8 bytes for low-speed transactions but up to 64 bytes for full-speed transactions.

Multi-Point Bandwidth Considerations

The available bandwidth determines the ability of a multi-point system to cope with isochronous transactions.

Once an endpoint is set up, hardware handles all scheduling. However, as with PC-based EHCI/OHCI/UHCI hosts, before opening a periodic pipe (for use by isochronous or interrupt traffic), software must determine that there is sufficient bandwidth available.

Further, if the periodic pipe is opened to a full-speed device through a high-speed hub, software must confirm that sufficient bandwidth is available. Bandwidth must be available on both on the local high-speed bus and the full-speed bus generated by the transaction translator in the hub.

The bandwidth required for different transactions can be determined using algorithms similar to the ones used with PC-based hosts (detailed in Section 5.11.3 of the USB 2.0 Specification).

The available bandwidth is greater where the hub used supports multiple transaction translators.

Babble Interrupt

If the bus is still active at the end of a frame, the USB controller assumes that the function it is connected to has malfunctioned. It suspends all transactions, and generates a babble interrupt (`USB_IRQ.RSTBABBLE`). The USB controller does not start a transaction until the bus is inactive for at least the minimum inter-packet delay. The controller also does not start a transaction unless it can be finished before the end of the frame.

To recover from a babble error condition, the processor must take the following actions inside the interrupt service routine.

1. Turn off VBUS. Wait until the VBUS level indicator reads b#01.
2. Turn on VBUS. Wait until the VBUS level indicator reads b#11.
3. Set the `USB_IRQ.SESSREQ` bit

The VBUS level indicator is the `USB_DEV_CTL.VBUS` bit field

NOTE: Because VBUS is sourced external to the processor, make sure that the hardware design connects a GPIO or the dedicated `USB_VBUS` signal to the external source. This connection enables software to turn VBUS on and off.

VBUS Events

The USB On-The-Go specification defines a series of thresholds to which the devices involved in point-to-point communications must respond.

- VBUS Valid (between 4.4 V and 4.75 V)
- Session Valid for A device (between 0.8 V and 2.1 V)
- Session End (between 0.2 V and 0.8 V)

The critical thresholds and the processor response depend on whether the device is an A device or a B device and the circumstances of the event. The following sections describe these actions.

Actions as an A Device

VBUS >VBUS Valid with session initiated by USB controller. VBUS level indicator = b#11 and the session bit is set. When VBUS is greater than VBUS valid, the USB controller selects host mode and waits for a device to connect. It then generates a connect interrupt. The processor resets and enumerates the connected B device.

VBUS > Session valid with session initiated by B device. VBUS level indicator = b#10 and the session bit is clear. When VBUS is greater than session valid, the USB controller generates a session request interrupt. The processor sets the session bit. The USB controller either stays in host mode or changes to peripheral mode, depending upon the state of the pull-up resistor on the B device. For more information, refer to the host negotiation protocol of the OTG specification. The state of the host mode bit indicates the selected mode.

VBUS below VBUS Valid while the Session bit remains set. VBUS level indicator b#11 and the session bit is set. This event indicates a problem with the VBUS power level. For example, the battery power could have dropped too low to sustain VBUS valid. Or, the B device could be drawing more current than the A device can provide. In either case, the USB controller automatically terminates the session and generates a VBUS error interrupt.

To recover from this VBUS error condition, the processor must take the following actions inside the VBUS error interrupt handler.

- Turn off VBUS and wait until the `USB_DEV_CTL.VBUS` reads b#01.
- Turn on VBUS and wait until the `USB_DEV_CTL.VBUS` reads b#11.
- Set the `USB_DEV_CTL.SESSION` bit

The `USB_DEV_CTL.VBUS` bit field indicates the VBUS level.

NOTE: Because VBUS is sourced external to the processor, make sure that the hardware design connects a GPIO or the dedicated `DrvVBUS` signal to the external source. Then, the software can be used to turn VBUS on and off.

Actions as a B Device

VBUS > Session Valid. VBUS level indicator = b#10 and session bit is clear. This event indicates activity from the A device. The USB controller sets the session bit and disconnects the pull down resistor on the D+ line.

VBUS < Session Valid. While the session bit remains set, VBUS level indicator = b#01 and session bit is set. This event indicates that the A device has lost power (or become disconnected). The USB controller clears the session bit and generates a disconnect interrupt. The processor ends the session.

VBUS < Session End. VBUS level indicator = b#00. This event is the condition under which a B device can initiate a session request. If the session bit is set, then after 2 ms of SE0 on the bus, the USB controller starts SRP by first pulsing the data line, then pulsing the `USB_VBUS` signal.

Host Mode Reset

If the USB_POWER.RESET is set while the USB controller is in host mode, the USB controller generates reset signaling on the bus. The processor core must keep this bit set for 20 ms to ensure correct resetting of the target device. After the processor core clears the bit, the USB controller starts its frame counter and transaction scheduler.

Host Mode Suspend

The controller has a suspend mode that allows power savings for the processor. The mode operates as follows.

Entry into Suspend mode. When operating as a host, the USB controller can be prompted to go into suspend mode by setting the USB_POWER.SUSPEND bit. When this bit is set, the USB controller completes the current transaction then stops the transaction scheduler and frame counter. No further transactions start. No SOF packets are generated. If the USB_POWER.SUSPEND bit is set, the UTMI+ PHY goes into low-power mode when the USB controller goes into suspend mode and stops the clock.

Sending Resume Signaling. When the application requires the USB controller to leave suspend mode, it must clear and then set the USB_POWER.RESUME bit, and leave it set for 20 ms. While the USB_POWER.RESUME bit is high, the USB controller generates resume signaling on the bus. After 20 ms, the processor core must clear the USB_POWER.RESUME bit, at which point the frame counter and transaction scheduler start.

Responding to Remote Wake-up. If resume signaling is detected from the target while the USB controller is in suspend mode, the UTMI+ PHY is brought out of low-power mode and the clock restarts. The USB controller then exits suspend mode and automatically sets the USB_POWER.RESUME bit to take over generating the resume signaling from the target. If the USB_IRQ.RESUME bit=1, software generates an interrupt.

Suspending and Resuming the Controller

With the introduction of link power management, there are two basic methods to suspend and resume the USB controller. The ***Basic LPM transaction*** diagram demonstrates these two methods.

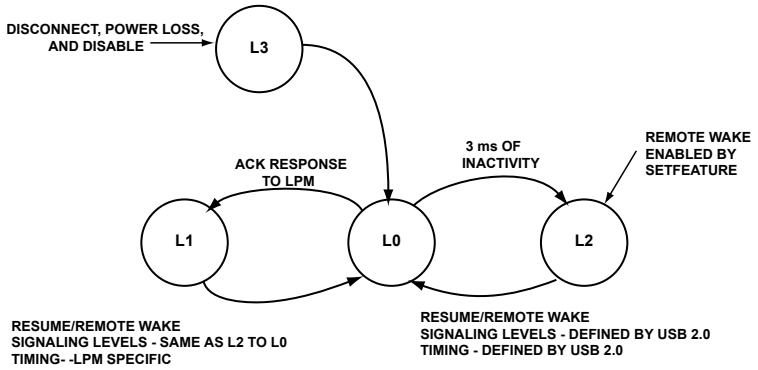

Figure 22-7: Basic LPM Transaction

The procedure that suspends and resumes the USB controller depends on whether the core operates as a device or a host, and the method of suspend desired. The following sections describe these options.

Suspend or Resume by Inactivity on the USB Bus (L0 to L2 State) in Peripheral Mode

The following steps occur in this mode.

1. Entry into suspend mode. When operating as a peripheral, the USB controller monitors activity on the USB and when no activity has occurred for 3 ms, the controller goes into suspend mode. If the USB_IRQ.SUSPEND interrupt has been enabled, the USB controller now generates an interrupt. The USB_IRQ.SUSPEND output also goes low (if enabled).

   The *POWERDWN* signal is also asserted to indicate that the application can stop USB_CLKIN to save power. *POWERDWN* then remains asserted until either power is removed from the bus (indicating that the device has been disconnected) or resume signaling or reset signaling is detected on the bus.

2. When resume signaling occurs on the bus, the USB_CLKIN must be restarted, if necessary. The USB controller then automatically exits suspend mode. If the USB_IRQ.RESUME interrupt is enabled, the USB controller generates an interrupt.

3. Initiating a remote wake-up. To initiate a remote wake-up while the controller is in suspend mode, set the USB_POWER.RESUME bit=1. (Note: If USB_CLKIN has been stopped, it must be restarted before this write can occur.) The software must leave then this bit set for approximately 10 ms (minimum of 2 ms, a maximum of 15 ms) before resetting it to 0. By this time the hub is driving resume signaling on the USB.

NOTE: The USB_IRQ.RESUME interrupt is not generated when the software initiates a remote wake-up.

Suspend or Resume by Inactivity on the USB Bus (L0 To L2 State) in Host Mode

The following steps occur in this mode.

1. Entry into suspend mode. When operating as a host, the USB controller can be prompted to go into suspend mode by setting the USB_POWER.SUSPEND bit. When this bit is set, the USB controller completes the current transaction then stops the transaction scheduler and frame counter. No further transactions start and no SOF packets are generated. If the USB_POWER.SUSEN bit is set, the UTMI+ PHY goes into low-power mode when the controller goes into suspend mode and stops USB_CLKIN.

2. Sending resume signaling. When the application requires the controller to leave suspend mode, it clears the USB_POWER.SUSPEND bit, sets the USB_POWER.RESUME bit, and leaves it set for 20 ms. While the USB_POWER.RESUME bit is high, the controller generates resume signaling on the bus. After 20 ms, the processor core must clear the USB_POWER.RESUME bit, at which point the frame counter and transaction scheduler start.

3. Responding to remote wake-up. If resume signaling is detected from the target while the USB controller is in suspend mode, the UTMI+ PHY is brought out of low-power mode and restarts USB_CLKIN. The controller then exits suspend mode and automatically sets the USB_POWER.RESUME bit to 1 to take over generating the resume signaling from the target. If the USB_IRQ.RESUME interrupt is enabled, the USB controller generates an interrupt.

Suspend or Resume by an LPM Transaction (L0 To L1 State) in Peripheral Mode

The following steps occur in this mode.

1. Enter into suspend mode. When operating as a peripheral, the controller never initiates an LPM suspend (transition from the L0 state to the L1 state). Rather, the controller only suspends at the request of the host. Configure the USB_LPM_CTL register appropriately to enable the LPM feature. The USB controller uses the register field USB_LPM_CTL.EN bit to enable and support extended and LPM transactions. The USB controller uses the USB_LPM_CTL.TX field to instruct the hardware that it is ready to suspend and to respond to the next LPM transaction with an ACK. In this case, the controller responds to the next LPM transaction with an ACK if all other conditions are met. The ***Response to LPM Transaction*** table summarizes the response of the USB controller to an LPM transaction.

Table 22-6: Response to LPM Transaction

LPMXMT	LPMCNTRL	Data Pending (Resides in Tx FIFOs)	Response to Next LPM Transaction
1'b0, 1'b0 1'b1 1'b1	2'b00, 2'b10 2'b00 2'b10	Do-not-care	Timeout
1'b0, 1'b1	2'b01	Do-not-care	STALL
1'b0	2'b11	Do-not-care	NYET
1'b1	2'b11	Yes	NYET
1'b1	2'b11	No	ACK

For all cases in the table in which the controller responds (no timeout occurs), an LPM interrupt is generated in the USB_LPM_IRQ register. The controller responds with an ACK only if there is no data pending in any of the Tx endpoint FIFOs. If there is data pending, the USB controller responds with a NYET.

Once an LPM transaction is successfully received, three events occur:

a. The USB_LPM_ATTR register is updated with values received in the LPM transaction. See the "Register Descriptions" section of this chapter for complete information on this register.

b. The controller suspends 9 µs after transmitting the ACK. The host or the controller can drive resume signaling 50 µs after this event. During the 9 µs interval, the host can continue to transmit the LPM transaction. The controller responds with an ACK in this case regardless of the USB_LPM_CTL.TX bit value.

c. An interrupt is generated informing software of the response (an ACK in this case). An ACK response is the indication to software that the controller has suspended.

   Since the primary purpose of LPM is to save power, software reads the USB_LPM_ATTR register to determine the attributes of the suspend. Software must make a determination based on these attributes whether there are more potential power savings in the system. In making this determination, note that if the host

initiates the resume signaling, the controller must respond to packet transmissions within the time specified by USB_LPM_ATTR.HIRD + 10 µs.

2. When resume signaling occurs on the bus. When the host resumes the bus, it drives resume signaling for a minimum time specified by the host initiated resume duration bit field (USB_LPM_ATTR.HIRD). The controller must be able to respond to traffic within the time HIRD + 10 µs. The controller transitions to a normal operating state automatically and a resume interrupt is generated in the USB_LPM_IRQ register.

   However for this event to occur, the inputs *CLK* and *XCLK* must be available. To facilitate the resume timing requirement, a negative ACK (NAK) is provided using the USB_LPM_CTL.NAK bit. If this bit is set to 1'b1, all endpoints respond to any transaction (other then an LPM) with a NAK. This bit only takes effect after the controller has suspended LPM. Typically, this bit is asserted when the USB_LPM_CTL.TX field is also asserted. Using this feature can simplify the resume timing requirement because the controller only needs *XCLK* to respond (with a NAK) to traffic. Software can continue to restore the system to normal operation while the controller responds to all transactions with a NAK. After software completely restores the system, it can then clear the USB_LPM_CTL.NAK bit.

3. Initiating remote wake-up. To initiate a remote wake-up while in suspend mode, the controller writes a 1'b1 to theUSB_LPM_CTL.RESUME bit. This bit is self clearing. Writing a 1'b1 drives resume signaling on the bus for 50 µs. The host responds by driving resume for 60 µs to 990 µs. 10 µs after the host stops driving resume, the controller transitions to its normal operational state and is ready for packet transmission. A resume interrupt is generated in the USB_LPM_IRQ register.

Suspend or Resume by an LPM Transaction (L0 to L1 State) in Host Mode

The following steps occur in this mode.

1. Enter into suspend mode. When operating as a host, the controller initiates an LPM suspend (transition from the L0 state to the L1 state) by initiating an LPM transaction as follows.

   a. Software sets up the desired attributes of the suspend in the USB_LPM_ATTR register. Enabling remote wake-up and a large HIRD gives the peripheral more opportunity to conserve power.

   b. Enable all LPM interrupts in the USB_LPM_IEN register.

   c. Software writes 0x01 to the USB_LPM_CTL register to initiate the transaction.

   d. An interrupt is generated to inform software of the response to the LPM transaction. If an ACK was received, then the controller suspends automatically within 8 µs. This event indicates that the controller has suspended.

   If the response from the device has a bit stuff error or a PID error, then an USB_LPM_IRQ.LPMERR interrupt is generated. The hardware immediately attempts the LPM transaction two more times. The device does not suspend for 8 µs after the initial LPM so it can respond to either of these subsequent LPM transactions. If an LPM timeout has occurred three times, the USB_LPM_IRQ.LPMNC and the USB_LPM_IRQ.LPMERR interrupts are set. Now, software is unaware of the device state and must deduce it by other means.

2. Send resume signaling. Software generates resume signaling as follows.

a. Enable all LPM interrupts in the USB_LPM_IEN register.

b. Software writes to the USB_LPM_CTL.RESUME bit which is self-clearing. This operation causes resume signaling on the bus for the time specified in the USB_LPM_ATTR.HIRD bit field. Hardware assumes that the last LPM transaction that caused the suspend used this value.

c. After HIRD + 10 µs, the controller transitions to its normal operational state and is ready for packet transmission and a USB_LPM_IRQ.LPMRES interrupt is generated.

NOTE: Prior to resuming, software must ensure that the system is restored from a low-power state and that the inputs *CLK* and *XCLK* are available.

3. Responding to remote wake-up. If the remote wake-up feature is enabled in the LPM transaction that caused the suspend, then the device can drive resume signaling on the bus. When this event occurs, the device drives resume signaling BUS for 50 µs. The controller immediately begins driving resume signaling on the BUS and continues for 60 µs. 10 µs after completion of the resume signaling, the controller transitions to its normal operating state and is ready for packet transmission. Then, the USB_LPM_IRQ.LPMRES interrupt is generated.

USB Event Control

The following sections provide information on the use of interrupts, reset, and the reporting of errors and interface status.

Interrupt Signals

The "Interrupt Table" section at the beginning of this chapter shows the two interrupts generated from the USB controller.

The software generates interrupts from control endpoint zero under the following conditions

- When a control transaction ends before the end of the data is transferred.
- When a data packet is sent or received from the endpoint 0 FIFOs.

The USB controller generates interrupts from transmit endpoints (USB_INTRTX) under the following conditions:

- A packet is sent from the TX FIFO (host and peripheral mode)
- After three attempts at transmitting a packet, no valid handshake packet is received (host mode)

The software generates interrupts from receive endpoints (USB_INTRRX) under the following conditions:

- A packet is received into the RX FIFO (host and peripheral mode)
- A stall handshake is received (host mode)
- After three attempts at receiving a packet, no data packet is received (host mode).

The software generates interrupts from the USB status (USB_IRQ) under the following conditions:

- When VBUS drops below the VBUS valid threshold during a session (A device only).

- When SRP signaling is detected (A device only).
- When device disconnect is detected (host mode).
- When a session ends (peripheral mode).
- When a device connection is detected (host mode).
- At start-of-frame (SOF)
- When reset signaling is detected on USB (peripheral mode).
- When babble is detected (host mode).
- In suspend mode, when resume signaling is detected on USB.
- When suspend signaling is detected (peripheral mode).

The software generates interrupts for the following VBUS control requests:

- Drive VBUS greater than 4.4 V (default A device)
- Stop driving VBUS
- Start charging VBUS (peripheral mode)
- Stop charging VBUS
- Start discharging VBUS (peripheral mode)
- Stop discharging VBUS

Interrupt Handling

When interrupted with a USB interrupt, the processor core must read the interrupt status register to determine which endpoints have caused the interrupt and jump to the appropriate routine. If multiple endpoints have caused the interrupt, endpoint 0 must be serviced first, followed by the other endpoints. The ***USB Interrupt Service Routine*** figure shows a flowchart for the USB interrupt service routine.

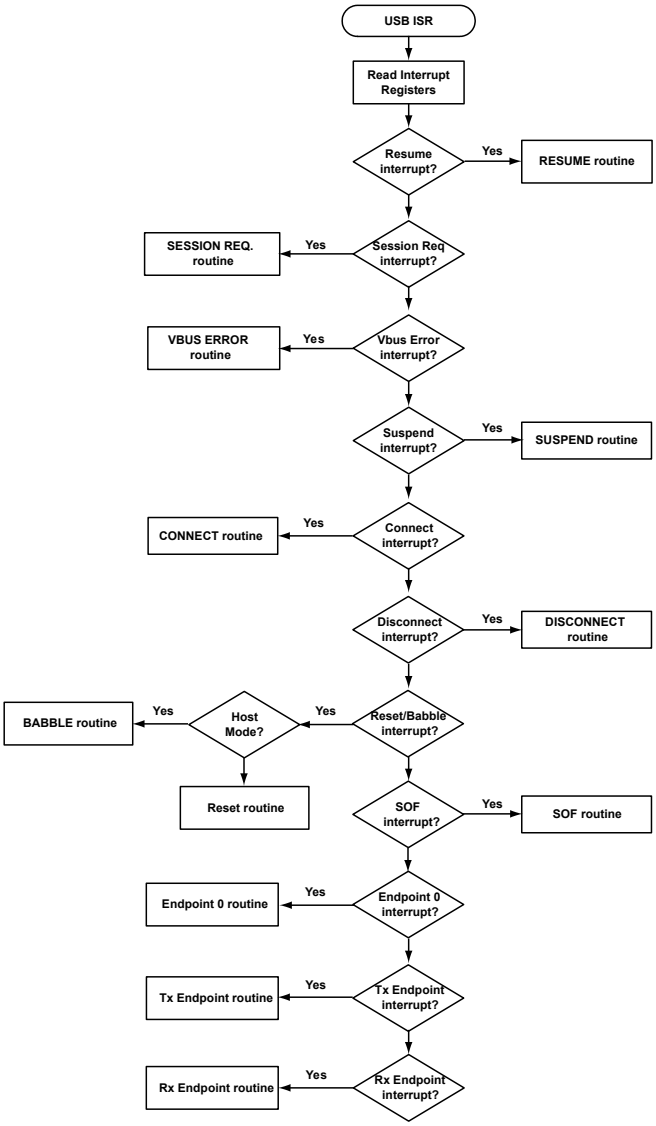

Figure 22-8: USB Interrupt Service Routine

Reset Signals

The USB controller includes an active-high synchronous hardware reset sourced from the processor core. Another source of peripheral reset is through the USB, when USB reset signaling is detected on the I/O lines. Per the USB 2.0 Specification, this state is entered when both the D+ and D- inputs are driven low for 2.5 ms or more. (The USB host typically holds the reset for greater than 10 ms).

Reset in Peripheral Mode

When the USB controller detects a reset, it performs the following actions:

- Sets the `USB_FADDR` register to zero
- Sets the `USB_INDEX` register to zero

- Flushes all endpoint FIFOs
- Clears all control and status registers
- Enables all interrupts
- Generates a reset interrupt

The USB controller reset does not affect the USB_IRQ and USB_VBUS_CTL registers. These registers are only reset (along with the ones listed) during a system reset.

When the application software receives a reset interrupt, it closes any open pipes and waits for bus enumeration to begin.

USB Reset in Host Mode

If the USB_POWER.RESET bit =1 while the USB controller is in host mode, the controller generates reset signaling on the bus.

The processor core must keep the USB_POWER.RESET bit set for at least 20 ms to ensure correct resetting of the target device. After the processor core clears the bit, the USB controller starts its frame counter and transaction scheduler.

USB Programming Model

The following sections describe the USB OTG programming model.

Peripheral Mode Flow Charts

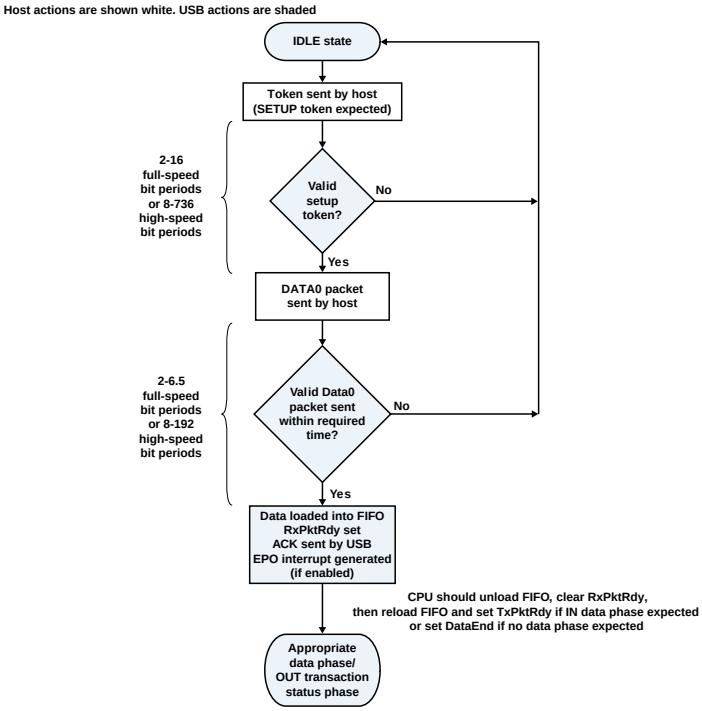

Figure 22-9: USB Control Setup Phase

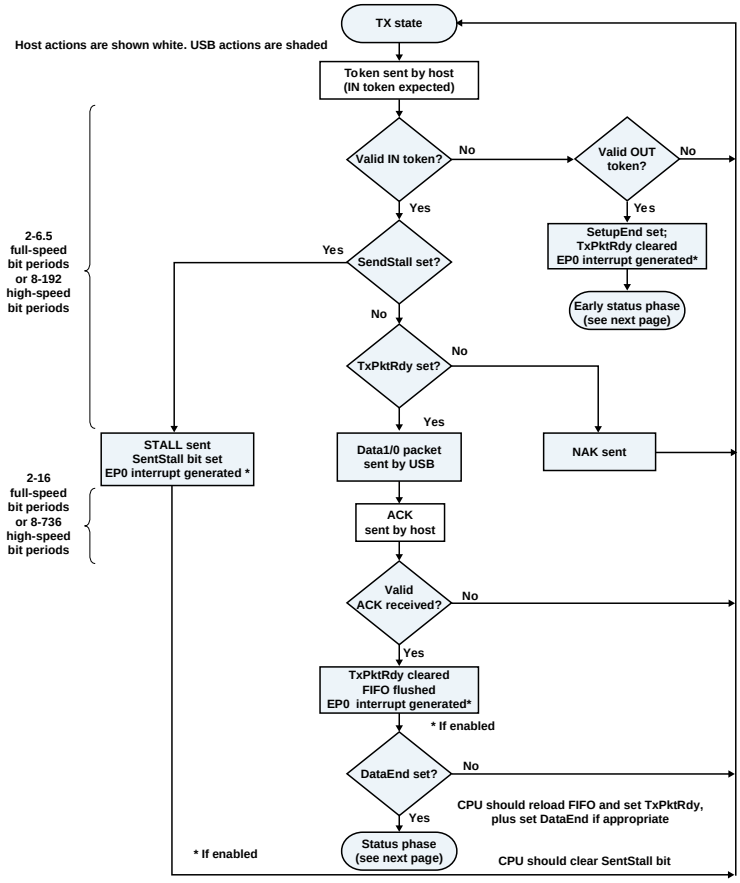

Figure 22-10: Control In Data Phase

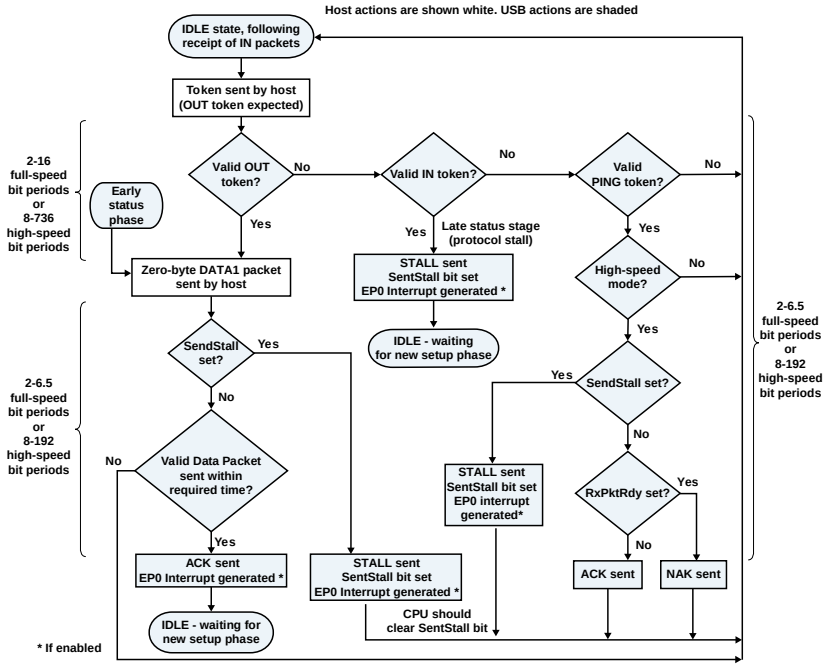

Figure 22-11: Control In Status Phase

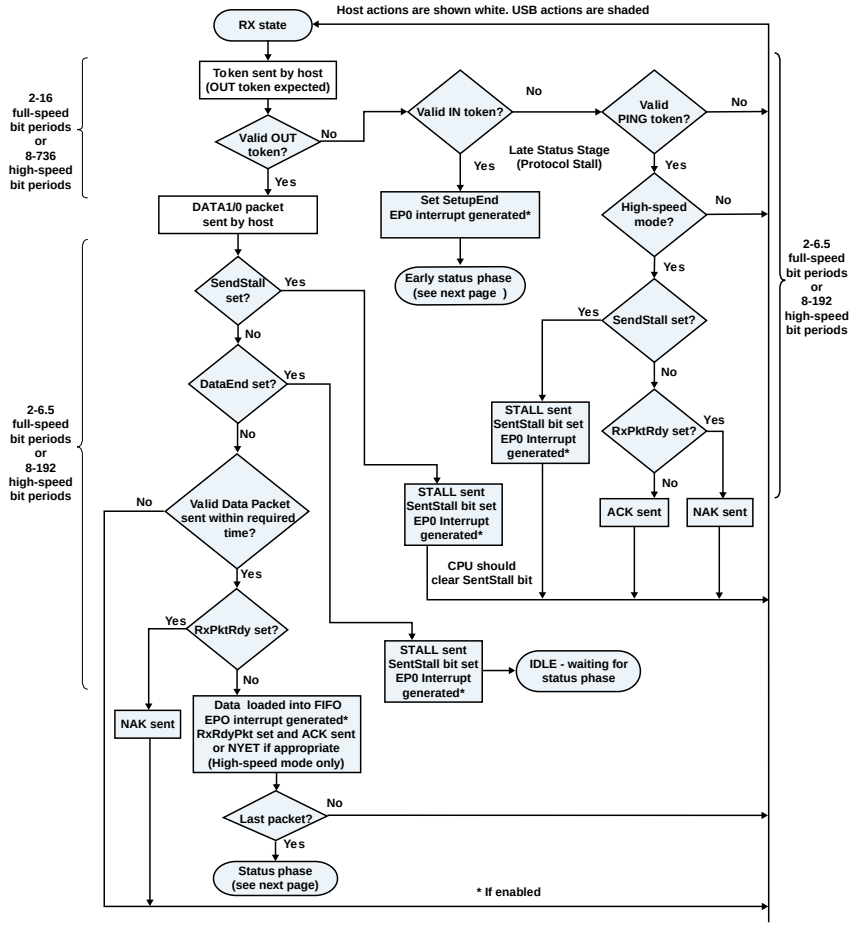

Figure 22-12: Control Out Data Phase

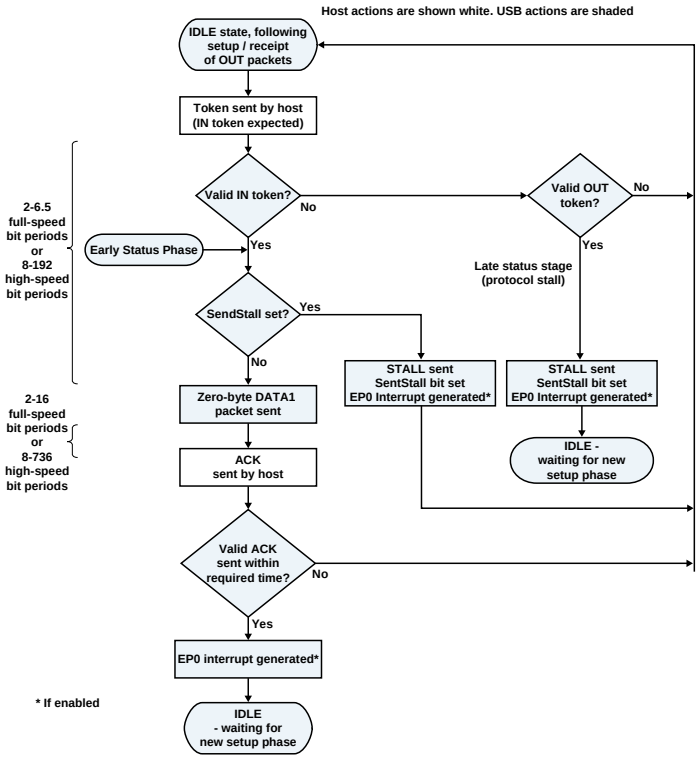

Figure 22-13: Control Out Status Phase

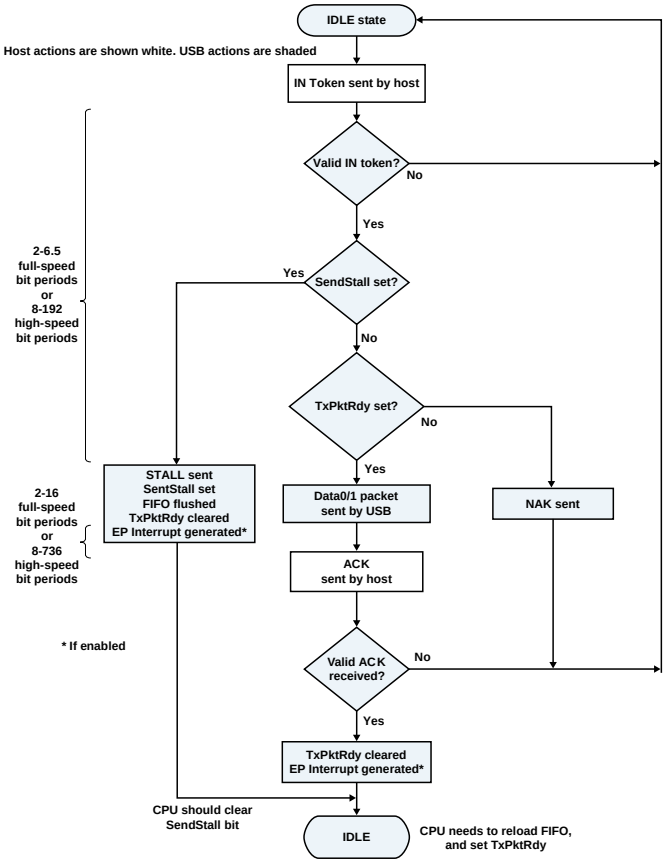

Figure 22-14: Bulk/Low Bandwidth Interrupt In Transaction

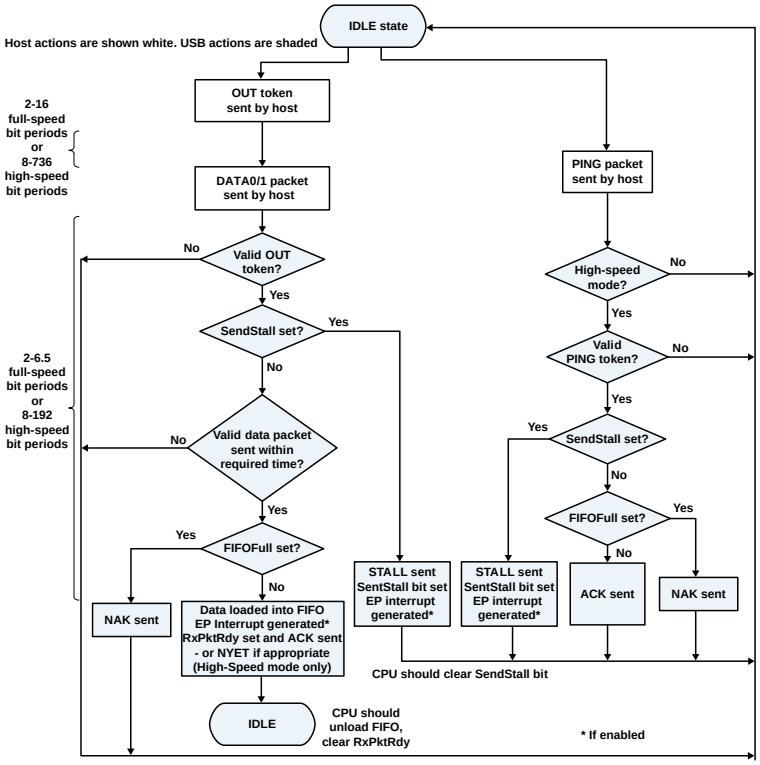

Figure 22-15: Bulk/Low Bandwidth Interrupt Out Transaction

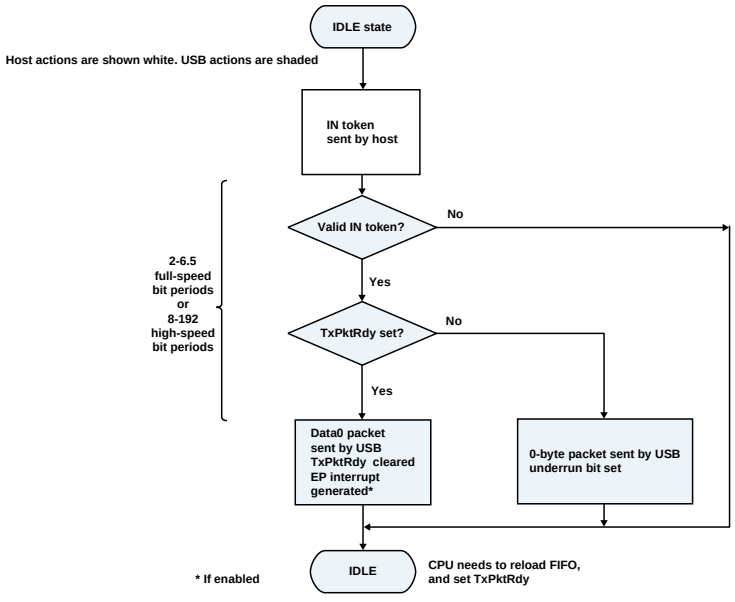

Figure 22-16: Full-speed/Low Bandwidth Isochronous In Transaction

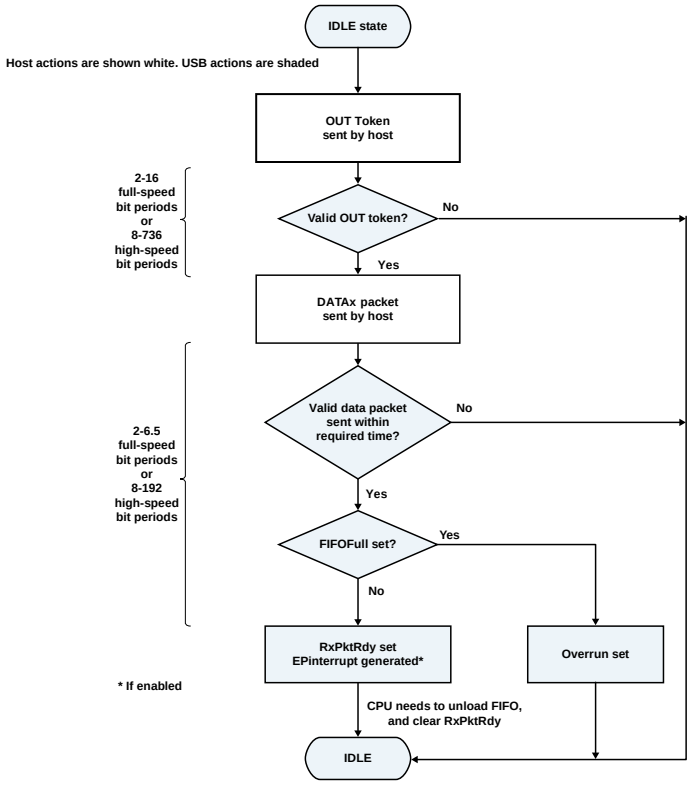

Figure 22-17: Full-speed/Low Bandwidth Isochronous Out Transaction

Host Mode Flow Charts

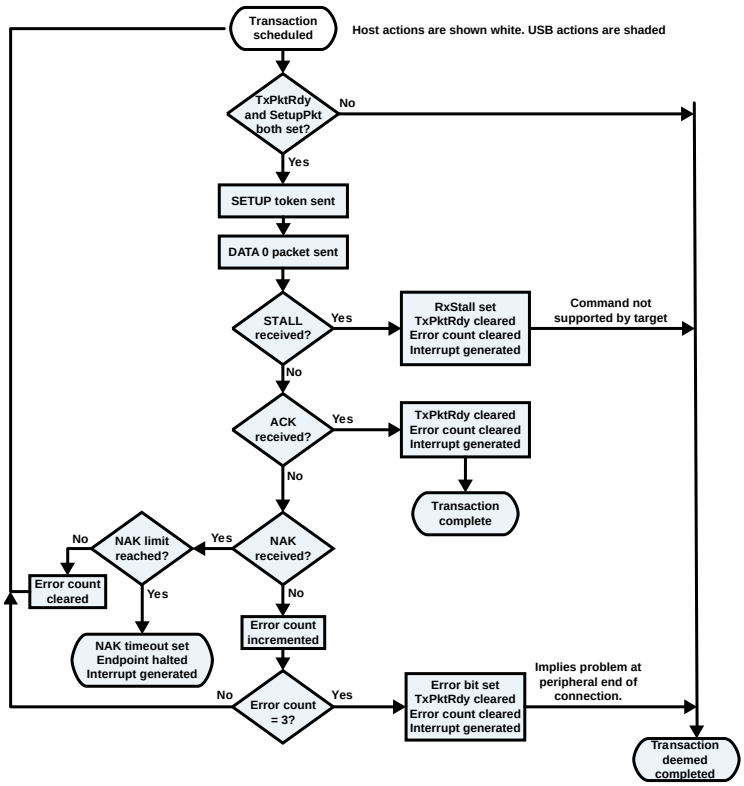

Figure 22-18: USB Control Setup Phase

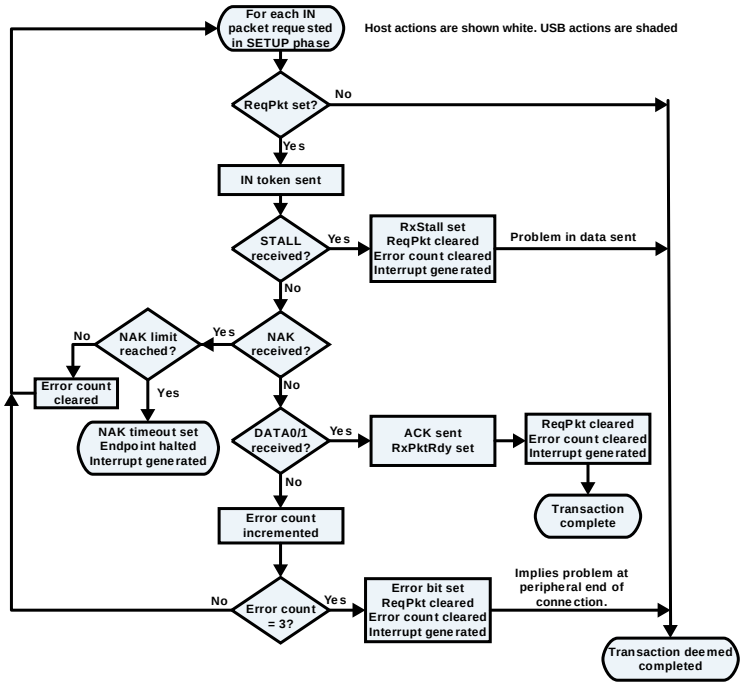

Figure 22-19: Control In Data Phase

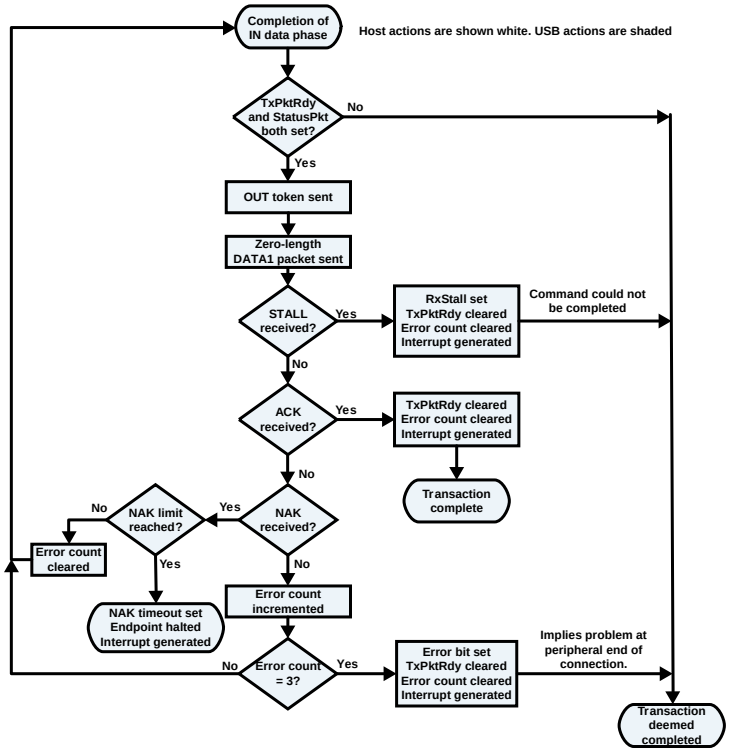

Figure 22-20: Control In Data Status Phase

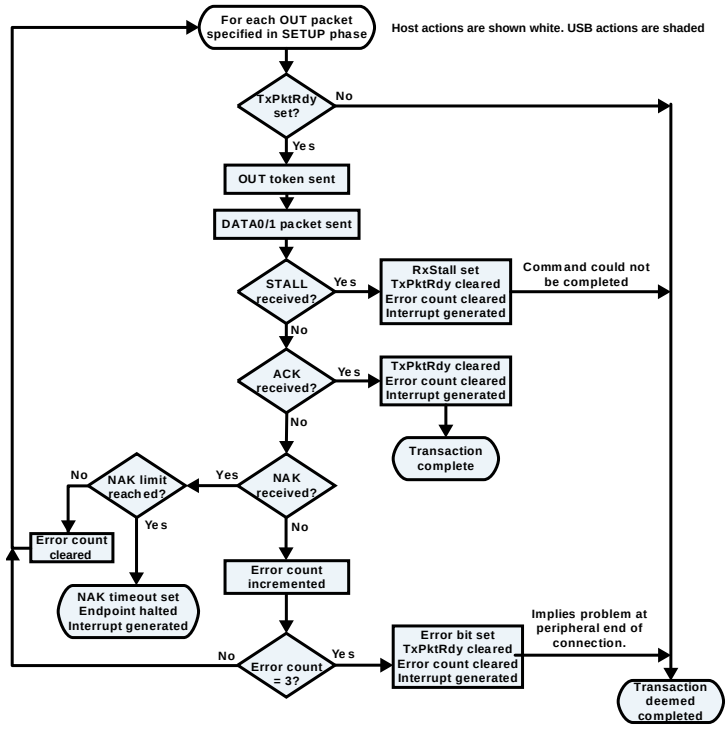

Figure 22-21: Control Out Data Phase

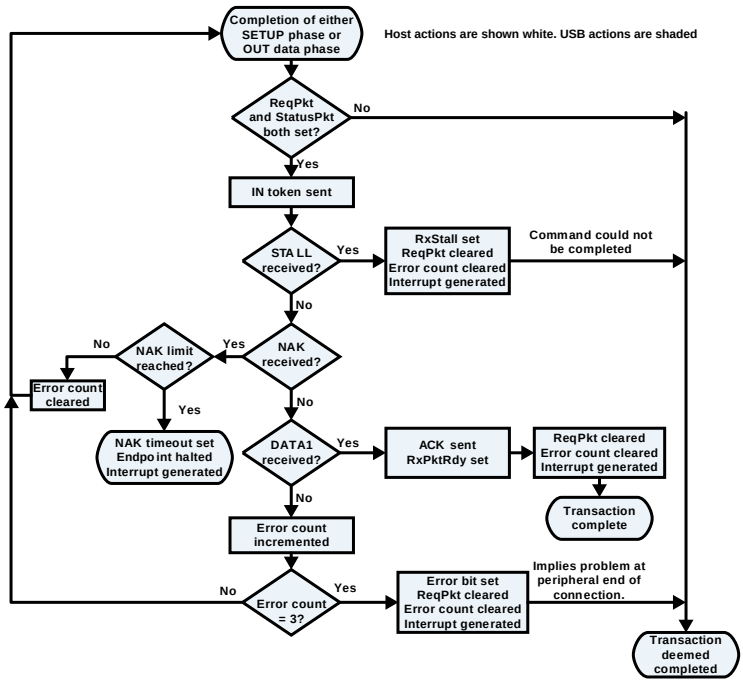

Figure 22-22: Control Out Data Status Phase

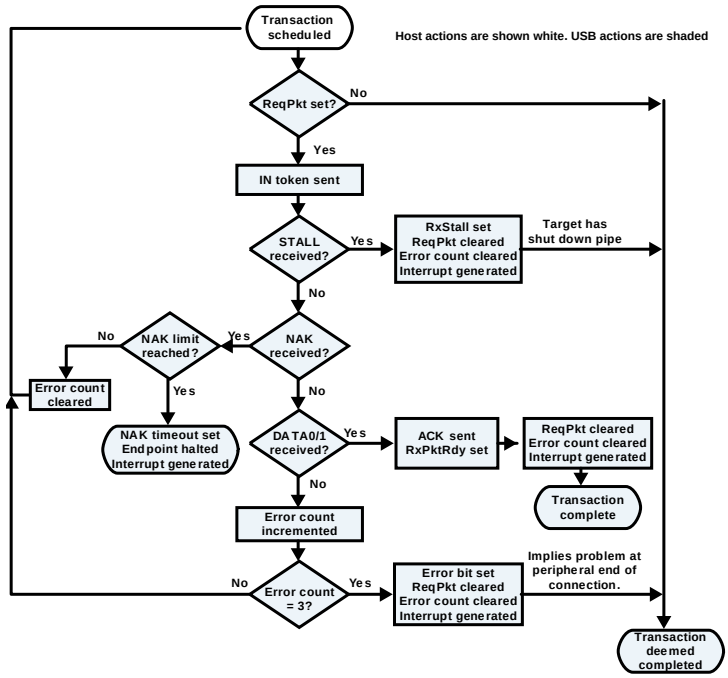

Figure 22-23: Bulk/Low Bandwidth Interrupt In Transaction

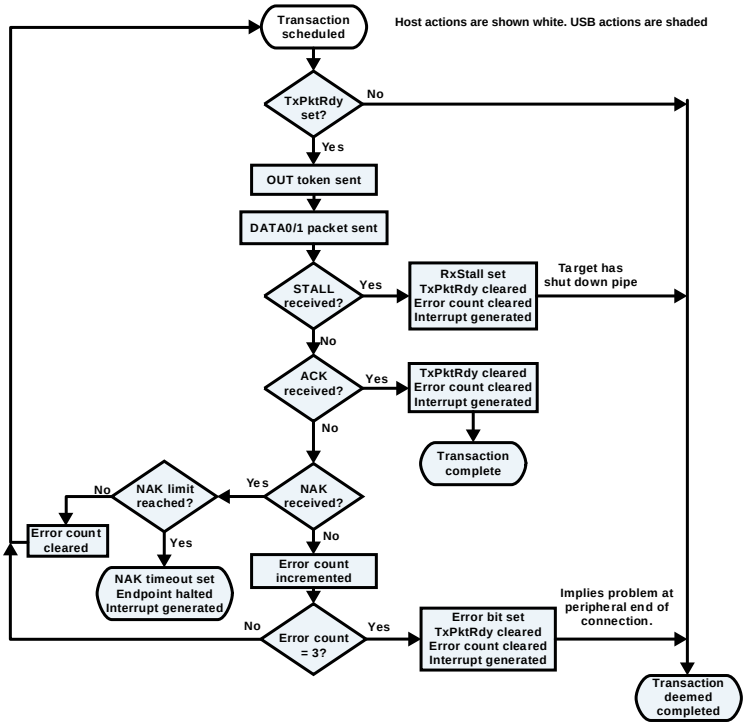

Figure 22-24: Bulk/Low Bandwidth Interrupt Out Transaction

Figure 22-25: Full-speed/Low Bandwidth Isochronous In Transaction

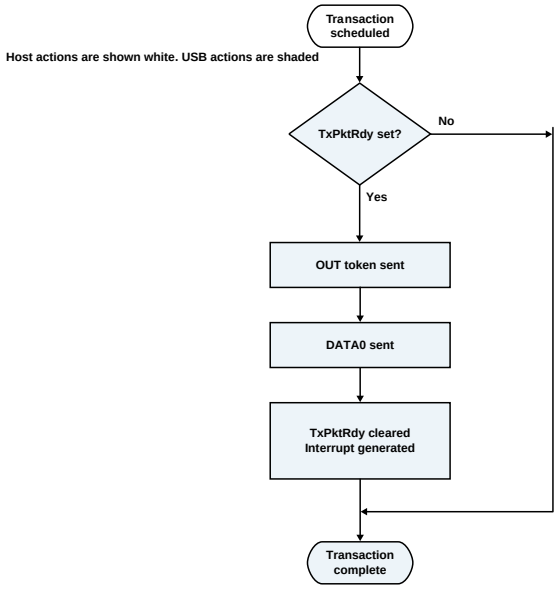

Figure 22-26: Full-speed/Low Bandwidth Isochronous Out Transaction

DMA Mode Flow Charts

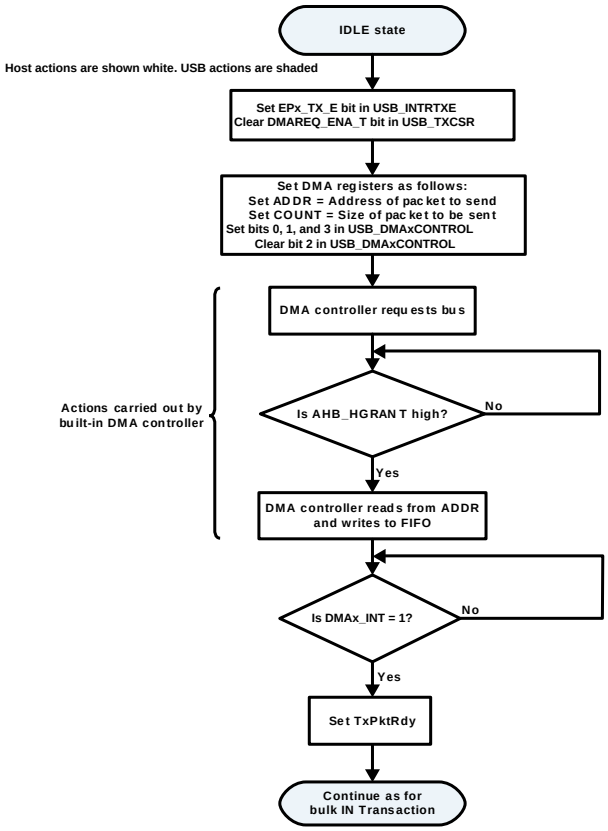

Figure 22-27: Single Packet Transmit During DMA Operation

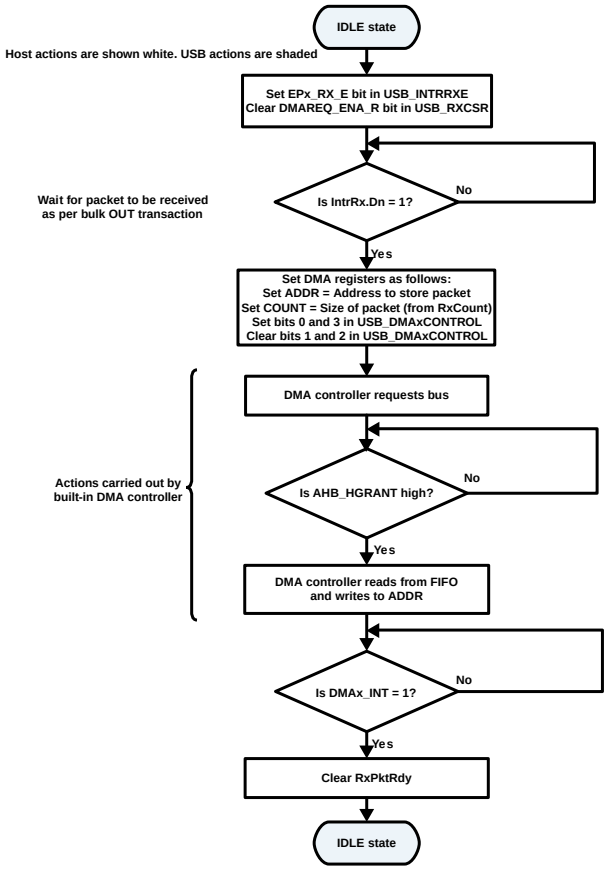

Figure 22-28: Single Packet Receive During DMA Operation

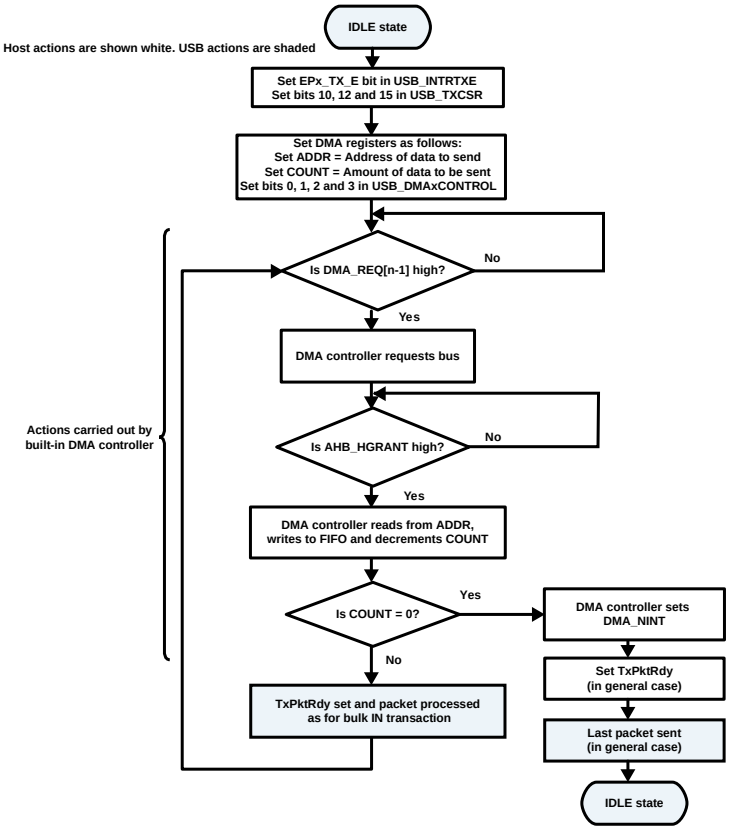

Figure 22-29: Multiple Packet Transmit During DMA Operation

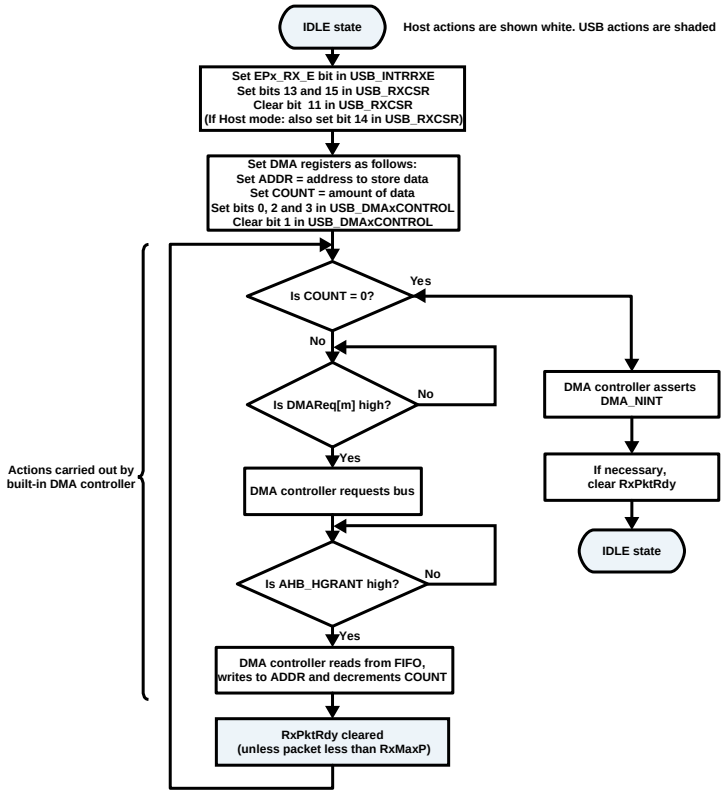

Figure 22-30: Multiple Packet Receive During DMA Operation (Data Size Known)

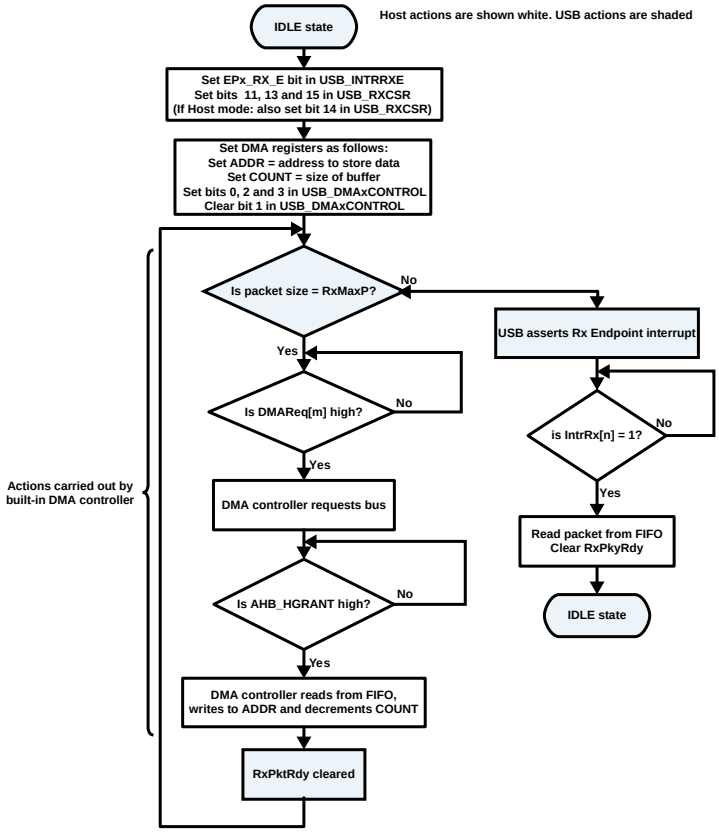

Figure 22-31: Multiple Packet Receive During DMA Operation (Data Size Not-known)

OTG Session Request

To conserve power, the USB on-the-go supplement allows VBUS to only power-up when required and to turn off when the bus is not in use.

The A device on the bus always supplies VBUS. The USB controller samples the `USB_ID` input from the PHY to determine whether it is the A device or the B device. The signal is pulled low when an A-type plug is sensed (signifying that the USB controller is the A device). The input is taken high when a B-type plug is sensed (signifying that the USB controller is the B device).

Starting a Session

When the device containing the USB controller wants to start a session, the processor core must set the `USB_DEV_CTL.SESSION` bit. The USB controller then enables ID pin sensing. This activity results in the `USB_ID` input either being taken low if an A-type connection is detected or high if a B-type connection is detected. The `USB_DEV_CTL.BDEVICE` bit is also set to indicate whether the USB controller has adopted the role of the A device or the B device.

The USB controller is the A device. The USB controller then enters host mode (the A device is always the default host). It waits for VBUS to go above the VBUS valid threshold, as indicated when the `USB_DEV_CTL.VBUS` bits go to 11.

The USB controller then waits for a peripheral to be connected. When the USB controller detects a peripheral, it generates a connect interrupt (USB_IRQ.CON bit) (if enabled). It sets either the USB_DEV_CTL.FSDEV or USB_DEV_CTL.LSDEV bits, depending on whether a full-speed peripheral or a low-speed peripheral was detected. The processor core then resets this peripheral. To end the session, the processor core must clear the USB_DEV_CTL.SESSION bit.

The USB controller is the B device. The USB controller requests a session using the session request protocol defined in the USB on-the-go supplement. This functionality is accomplished by setting the USB_DEV_CTL.SESSION bit.

At the end of the session, typically the USB controller clears the USB_DEV_CTL.SESSION bit. But, the processor core can also clear it when the application software must perform a software disconnect. For more information, see the description of the USB_DEV_CTL register. The USB controller switches on the pull-up resistor on D+. This activity signals to the A device to end the session.

Detecting Activity

When the other device of the OTG set-up wants to start a session, USB controller either:

- Raises VBUS above the session valid threshold (if A device) or
- First pulses the data line, then pulses VBUS (if B device)

Depending on which of these actions happens, the USB controller can determine whether it is the A device or the B device in the current set-up and act accordingly. (The USB_DEV_CTL.VBUS bits=10 indicates if it is the A device.)

If VBUS is raised above the session valid threshold, the USB controller is the B device. The USB controller sets the USB_DEV_CTL.SESSION bit. When reset signaling is detected on the bus, a reset interrupt (USB_IRQ.RSTBABBLE=1) is generated (if enabled) that the processor core interprets as the start of a session. The USB controller is in peripheral mode as the B device is the default peripheral.

At the end of the session, the A device turns off the power to VBUS. When VBUS drops below the session valid threshold, the USB controller detects this state and clears the USB_DEV_CTL.SESSION bit to indicate that the session has ended. (The USB_DEV_CTL.VBUS bits=01 indicates that VBUS has dropped below the session valid threshold). A disconnect interrupt (USB_IRQ.DISCON bit) is also generated (if enabled).

If data line or VBUS pulsing is detected, the USB controller is the A device. The controller generates a USB_IRQ.SESSREQ interrupt to indicate that the B device is requesting a session. The processor core must then start a session by setting the USB_DEV_CTL.SESSION bit.

Host Negotiation Protocol

When the USB controller is the A device (USB_ID low, USB_DEV_CTL.BDEVICE=0), the controller automatically enters host mode when a session starts.

When the USB controller is the B device (USB_ID high, USB_DEV_CTL.BDEVICE=1), the controller automatically enters peripheral mode when a session starts. The processor core can request that the USB controller become the host by setting the USB_DEV_CTL.HOSTREQ bit. This bit can be set either when requesting a session start by setting the USB_DEV_CTL.SESSION bit or at any time after a session has started.

When the USB controller enters suspend mode (no activity on the bus for 3 ms), and assuming the USB_DEV_CTL.HOSTREQ bit remains set, it then enters host mode. It begins host negotiation (as specified in the USB OTG supplement), causing the PHY to disconnect the pull-up resistor on the D+ line. This event causes the A device to switch to peripheral mode and to connect its own pull-up resistor. When the USB controller detects this activity, it generates a connect interrupt (USB_IRQ.CON bit). The controller also sets the USB_POWER.RESET bit to begin resetting the A device. (The USB controller begins this reset sequence automatically to ensure that reset is started as required within 1 ms of the A device connecting its pull-up resistor). The processor core must wait at least 20 ms, then clear the USB_POWER.RESET bit and enumerate the A device.

When the USB controller-based B device has finished using the bus, the processor core must put it into suspend mode by setting the USB_POWER.SUSPEND. The A device detects this state and either terminates the session or reverts to host mode. If the A device is USB controller-based, it generates a disconnect interrupt (USB_IRQ.DISCON bit) if enabled.

Data Transfer

Whether the USB controller is operating in host or peripheral mode, data channels through the endpoint FIFOs to construct packets that are sent or received over the USB. The USB controller uses the Rx FIFOs to receive OUT packets when in peripheral mode and IN packets when operating in host mode. Similarly, The USB controller uses the Tx FIFOs to transmit IN packets when in peripheral mode and OUT packets as a host.

Data can be moved between the FIFOs and memory using either DMA or core accesses. Each endpoint FIFO has its own individually programmable options so that each can be set up separately. The system must treat each transfer type differently. Data transfers of significant size almost certainly require DMA to move the data around; but the processor can handle smaller packet sizes completely.

Each data endpoint supports both double and single-buffering modes. In single-buffered operation, the processor loads and unloads FIFOs on a packet-by-packet basis. Double-buffering imposes less burden on the system by allowing two packets to be buffered in a FIFO before it is necessary to use DMA or interrupts to service the FIFO. Double-buffering mode is automatically enabled when a *MaxPktSize* is set for an endpoint that is equal to or less than half the size in bytes of that FIFO.

Loading or Unloading Packets from Endpoints

Transfers to and from the FIFOs can be 32-bit, 16-bit, or 8-bit. When using core accesses, use the same width for transfers associated with one data packet, so that data is consistently byte, half-word, or word aligned. The last transfer can, however, contain fewer bytes than the previous transfers in order to complete an 8-bit or 16-bit transfer.

When using the DMA to access the FIFOs, the starting DMA address must be word aligned, or aligned on a 32-bit boundary. The packet transfer starts with a word transfer, but half-word or byte transfers can be added at the end to handle any leftovers.

DMA Master Channels

The USB controller provides seven DMA master channels.

These channels provide a more efficient transfer of larger amounts of data between the FIFOs and the processor core, and the channels free up the processor core for other tasks. The processor uses the DMA control registers to configure and control each of these channels.

Each DMA controller can operate in one of two DMA modes: 0 or 1. When operating in mode 0, the DMA controller can only be programmed to load or unload one packet, so processor intervention is required for each packet transferred over the USB. The DMA controller can use this mode with any endpoint, whether it uses control, bulk, isochronous, or interrupt transactions.

When operating in DMA mode 1, the DMA controller can only be programmed to load or unload a complete bulk transfer, which can be many packets. After set up, the DMA controller loads or unloads the packets, interrupting the processor only when the transfer has completed. DMA mode 1 can only be used with endpoints that use bulk transactions. It is most valuable where large blocks of data are transferred to a bulk endpoint. The USB protocol requires splitting such packets into a series of packets of *MaxPktSize* for the endpoint.

The DMA controller can use mode 1 to avoid the overhead of having to interrupt the processor after each individual packet. It interrupts the processor only after the transfer completes. In some cases, the block of data transferred consists of a predefined number of these packets that the controlling software counts through the transfer process. In other cases, the last packet in the series can be less than the maximum packet size. The receiver can use this short packet to signal the end of the transfer. If the total size of the transfer is an exact multiple of the maximum packet size, the transmitting software must send a null packet for the receiver to detect.

NOTE: Each channel can be independently programmed for the selected operating mode.

For bulk OUT transfers using DMA mode 1, the DMA request line is asserted only when:

- There is an edge transition of the state of the `USB_EP[n]_RXCSR_H.RXPKTRDY`, and
- A payload of *MaxPktSize* has been received

If a data packet is in the FIFO prior to setting the DMA request mode bits, the DMA request line is not asserted when the DMA is enabled. DMA is enabled using the `USB_DMA[n]_CTL.EN` bit. (DMA request mode bits are `USB_EP[n]_RXCSR_H.DMAREQMODE` or `USB_EP[n]_RXCSR_P.DMAREQMODE`). The data is not read from the Rx FIFO in this situation, resulting in a DMA hang. However, since the packet arrived before DMA request mode and DMA request enable bits (`USB_EP[n]_RXCSR_H.DMAREQEN` or `USB_EP[n]_RXCSR_P.DMAREQEN`) were enabled, an Rx interrupt is generated for the corresponding endpoint. Therefore, the software must set the DMA request mode to request mode 0 to unload the pre-received packet. The Rx interrupt service routine can be similar to the following:

If `USB_EP[n]_RXCNT` = MaxPktSize

Switch to DMA mode 0 and unload the packet (in mode 0, the DMA request enable is always asserted, whenever there is data in the FIFO)

Set the `USB_EP[n]_RXCNT` to MaxPktSize so as to unload only one packet

If `USB_EP[n]_RXCSR_H.AUTOCLR` is set, it is not necessary to clear `USB_EP[n]_RXCSR_H.RXPKTRDY` manually

Switch back to DMA Mode 1 and set the count to

(Total_Count – MaxPktSize)

Else

Handle as normal for case of short packet

DMA transfers can be 8-bit, 16-bit, or 32-bit. All transfers associated with one packet (except for the last) must be of the same width, so that the data is consistently byte-aligned or word-aligned. The last transfer can contain fewer bytes than the previous transfers to complete an odd-byte or odd-word transfer.

DMA Bus Cycles

The DMA controller uses incrementing bursts of an unspecified length on the peripheral DMA bus. The controller starts a new burst when it is first granted bus mastership and when the peripheral address starts a new 1 KB block. (The controller is granted bus mastership at the start of a USB packet or when regaining the bus after losing it following a partial packet).

When unloading packets from the FIFOs, the DMA controller requests ahead to the USB controller. The DMA controller starts the transfer with two BUSY cycles while it is gets the first word from the FIFO. All subsequent words of the packet are immediately available and no further BUSY cycles are required. The DMA controller is associated with a two-word buffer, so it does not lose data when it loses bus mastership in the middle of unloading a packet. When the controller regains bus mastership, it can continue unloading the packet without adding any BUSY cycles.

The DMA start address (written to the `USB_DMA[n]_ADDR` register) must be word aligned. The DMA controller supports split transactions and retries.

The DMA request lines are individually enabled using the appropriate DMA request enable bit (there are four options: Tx peripheral and host and Rx peripheral and host) and operate in two modes, referred to as DMA request mode 0 and DMA request mode 1. The operating mode is configured using the appropriate DMA request mode bit (there are four options: Tx peripheral and host and Rx peripheral and host).

NOTE: When operating in host mode, if the `USB_EP[n]_TXCSR_H.RXSTALL` bit or the `USB_EP[n]_TXCSR_H.TXTOERR` is set following three failed transmit attempts, the DMA request line is disabled until the bits are cleared.

The mode selected also affects the generation of endpoint interrupts (if enabled). In DMA request mode 0, no interrupt is generated when packets are received but the appropriate endpoint interrupt is generated to prompt the loading of all packets. In DMA request mode 1, the endpoint interrupt is suppressed except following the receipt of a short packet (one less than `USB_EP[n]_RXMAXP` bytes).

Table 22-7: Endpoint Interrupt Associated with the Receive Packet Ready Bit=1

DMAReqEnab	DMAReqMode	EP Interrupt Generated?
0	X	YES
1	0	NO

Table 22-7: Endpoint Interrupt Associated with the Receive Packet Ready Bit=1 (Continued)

DMAReqEnab	DMAReqMode	EP Interrupt Generated?
1	1	Only is short packet

Table 22-8: Endpoint Interrupt Associated with the Receive Packet Ready Bit=0

DMAReqEnab	DMAReqMode	EP Interrupt Generated?
0	X	YES
1	0	YES
1	1	NO

NOTE: Set the USB_EP[n]_TXMAXP/USB_EP[n]_RXMAXP registers to an even number of bytes for proper interrupt generation in DMA mode 1.

DMA transfers can be 8-bit, 16-bit, or 32-bit as required. However, all transfers associated with one packet (except for the last) must be of the same width so that the data is consistently byte-aligned, word-aligned, or double-word-aligned. The last transfer can contain fewer bytes than the previous transfers to complete an odd-byte or odd-word transfer.

NOTE: Disable DMA requests before changing the DMA request mode bit. In particular, do not set the USB_EP[n]_TXCSR_H.DMAREQMODE bit to zero either before or in the same cycle as the corresponding USB_EP[n]_TXCSR_H.DMAREQEN bit is cleared to zero.

Transferring Packets Using DMA

Both the channel and the endpoint must be programmed appropriately to use the the DMA master channels to access the USB controller FIFOs. Many variations are possible. The following sections detail the standard set ups used for the basic actions of transferring individual and multiple packets.

Individual Rx Endpoint Packet

The transfer of individual packets normally uses DMA mode 0. Program the USB controller Rx endpoint as follows.

1. Set to 1 the relevant bit in the USB_INTRRXE register.
2. Set to 0 the DMA enable bit of the appropriate USB_EP[n]_RXCSR_H.DMAREQEN/ USB_EP[n]_RXCSR_P.DMAREQEN register. (There is no need to set the USB controller to support DMA for this operation.)
3. When the USB controller receives a packet, it generates the appropriate endpoint interrupt (using the USB_INTRRXE register). The processor must then program the appropriate DMA master channel as follows.
   - Configure the USB_DMA[n]_ADDR register with the memory address to store the packet
   - Configure the USB_DMA[n]_CNT register with the size of packet (determined by reading the USB controller USB_RQPKTCNT[n] register)

- Configure the USB_DMA[n]_CTL register using the following bit settings:USB_DMA[n]_CTL.IE=1 USB_DMA[n]_CTL.EN=1, USB_DMA[n]_CTL.DIR=0, USB_DMA[n]_CTL.MODE=0

The DMA controller then requests bus mastership and transfers the packet to memory. It interrupts the processor when it has completed the transfer. The processor then clears the USB_EP[n]_RXCSR_H.RXPKTRDY bit.

Individual Tx Endpoint Packet

Using DMA mode 0, program a USB controller Tx endpoint as follows.

1. Set to 1 the relevant bit in the USB_INTRTXE register.
2. Set to 0 the DMA enable bit of the appropriate USB_EP[n]_TXCSR_H.DMAREQEN/ USB_EP[n]_TXCSR_P.DMAREQEN register. (There is no need to set the USB controller to support DMA for this operation.)
3. When the FIFO can accommodate data, the USB controller interrupts the processor with the appropriate Tx endpoint interrupt. The processor must then program the DMA channel as follows:
   - Configure the USB_DMA[n]_ADDR register with the memory address to store the packet.
   - Configure the USB_DMA[n]_CNT register with the size of packet.
   - Configure the USB_DMA[n]_CTL register using the following bit settings: USB_DMA[n]_CTL.IE=1 USB_DMA[n]_CTL.EN=1, USB_DMA[n]_CTL.DIR=1, USB_DMA[n]_CTL.MODE=0.

The DMA controller then requests bus mastership and transfers the packet to the USB controller FIFO. When it has completed the transfer, it generates a DMA interrupt. The processor then sets the USB_EP[n]_TXCSR_H.TXPKTRDY bit.

Multiple Rx Endpoint Packets

The transfer of multiple packets normally uses DMA mode 1. Program the DMA controller using the DMA registers:

- Configure the USB_DMA[n]_ADDR register with the memory address of data block to send.
- Configure the USB_DMA[n]_CNT register with the maximum size of data buffer.
- Configure the USB_DMA[n]_CTL register using the following bit settings: USB_DMA[n]_CTL.EN=1, USB_DMA[n]_CTL.IE=1, USB_DMA[n]_CTL.DIR=0, USB_DMA[n]_CTL.MODE=1.

Program the USB controller Rx endpoint as follows:

1. Set to 1 the relevant bit in the USB_INTRRX register.
2. Set to 1 the USB_EP[n]_RXCSR_H.AUTOCLR, USB_EP[n]_RXCSR_H.DMAREQEN, and USB_EP[n]_RXCSR_H.DMAREQMODE bits of the appropriate receive control and status register (host or peripheral). In host mode, set to 1 the USB_EP[n]_RXCSR_H.AUTOREQ and USB_EP[n]_RXCSR_H.DMAREQMODE bits.

As the USB controller receives each packet, the DMA master channel requests bus mastership and transfers the packet to memory. With USB_EP[n]_RXCSR_H.AUTOCLR set, the USB controller automatically clears its USB_EP[n]_RXCSR_H.RXPKTRDY bit. This process continues automatically until the USB controller receives a short packet (one of less than the maximum packet size for the endpoint) signifying the end of the transfer. The DMA controller does not transfer this short packet: instead the USB controller interrupts the processor by generating the appropriate endpoint interrupt. The processor can then read the USB_EP[n]_RXCNT register to see the size of the short packet. It either unloads the packet manually or reprograms the DMA controller in mode 0 to unload the packet.

The USB_DMA[n]_ADDR register is incremented as the DMA controller unloads the packets. The processor determines the size of the transfer by comparing the current value of USB_DMA[n]_ADDR with the start address of the memory buffer.

If the size of the transfer exceeds the data buffer size, the DMA controller stops unloading the FIFO and interrupts the processor.

Multiple Tx Endpoint Packets

Using DMA mode 1 for a Tx endpoint, program the DMA controller as follows:

- Configure the USB_DMA[n]_ADDR register with the memory address of data block to send.
- Configure the USB_DMA[n]_CNT register with the size of the data block.
- Configure the USB_DMA[n]_CTL register using the following bit settings: USB_DMA[n]_CTL.EN=1, USB_DMA[n]_CTL.IE=1 USB_DMA[n]_CTL.DIR=1, USB_DMA[n]_CTL.MODE=1.

Program the USB controller Tx endpoint as follows:

1. Set to 1 the relevant bit in the USB_INTRTXE register.
2. Set to 1 the USB_EP[n]_TXCSR_H.AUTOSET and USB_EP[n]_TXCSR_H.DMAREQEN bits of the appropriate transmit control and status register (host or peripheral).

When the FIFO in the USB controller becomes available, the DMA controller requests bus mastership and transfers a packet to the FIFO. With USB_EP[n]_TXCSR_H.AUTOSET set, the USB controller automatically sets the USB_EP[n]_TXCSR_H.TXPKTRDY bit. This process continues until the entire data block is transferred to the USB controller.

The DMA controller then interrupts the processor by taking the appropriate USB_DMA_IRQ register bit low.

- If the last packet loaded was less than the maximum packet size for the endpoint, the USB_EP[n]_TXCSR_H.TXPKTRDY bit is not set for this packet. The processor must respond to the DMA interrupt by setting the USB_EP[n]_TXCSR_H.TXPKTRDY bit to allow the last short packet to be sent.
- If the last packet loaded is the maximum packet size, then the appropriate action depends on whether the transfer is under the control of an application. One example is the mass storage software on Windows system that keeps count of the individual packets sent.

- If the transfer is not under such control, the processor must respond to the DMA interrupt by setting the USB_EP[n]_TXCSR_H.TXPKTRDY bit. This operation sends a null packet for the receiving software to interpret as indicating the end of the transfer.

ADSP-CM40x USB Register Descriptions

Universal Serial Bus Controller (USB) contains the following registers.

Table 22-9: ADSP-CM40x USB Register List

Name	Description
USB_DEV_CTL	Device Control Register
USB_DMA[n]_ADDR	DMA Channel n Address Register
USB_DMA[n]_CNT	DMA Channel n Count Register
USB_DMA[n]_CTL	DMA Channel n Control Register
USB_DMA_IRQ	DMA Interrupt Register
USB_EP0I_CFGDATA[N]	EP0 Configuration Information Register
USB_EP0I_CNT[N]	EP0 Number of Received Bytes Register
USB_EP0I_CSR[N]_H	EP0 Configuration and Status (Host) Register
USB_EP0I_CSR[N]_P	EP0 Configuration and Status (Peripheral) Register
USB_EP0I_NAKLIMIT[N]	EP0 NAK Limit Register
USB_EP0I_TYPE[N]	EP0 Connection Type Register
USB_EP0_CFGDATA[n]	EP0 Configuration Information Register
USB_EP0_CNT[n]	EP0 Number of Received Bytes Register
USB_EP0_CSR[n]_H	EP0 Configuration and Status (Host) Register
USB_EP0_CSR[n]_P	EP0 Configuration and Status (Peripheral) Register
USB_EP0_NAKLIMIT[n]	EP0 NAK Limit Register
USB_EP0_TYPE[n]	EP0 Connection Type Register
USB_EPINFO	Endpoint Information Register
USB_EPI[N]_RXCNT	EPn Number of Bytes Received Register
USB_EPI[N]_RXCSR_H	EPn Receive Configuration and Status (Host) Register
USB_EPI[N]_RXCSR_P	EPn Receive Configuration and Status (Peripheral) Register
USB_EPI[N]_RXINTERVAL	EPn Receive Polling Interval Register
USB_EPI[N]_RXMAXP	EPn Receive Maximum Packet Length Register
USB_EPI[N]_RXTYPE	EPn Receive Type Register
USB_EPI[N]_TXCSR_H	EPn Transmit Configuration and Status (Host) Register

Table 22-9: ADSP-CM40x USB Register List (Continued)

Name	Description
USB_EPI[N]_TXCSR_P	EPn Transmit Configuration and Status (Peripheral) Register
USB_EPI[N]_TXINTERVAL	EPn Transmit Polling Interval Register
USB_EPI[N]_TXMAXP	EPn Transmit Maximum Packet Length Register
USB_EPI[N]_TXTYPE	EPn Transmit Type Register
USB_EPI_FIFOSZ	FIFO Size
USB_EP[n]_FIFOSZ	FIFO Size
USB_EP[n]_RXCNT	EPn Number of Bytes Received Register
USB_EP[n]_RXCSR_H	EPn Receive Configuration and Status (Host) Register
USB_EP[n]_RXCSR_P	EPn Receive Configuration and Status (Peripheral) Register
USB_EP[n]_RXINTERVAL	EPn Receive Polling Interval Register
USB_EP[n]_RXMAXP	EPn Receive Maximum Packet Length Register
USB_EP[n]_RXTYPE	EPn Receive Type Register
USB_EP[n]_TXCSR_H	EPn Transmit Configuration and Status (Host) Register
USB_EP[n]_TXCSR_P	EPn Transmit Configuration and Status (Peripheral) Register
USB_EP[n]_TXINTERVAL	EPn Transmit Polling Interval Register
USB_EP[n]_TXMAXP	EPn Transmit Maximum Packet Length Register
USB_EP[n]_TXTYPE	EPn Transmit Type Register
USB_FADDR	Function Address Register
USB_FIFOB[n]	FIFO Byte (8-Bit) Register
USB_FIFOH[n]	FIFO Half-Word (16-Bit) Register
USB_FIFO[n]	FIFO Word (32-Bit) Register
USB_FRAME	Frame Number Register
USB_FS_EOF1	Full-Speed EOF 1 Register
USB_IDCTL	ID Control
USB_IEN	Common Interrupts Enable Register
USB_INDEX	Index Register
USB_INTRRX	Receive Interrupt Register
USB_INTRRXE	Receive Interrupt Enable Register
USB_INTRTX	Transmit Interrupt Register
USB_INTRTXE	Transmit Interrupt Enable Register
USB_IRQ	Common Interrupts Register

Table 22-9: ADSP-CM40x USB Register List (Continued)

Name	Description
USB_LINKINFO	Link Information Register
USB_LPM_ATTR	LPM Attribute Register
USB_LPM_CTL	LPM Control Register
USB_LPM_FADDR	LPM Function Address Register
USB_LPM_IEN	LPM Interrupt Enable Register
USB_LPM_IRQ	LPM Interrupt Status Register
USB_LS_EOF1	Low-Speed EOF 1 Register
USB_MP[n]_RXFUNCADDR	MPn Receive Function Address Register
USB_MP[n]_RXHUBADDR	MPn Receive Hub Address Register
USB_MP[n]_RXHUBPORT	MPn Receive Hub Port Register
USB_MP[n]_TXFUNCADDR	MPn Transmit Function Address Register
USB_MP[n]_TXHUBADDR	MPn Transmit Hub Address Register
USB_MP[n]_TXHUBPORT	MPn Transmit Hub Port Register
USB_PHY_CTL	FS PHY Control
USB_PHY_STAT	FS PHY Status
USB_POWER	Power and Device Control Register
USB_RAMINFO	RAM Information Register
USB_RQPKTCNT[n]	EPn Request Packet Count Register
USB_RXDPKTBUFDIS	RX Double Packet Buffer Disable for Endpoints 1 to 3
USB_SOFT_RST	Software Reset Register
USB_TESTMODE	Testmode Register
USB_TXDPKTBUFDIS	TX Double Packet Buffer Disable for Endpoints 1 to 3
USB_VBUS_CTL	VBUS Control Register
USB_VPLEN	VBUS Pulse Length Register

Device Control Register

The USB_DEV_CTL register selects whether the USB controller is operating in peripheral mode or in host mode and is used for controlling and monitoring the VBUS line.

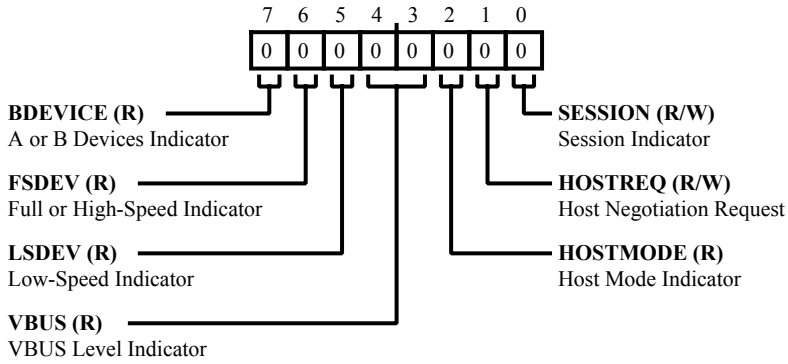

Figure 22-32: USB_DEV_CTL Register Diagram

Table 22-10: USB_DEV_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	BDEVICE	A or B Devices Indicator. The USB_DEV_CTL.BDEVICE bit indicates whether the USB controller is operating as the A device or the B device. This bit is only valid while a session is in progress.	
		0	A Device Detected
		1	B Device Detected
6 (R/NW)	FSDEV	Full or High-Speed Indicator. The USB_DEV_CTL.FSDEV bit is set when a full-speed or high-speed device is detected being connected to the port. High speed devices are distinguished from full-speed by checking for high-speed chirps when the device detects a USB controller reset. This bit is only valid in host mode.	
		0	Not Detected
		1	Full or High Speed Detected
5 (R/NW)	LSDEV	Low-Speed Indicator. The USB_DEV_CTL.LSDEV bit is set when a low-speed device is detected being connected to the port. This bit is only valid in host mode.	
		0	Not Detected
		1	Low Speed Detected

Table 22-10: USB_DEV_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4:3 (R/NW)	VBUS	VBUS Level Indicator. The USB_DEV_CTL.VBUS bits indicated the current VBUS level.	
		0	Below SessionEnd
		1	Above SessionEnd, below AValid
		2	Above AValid, below VBUSValid
		3	Above VBUSValid
2 (R/NW)	HOSTMODE	Host Mode Indicator. The USB_DEV_CTL.HOSTMODE bit is set when the USB controller is acting as a host.	
		0	Peripheral Mode
		1	Host Mode
1 (R/W)	HOSTREQ	Host Negotiation Request. When the USB_DEV_CTL.HOSTREQ bit is set, the USB controller initiates the host negotiation when Suspend mode is entered. This bit is cleared when host negotiation is completed. The USB_DEV_CTL.HOSTREQ bit applies when the USB controller is operating as a B device only.	
		0	No Request
		1	Place Request
0 (R/W)	SESSION	Session Indicator. When operating as an A device, the USB_DEV_CTL.SESSION is set or cleared by the processor core to start or end a session. When operating as a B device, the USB_DEV_CTL.SESSION bit is set or cleared by the USB controller when a session starts or ends. This bit is also set by the processor core to initiate the session request protocol. When the USB controller is in Suspend mode, the bit may be cleared by the processor core to perform a software disconnect.	
		0	Not Detected
		1	Detected Session

DMA Channel n Address Register

The USB_DMA[n]_ADDR register indicates the location in on-chip memory where DMA data is written or read. The address must be aligned to 32-bit words (The lower two address bits are always zero.) This register increments as the DMA transfer progresses.

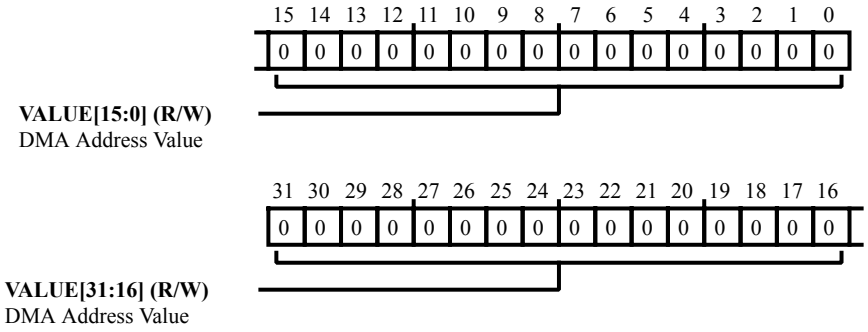

Figure 22-33: USB_DMA[n]_ADDR Register Diagram

Table 22-11: USB_DMA[n]_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	DMA Address Value. The USB_DMA[n]_ADDR.VALUE bits hold the address value for the location in on-chip memory where DMA data is written or read.

DMA Channel n Count Register

The USB_DMA[n]_CNT register holds the DMA count, indicating the number of bytes to be transferred for a given DMA work block. If this field is set to zero, no data is transferred, and an interrupt is generated.

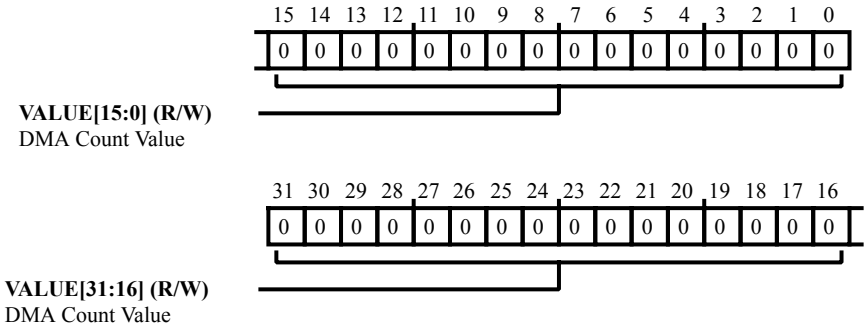

Figure 22-34: USB_DMA[n]_CNT Register Diagram

Table 22-12: USB_DMA[n]_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	DMA Count Value. The USB_DMA[n]_CNT.VALUE bits indicate the number of bytes to be transferred for a given DMA work block.

DMA Channel n Control Register

There is a USB_DMA[n]_CTL register for each DMA master channel. This register assigns, configures, and controls each endpoint with a corresponding DMA master channel.

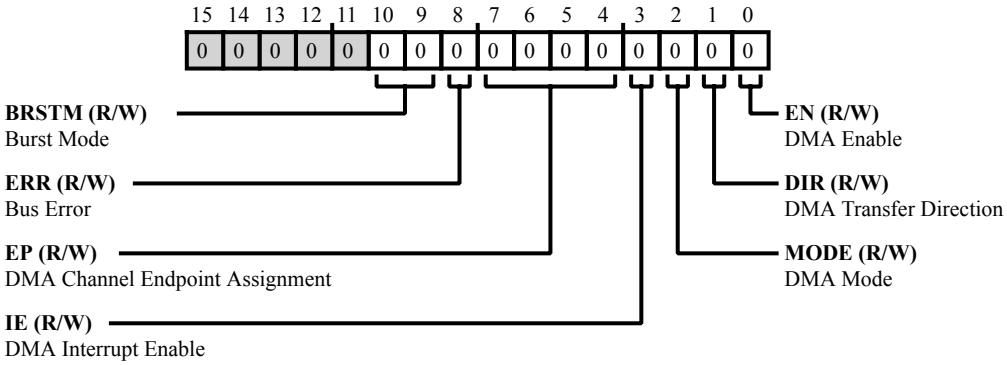

Figure 22-35: USB_DMA[n]_CTL Register Diagram

Table 22-13: USB_DMA[n]_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10:9 (R/W)	BRSTM	Burst Mode. The USB_DMA[n]_CTL.BRSTM bits select the type or length of burst transfer used by the corresponding DMA channel to transfer data.	
		0	Unspecified Length
		1	INCR4 or Unspecified Length
		2	INCR8, INCR4, or Unspecified Length
		3	INCR16, INCR8, INCR4, or Unspecified Length
8 (R/W)	ERR	Bus Error. The USB_DMA[n]_CTL.ERR bit indicates when a peripheral bus error has been encountered by the master channel. This bit is cleared by software.	
		0	No Status
		1	Bus Error
7:4 (R/W)	EP	DMA Channel Endpoint Assignment. The USB_DMA[n]_CTL.EP bits select the endpoint assignments for the DMA channel. (Enumeration values not shown are reserved.)	
		0	Endpoint 0
		1	Endpoint 1
		2	Endpoint 2
		3	Endpoint 3

Table 22-13: USB_DMA[n]_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		4	Endpoint 4
		5	Endpoint 5
		6	Endpoint 6
		7	Endpoint 7
		8	Endpoint 8
		9	Endpoint 9
		10	Endpoint 10
		11	Endpoint 11
		12	Endpoint 12
		13	Endpoint 13
		14	Endpoint 14
		15	Endpoint 15
3 (R/W)	IE	DMA Interrupt Enable. The `USB_DMA[n]_CTL.IE` bit enables DMA interrupts for the DMA channel, enabling operation of the channels corresponding bit in the `USB_DMA_IRQ` register.	
		0	Disable Interrupt
		1	Enable Interrupt
2 (R/W)	MODE	DMA Mode. The `USB_DMA[n]_CTL.MODE` bit selects whether the DMA channel operates in DMA mode 0 or operates in DMA mode 1. Note that DMA mode 1 may only be used with bulk endpoints.	
		0	DMA Mode 0
		1	DMA Mode 1
1 (R/W)	DIR	DMA Transfer Direction. The `USB_DMA[n]_CTL.DIR` bit selects the DMA channel transfer direction, which must be selected for use with receive endpoints (DMA write=0) or transmit endpoints (DMA read=1).	
		0	DMA Write (for Rx Endpoint)
		1	DMA Read (for Tx Endpoint)

Table 22-13: USB_DMA[n]_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	DMA Enable. The `USB_DMA[n]_CTL.EN` bit enables the DMA channel starts the DMA transfer.	
		0	Disable DMA
		1	Enable DMA (Start Transfer)

DMA Interrupt Register

The USB_DMA_IRQ register indicates which of the DMA master channels have a pending interrupt. The USB controller generates the interrupt when the corresponding DMA count register (USB_DMA[n]_CNT) reaches zero. The USB controller clears this register when it is read.

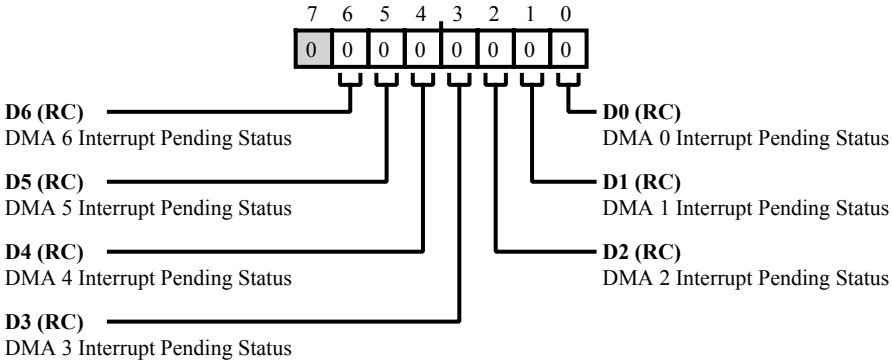

Figure 22-36: USB_DMA_IRQ Register Diagram

Table 22-14: USB_DMA_IRQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (RC/NW)	D6	DMA 6 Interrupt Pending Status. The USB_DMA_IRQ.D6 indicates whether there is a DMA 6 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
5 (RC/NW)	D5	DMA 5 Interrupt Pending Status. The USB_DMA_IRQ.D5 indicates whether there is a DMA 5 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
4 (RC/NW)	D4	DMA 4 Interrupt Pending Status. The USB_DMA_IRQ.D4 indicates whether there is a DMA 4 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
3 (RC/NW)	D3	DMA 3 Interrupt Pending Status. The USB_DMA_IRQ.D3 indicates whether there is a DMA 3 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt

Table 22-14: USB_DMA_IRQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (RC/NW)	D2	DMA 2 Interrupt Pending Status. The `USB_DMA_IRQ.D2` indicates whether there is a DMA 2 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
1 (RC/NW)	D1	DMA 1 Interrupt Pending Status. The `USB_DMA_IRQ.D1` indicates whether there is a DMA 1 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
0 (RC/NW)	D0	DMA 0 Interrupt Pending Status. The `USB_DMA_IRQ.D0` indicates whether there is a DMA 0 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt

EP0 Configuration Information Register

The USB_EP0I_CFGDATA[N] register describes the USB controller hardware configuration. This register only exists for endpoint 0.

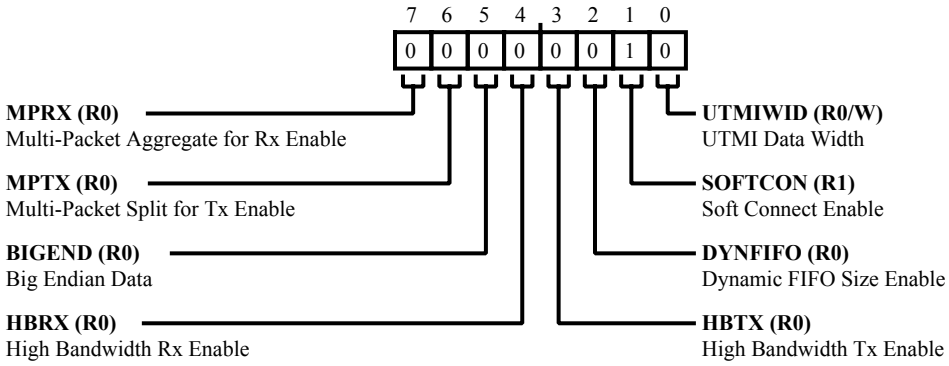

Figure 22-37: USB_EP0I_CFGDATA[N] Register Diagram

Table 22-15: USB_EP0I_CFGDATA[N] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R0/NW)	MPRX	Multi-Packet Aggregate for Rx Enable. The USB_EP0I_CFGDATA[N].MPRX bit indicates whether the USB controller aggregates receive packets into bulk packets before the processor core reads the data.	
		0	No Aggregate Rx Bulk Packets
		1	Aggregate Rx Bulk Packets
6 (R0/NW)	MPTX	Multi-Packet Split for Tx Enable. The USB_EP0I_CFGDATA[N].MPTX bit indicates whether the USB controller permits transmit of large packets through writing to bulk endpoints. The USB controller splits the transmit data into packets, which are appropriately sized for transmit.	
		0	No Split Tx Bulk Packets
		1	Split Tx Bulk Packets
5 (R0/NW)	BIGEND	Big Endian Data. The USB_EP0I_CFGDATA[N].BIGEND bit indicates whether the USB controller uses big endian configuration or little endian configuration.	
		0	Little Endian Configuration
		1	Big Endian Configuration

Table 22-15: USB_EP0I_CFGDATA[N] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R0/NW)	HBRX	High Bandwidth Rx Enable. The `USB_EP0I_CFGDATA[N].HBRX` bit indicates whether the USB controller supports high-bandwidth receive ISO endpoint support.	
		0	No High Bandwidth Rx
		1	High Bandwidth Rx
3 (R0/NW)	HBTX	High Bandwidth Tx Enable. The `USB_EP0I_CFGDATA[N].HBTX` bit indicates whether the USB controller supports high bandwidth transmit ISO endpoint support.	
		0	No High Bandwidth Tx
		1	High Bandwidth Tx
2 (R0/NW)	DYNFIFO	Dynamic FIFO Size Enable. The `USB_EP0I_CFGDATA[N].DYNFIFO` bit indicates whether the USB controller uses dynamic FIFO size support (on products supporting this feature), enabling the dynamic FIFO registers. These registers are accessed using the configuration set in the endpoints indexed FIFO size and FIFO address registers, except for endpoint 0.	
		0	No Dynamic FIFO Size
		1	Dynamic FIFO Size
1 (R1/NW)	SOFTCON	Soft Connect Enable. The `USB_EP0I_CFGDATA[N].SOFTCON` bit indicates whether the USB controller uses soft connect.	
		0	No Soft Connect
		1	Soft Connect
0 (R0/W)	UTMIWID	UTMI Data Width. The `USB_EP0I_CFGDATA[N].UTMIWID` bit indicates whether the USB controller uses an 8-bit or 16-bit UTMI data width.	
		0	8-bit UTMI Data Width
		1	16-bit UTMI Data Width

EP0 Number of Received Bytes Register

The USB_EP0I_CNT[N] register indicates the number of received data bytes in the endpoint 0 FIFO. The value returned changes as the contents of the FIFO change and is only valid while the USB_EP0_CSR[n]_H.RXPKTRDY bit or USB_EP0_CSR[n]_P.RXPKTRDY bit is set.

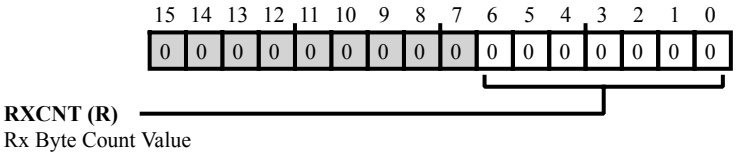

Figure 22-38: USB_EP0I_CNT[N] Register Diagram

Table 22-16: USB_EP0I_CNT[N] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/NW)	RXCNT	Rx Byte Count Value. The USB_EP0I_CNT[N].RXCNT bits holds the number of data bytes currently inline ready to be read from the Rx FIFO. The value returned changes as the FIFO is unloaded and is only valid while USB_EP0_CSR[n]_H.RXPKTRDY bit or USB_EP0_CSR[n]_P.RXPKTRDY bit is set.

EP0 Configuration and Status (Host) Register

The USB_EP0I_CSR[N]_H register provides control and status bits for endpoint 0 in host mode. Note that some bits may be set to clear automatically.

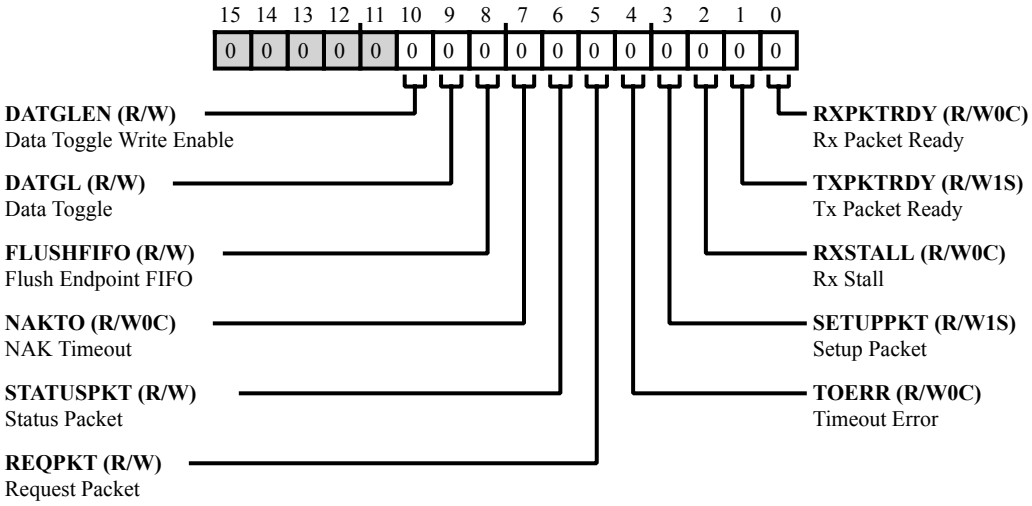

Figure 22-39: USB_EP0I_CSR[N]_H Register Diagram

Table 22-17: USB_EP0I_CSR[N]_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W)	DATGLEN	Data Toggle Write Enable. The USB_EP0I_CSR[N]_H.DATGLEN bit enables (in host mode) the USB controller to write the current state of the endpoint 0 USB_EP0I_CSR[N]_H.DATGL bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL
9 (R/W)	DATGL	Data Toggle. The USB_EP0I_CSR[N]_H.DATGL bit indicates (in host mode) the current state of the endpoint 0 data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is Set
		1	DATA1 is Set

Table 22-17: USB_EP0I_CSR[N]_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EP0I_CSR[N]_H.FLUSHFIFO` bit directs (in host mode) the USB controller to flush data from the endpoint 0 FIFO and clear the `USB_EP0I_CSR[N]_H.TXPKTRDY` and `USB_EP0I_CSR[N]_H.RXPKTRDY` bits. The `USB_EP0I_CSR[N]_H.FLUSHFIFO` bit should only be set if the `USB_EP0I_CSR[N]_H.TXPKTRDY` and `USB_EP0I_CSR[N]_H.RXPKTRDY` bits are set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
7 (R/W0C)	NAKTO	NAK Timeout. The `USB_EP0I_CSR[N]_H.NAKTO` bit indicates (in host mode) when endpoint 0 is halted following the receipt of NAK responses for longer than the time set by the `USB_EP0_NAKLIMIT[n]` register. The processor core should clear this bit to allow the endpoint to continue.	
		0	No Status
		1	Endpoint Halted (NAK Timeout)
6 (R/W)	STATUSPKT	Status Packet. The `USB_EP0I_CSR[N]_H.STATUSPKT` bit directs (in host mode) the USB controller to perform a status stage transaction. This bit is set at the same time as the `USB_EP0I_CSR[N]_H.TXPKTRDY` `USB_EP0I_CSR[N]_H.RXPKTRDY`. Setting this bit ensures that the data toggle is set to 1 so that a DATA1 packet is used for the status stage transaction.	
		0	No Request
		1	Request Status Transaction
5 (R/W)	REQPKT	Request Packet. The `USB_EP0I_CSR[N]_H.REQPKT` bit directs (in host mode) the USB controller to request an IN transaction. This bit is cleared when `USB_EP0I_CSR[N]_H.RXPKTRDY` is set.	
		0	No Request
		1	Send IN Tokens to Device
4 (R/W0C)	TOERR	Timeout Error. The `USB_EP0I_CSR[N]_H.TOERR` bit indicates (in host mode) when three attempts have been made to perform a transaction with no response from the peripheral. The processor core should clear this bit. An interrupt is generated when this bit is set.	
		0	No Status
		1	Timeout Error

Table 22-17: USB_EP0I_CSR[N]_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W1S)	SETUPPKT	Setup Packet. The `USB_EP0I_CSR[N]_H.SETUPPKT` bit directs (in host mode) the USB controller to send a SETUP token instead of an OUT token for the transaction. This bit is set at the same time as the `USB_EP0I_CSR[N]_H.TXPKTRDY` bit is set.	
		0	No Request
		1	Send SETUP token
2 (R/W0C)	RXSTALL	Rx Stall. The `USB_EP0I_CSR[N]_H.RXSTALL` bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
1 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EP0I_CSR[N]_H.TXPKTRDY` bit should be set (in host mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EP0I_CSR[N]_H.RXPKTRDY` is set (in host mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EP0 Configuration and Status (Peripheral) Register

The USB_EP0I_CSR[N]_P register provides control and status bits for endpoint 0 in peripheral mode. Note that some bits may be set to clear automatically.

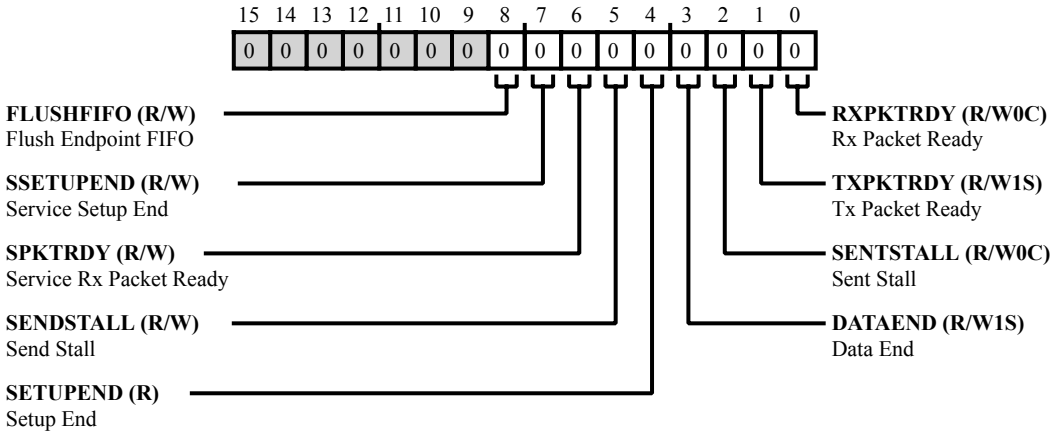

Figure 22-40: USB_EP0I_CSR[N]_P Register Diagram

Table 22-18: USB_EP0I_CSR[N]_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EP0I_CSR[N]_P.FLUSHFIFO bit directs (in peripheral mode) the USB controller to flush data from the endpoint 0 FIFO and clear the USB_EP0I_CSR[N]_P.TXPKTRDY and USB_EP0I_CSR[N]_P.RXPKTRDY bits. The USB_EP0I_CSR[N]_P.FLUSHFIFO bit should only be set if the USB_EP0I_CSR[N]_P.TXPKTRDY and USB_EP0I_CSR[N]_P.RXPKTRDY bits are set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
7 (R/W)	SSETUPEND	Service Setup End. The USB_EP0I_CSR[N]_P.SSETUPEND bit is set (in peripheral mode) by the processor core to clear the USB_EP0I_CSR[N]_P.SETUPEND. This bit is cleared automatically.	
		0	No Action
		1	Clear SETUPEND Bit

Table 22-18: USB_EP0I_CSR[N]_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	SPKTRDY	Service Rx Packet Ready. The USB_EP0I_CSR[N]_P.SPKTRDY bit is set (in peripheral mode) by the processor core to clear the USB_EP0I_CSR[N]_P.RXPKTRDY bit. This bit is cleared automatically.	
		0	No Action
		1	Clear RXPKTRDY Bit
5 (R/W)	SENDSTALL	Send Stall. The USB_EP0I_CSR[N]_P.SENDSTALL bit is set (in peripheral mode) by the processor core to terminate the current transaction. The STALL handshake is transmitted, then this bit automatically is cleared.	
		0	No Action
		1	Terminate Current Transaction
4 (R/NW)	SETUPEND	Setup End. The USB_EP0I_CSR[N]_P.SETUPEND bit indicates (in peripheral mode) when a control transaction ends before the USB_EP0I_CSR[N]_P.DATAEND bit is set. An interrupt is generated and the FIFO is flushed at this time. This bit is cleared when the processor core sets the USB_EP0I_CSR[N]_P.SSETUPEND bit.	
		0	No Status
		1	Setup Ended before DATAEND
3 (R/W1S)	DATAEND	Data End. The USB_EP0I_CSR[N]_P.DATAEND bit is set (in peripheral mode) by the processor core sets when the core: • Sets the USB_EP0I_CSR[N]_P.TXPKTRDY bit for the last data packet. • Clears the USB_EP0I_CSR[N]_P.RXPKTRDY bit after unloading the last data packet. • Sets the USB_EP0I_CSR[N]_P.TXPKTRDY bit for a zero length data packet. The USB_EP0I_CSR[N]_P.DATAEND bit is cleared automatically.	
		0	No Status
		1	Data End Condition
2 (R/W0C)	SENTSTALL	Sent Stall. The USB_EP0I_CSR[N]_P.SENTSTALL bit is set (in peripheral mode) when a STALL handshake is transmitted. The processor core should clear this bit.	
		0	No Status
		1	Transmitted STALL Handshake

Table 22-18: USB_EP0I_CSR[N]_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EP0I_CSR[N]_P.TXPKTRDY` bit should be set (in peripheral mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	
		1	Set this bit after loading a data packet into the FIFO
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EP0I_CSR[N]_P.RXPKTRDY` is set (in peripheral mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core clears this bit by setting the `USB_EP0I_CSR[N]_P.SPKTRDY` bit.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EP0 NAK Limit Register

The `USB_EP0I_NAKLIMIT[N]` register determines the number of frames/micro-frames after which endpoint 0 should timeout on receiving a stream of NAK responses for bulk endpoints.

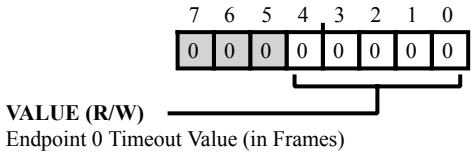

Figure 22-41: USB_EP0I_NAKLIMIT[N] Register Diagram

Table 22-19: USB_EP0I_NAKLIMIT[N] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	VALUE	Endpoint 0 Timeout Value (in Frames). The `USB_EP0I_NAKLIMIT[N].VALUE` bits hold the endpoint 0 timeout value (number of frames).

EP0 Connection Type Register

The USB_EP0I_TYPE[N] register selects the USB controller operating speed for endpoint 0 when acting as a host connected to devices through a hub.

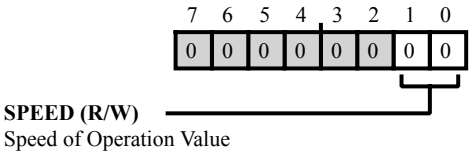

Figure 22-42: USB_EP0I_TYPE[N] Register Diagram

Table 22-20: USB_EP0I_TYPE[N] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	SPEED	Speed of Operation Value. The USB_EP0I_TYPE[N].SPEED bits select the USB controller operating speed for endpoint 0 when acting as a host connected to devices through a hub. In these instances. the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When not connected to devices through a hub, program this field with 00.	
		0	Same Speed as Processor Core
		1	High Speed
		2	Full Speed
		3	Low Speed

EP0 Configuration Information Register

The USB_EP0_CFGDATA[n] register describes the USB controller hardware configuration. This register only exists for endpoint 0.

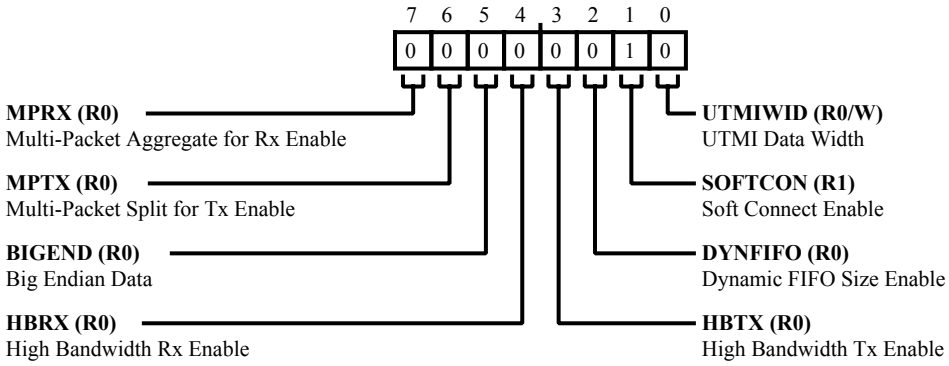

Figure 22-43: USB_EP0_CFGDATA[n] Register Diagram

Table 22-21: USB_EP0_CFGDATA[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R0/NW)	MPRX	Multi-Packet Aggregate for Rx Enable. The USB_EP0_CFGDATA[n].MPRX bit indicates whether the USB controller aggregates receive packets into bulk packets before the processor core reads the data.	
		0	No Aggregate Rx Bulk Packets
		1	Aggregate Rx Bulk Packets
6 (R0/NW)	MPTX	Multi-Packet Split for Tx Enable. The USB_EP0_CFGDATA[n].MPTX bit indicates whether the USB controller permits transmit of large packets through writing to bulk endpoints. The USB controller splits the transmit data into packets, which are appropriately sized for transmit.	
		0	No Split Tx Bulk Packets
		1	Split Tx Bulk Packets
5 (R0/NW)	BIGEND	Big Endian Data. The USB_EP0_CFGDATA[n].BIGEND bit indicates whether the USB controller uses big endian configuration or little endian configuration.	
		0	Little Endian Configuration
		1	Big Endian Configuration

Table 22-21: USB_EP0_CFGDATA[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R0/NW)	HBRX	High Bandwidth Rx Enable. The `USB_EP0_CFGDATA[n].HBRX` bit indicates whether the USB controller supports high-bandwidth receive ISO endpoint support.	
		0	No High Bandwidth Rx
		1	High Bandwidth Rx
3 (R0/NW)	HBTX	High Bandwidth Tx Enable. The `USB_EP0_CFGDATA[n].HBTX` bit indicates whether the USB controller supports high bandwidth transmit ISO endpoint support.	
		0	No High Bandwidth Tx
		1	High Bandwidth Tx
2 (R0/NW)	DYNFIFO	Dynamic FIFO Size Enable. The `USB_EP0_CFGDATA[n].DYNFIFO` bit indicates whether the USB controller uses dynamic FIFO size support (on products supporting this feature), enabling the dynamic FIFO registers. These registers are accessed using the configuration set in the endpoints indexed FIFO size and FIFO address registers, except for endpoint 0.	
		0	No Dynamic FIFO Size
		1	Dynamic FIFO Size
1 (R1/NW)	SOFTCON	Soft Connect Enable. The `USB_EP0_CFGDATA[n].SOFTCON` bit indicates whether the USB controller uses soft connect.	
		0	No Soft Connect
		1	Soft Connect
0 (R0/W)	UTMIWID	UTMI Data Width. The `USB_EP0_CFGDATA[n].UTMIWID` bit indicates whether the USB controller uses an 8-bit or 16-bit UTMI data width.	
		0	8-bit UTMI Data Width
		1	16-bit UTMI Data Width

EP0 Number of Received Bytes Register

The USB_EP0_CNT[n] register indicates the number of received data bytes in the endpoint 0 FIFO. The value returned changes as the contents of the FIFO change and is only valid while the USB_EP0_CSR[n]_H.RXPKTRDY bit or USB_EP0_CSR[n]_P.RXPKTRDY bit is set.

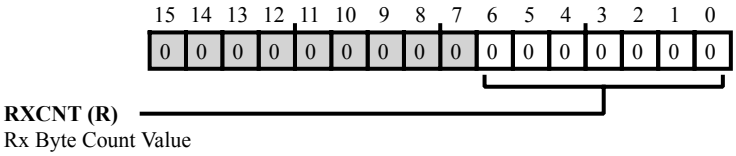

Figure 22-44: USB_EP0_CNT[n] Register Diagram

Table 22-22: USB_EP0_CNT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/NW)	RXCNT	Rx Byte Count Value. The USB_EP0_CNT[n].RXCNT bits holds the number of data bytes currently inline ready to be read from the Rx FIFO. The value returned changes as the FIFO is unloaded and is only valid while USB_EP0_CSR[n]_H.RXPKTRDY bit or USB_EP0_CSR[n]_P.RXPKTRDY bit is set.

EP0 Configuration and Status (Host) Register

The USB_EP0_CSR[n]_H register provides control and status bits for endpoint 0 in host mode. Note that some bits may be set to clear automatically.

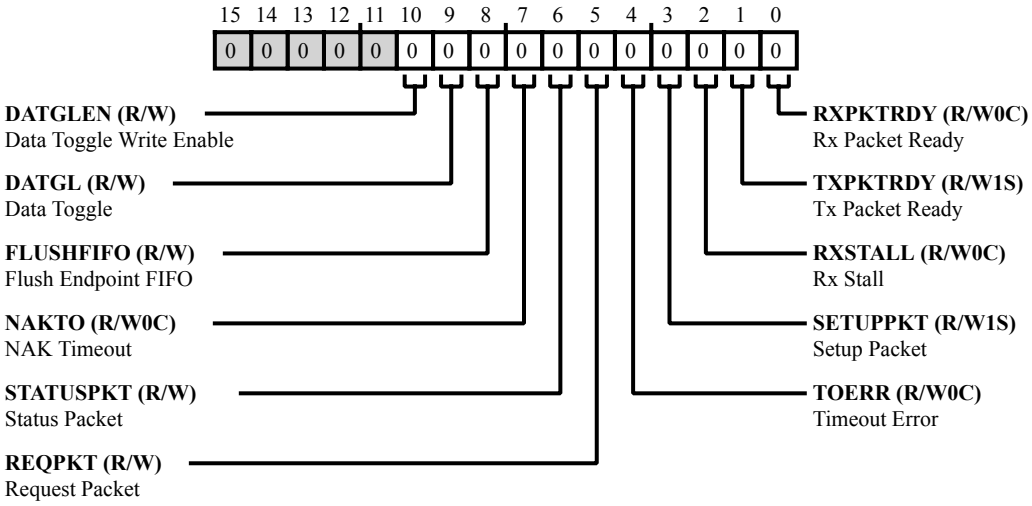

Figure 22-45: USB_EP0_CSR[n]_H Register Diagram

Table 22-23: USB_EP0_CSR[n]_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W)	DATGLEN	Data Toggle Write Enable. The USB_EP0_CSR[n]_H.DATGLEN bit enables (in host mode) the USB controller to write the current state of the endpoint 0 USB_EP0_CSR[n]_H.DATGL bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL
9 (R/W)	DATGL	Data Toggle. The USB_EP0_CSR[n]_H.DATGL bit indicates (in host mode) the current state of the endpoint 0 data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is Set
		1	DATA1 is Set

Table 22-23: USB_EP0_CSR[n]_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EP0_CSR[n]_H.FLUSHFIFO` bit directs (in host mode) the USB controller to flush data from the endpoint 0 FIFO and clear the `USB_EP0_CSR[n]_H.TXPKTRDY` and `USB_EP0_CSR[n]_H.RXPKTRDY` bits. The `USB_EP0_CSR[n]_H.FLUSHFIFO` bit should only be set if the `USB_EP0_CSR[n]_H.TXPKTRDY` and `USB_EP0_CSR[n]_H.RXPKTRDY` bits are set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
7 (R/W0C)	NAKTO	NAK Timeout. The `USB_EP0_CSR[n]_H.NAKTO` bit indicates (in host mode) when endpoint 0 is halted following the receipt of NAK responses for longer than the time set by the `USB_EP0_NAKLIMIT[n]` register. The processor core should clear this bit to allow the endpoint to continue.	
		0	No Status
		1	Endpoint Halted (NAK Timeout)
6 (R/W)	STATUSPKT	Status Packet. The `USB_EP0_CSR[n]_H.STATUSPKT` bit directs (in host mode) the USB controller to perform a status stage transaction. This bit is set at the same time as the `USB_EP0_CSR[n]_H.TXPKTRDY` `USB_EP0_CSR[n]_H.RXPKTRDY`. Setting this bit ensures that the data toggle is set to 1 so that a DATA1 packet is used for the status stage transaction.	
		0	No Request
		1	Request Status Transaction
5 (R/W)	REQPKT	Request Packet. The `USB_EP0_CSR[n]_H.REQPKT` bit directs (in host mode) the USB controller to request an IN transaction. This bit is cleared when `USB_EP0_CSR[n]_H.RXPKTRDY` is set.	
		0	No Request
		1	Send IN Tokens to Device
4 (R/W0C)	TOERR	Timeout Error. The `USB_EP0_CSR[n]_H.TOERR` bit indicates (in host mode) when three attempts have been made to perform a transaction with no response from the peripheral. The processor core should clear this bit. An interrupt is generated when this bit is set.	
		0	No Status
		1	Timeout Error

Table 22-23: USB_EP0_CSR[n]_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W1S)	SETUPPKT	Setup Packet. The `USB_EP0_CSR[n]_H.SETUPPKT` bit directs (in host mode) the USB controller to send a SETUP token instead of an OUT token for the transaction. This bit is set at the same time as the `USB_EP0_CSR[n]_H.TXPKTRDY` bit is set.	
		0	No Request
		1	Send SETUP token
2 (R/W0C)	RXSTALL	Rx Stall. The `USB_EP0_CSR[n]_H.RXSTALL` bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
1 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EP0_CSR[n]_H.TXPKTRDY` bit should be set (in host mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EP0_CSR[n]_H.RXPKTRDY` is set (in host mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EP0 Configuration and Status (Peripheral) Register

The USB_EP0_CSR[n]_P register provides control and status bits for endpoint 0 in peripheral mode. Note that some bits may be set to clear automatically.

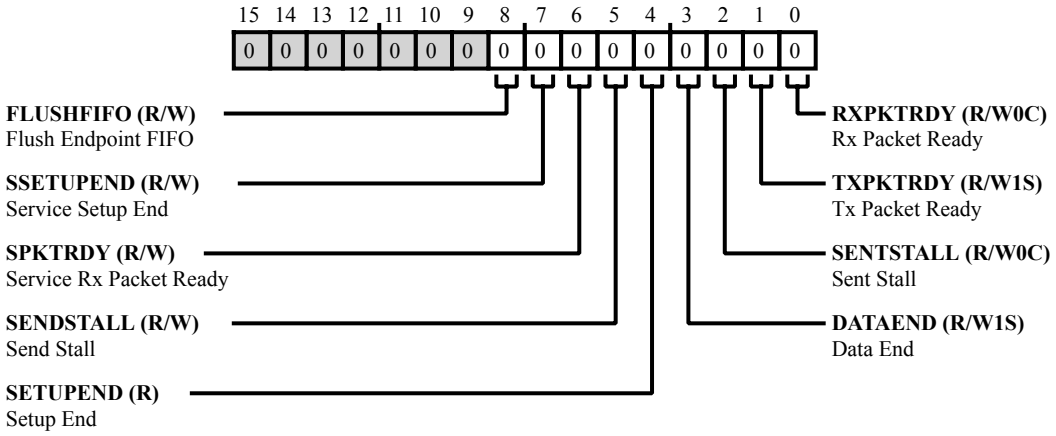

Figure 22-46: USB_EP0_CSR[n]_P Register Diagram

Table 22-24: USB_EP0_CSR[n]_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EP0_CSR[n]_P.FLUSHFIFO bit directs (in peripheral mode) the USB controller to flush data from the endpoint 0 FIFO and clear the USB_EP0_CSR[n]_P.TXPKTRDY and USB_EP0_CSR[n]_P.RXPKTRDY bits. The USB_EP0_CSR[n]_P.FLUSHFIFO bit should only be set if the USB_EP0_CSR[n]_P.TXPKTRDY and USB_EP0_CSR[n]_P.RXPKTRDY bits are set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
7 (R/W)	SSETUPEND	Service Setup End. The USB_EP0_CSR[n]_P.SSETUPEND bit is set (in peripheral mode) by the processor core to clear the USB_EP0_CSR[n]_P.SETUPEND. This bit is cleared automatically.	
		0	No Action
		1	Clear SETUPEND Bit

Table 22-24: USB_EP0_CSR[n]_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	SPKTRDY	Service Rx Packet Ready. The USB_EP0_CSR[n]_P.SPKTRDY bit is set (in peripheral mode) by the processor core to clear the USB_EP0_CSR[n]_P.RXPKTRDY bit. This bit is cleared automatically.	
		0	No Action
		1	Clear RXPKTRDY Bit
5 (R/W)	SENDSTALL	Send Stall. The USB_EP0_CSR[n]_P.SENDSTALL bit is set (in peripheral mode) by the processor core to terminate the current transaction. The STALL handshake is transmitted, then this bit automatically is cleared.	
		0	No Action
		1	Terminate Current Transaction
4 (R/NW)	SETUPEND	Setup End. The USB_EP0_CSR[n]_P.SETUPEND bit indicates (in peripheral mode) when a control transaction ends before the USB_EP0_CSR[n]_P.DATAEND bit is set. An interrupt is generated and the FIFO is flushed at this time. This bit is cleared when the processor core sets the USB_EP0_CSR[n]_P.SSETUPEND bit.	
		0	No Status
		1	Setup Ended before DATAEND
3 (R/W1S)	DATAEND	Data End. The USB_EP0_CSR[n]_P.DATAEND bit is set (in peripheral mode) by the processor core sets when the core: • Sets the USB_EP0_CSR[n]_P.TXPKTRDY bit for the last data packet. • Clears the USB_EP0_CSR[n]_P.RXPKTRDY bit after unloading the last data packet. • Sets the USB_EP0_CSR[n]_P.TXPKTRDY bit for a zero length data packet. The USB_EP0_CSR[n]_P.DATAEND bit is cleared automatically.	
		0	No Status
		1	Data End Condition
2 (R/W0C)	SENTSTALL	Sent Stall. The USB_EP0_CSR[n]_P.SENTSTALL bit is set (in peripheral mode) when a STALL handshake is transmitted. The processor core should clear this bit.	
		0	No Status
		1	Transmitted STALL Handshake

Table 22-24: USB_EP0_CSR[n]_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EP0_CSR[n]_P.TXPKTRDY` bit should be set (in peripheral mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	
		1	Set this bit after loading a data packet into the FIFO
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EP0_CSR[n]_P.RXPKTRDY` is set (in peripheral mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core clears this bit by setting the `USB_EP0_CSR[n]_P.SPKTRDY` bit.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EP0 NAK Limit Register

The USB_EP0_NAKLIMIT[n] register determines the number of frames/micro-frames after which endpoint 0 should timeout on receiving a stream of NAK responses for bulk endpoints.

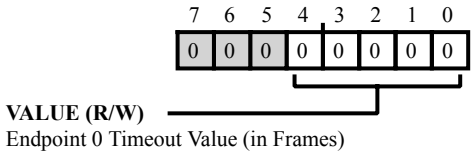

Figure 22-47: USB_EP0_NAKLIMIT[n] Register Diagram

Table 22-25: USB_EP0_NAKLIMIT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	VALUE	Endpoint 0 Timeout Value (in Frames). The USB_EP0_NAKLIMIT[n].VALUE bits hold the endpoint 0 timeout value (number of frames).

EP0 Connection Type Register

The `USB_EP0_TYPE[n]` register selects the USB controller operating speed for endpoint 0 when acting as a host connected to devices through a hub.

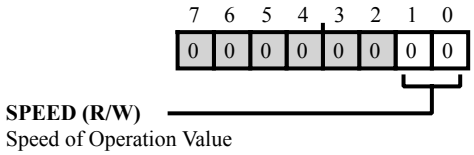

Figure 22-48: USB_EP0_TYPE[n] Register Diagram

Table 22-26: USB_EP0_TYPE[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	SPEED	Speed of Operation Value. The `USB_EP0_TYPE[n].SPEED` bits select the USB controller operating speed for endpoint 0 when acting as a host connected to devices through a hub. In these instances. the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When not connected to devices through a hub, program this field with 00.	
		0	Same Speed as Processor Core
		1	High Speed
		2	Full Speed
		3	Low Speed

Endpoint Information Register

The USB_EPINFO register allows read-back of the number of Tx and Rx endpoints available

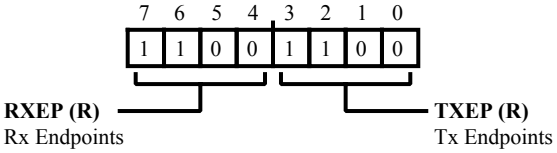

Figure 22-49: USB_EPINFO Register Diagram

Table 22-27: USB_EPINFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	RXEP	Rx Endpoints. The USB_EPINFO.RXEP bits indicate the number of receive endpoints. excluding EP0.
3:0 (R/NW)	TXEP	Tx Endpoints. The USB_EPINFO.TXEP bits indicate the number of transmit endpoints, excluding EP0.

EPn Number of Bytes Received Register

The USB_EPI[N]_RXCNT register indicates the number of received data bytes in the endpoint receive FIFO. The value returned changes as the contents of the FIFO change and is only valid while the USB_EP[n]_RXCSR_H.RXPKTRDY bit or USB_EP[n]_RXCSR_P.RXPKTRDY bit is set.

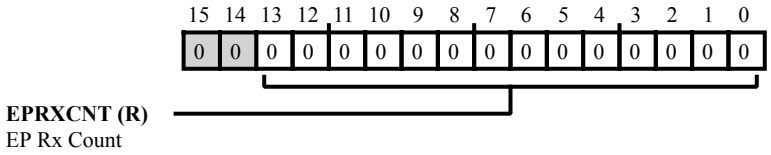

Figure 22-50: USB_EPI[N]_RXCNT Register Diagram

Table 22-28: USB_EPI[N]_RXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
13:0 (R/NW)	EPRXCNT	EP Rx Count. The USB_EPI[N]_RXCNT.EPRXCNT bits hold the number of data bytes ready to be read from the receive FIFO.

EPn Receive Configuration and Status (Host) Register

The USB_EPI[N]_RXCSR_H register provides (in host mode) control and status bits for transfers through the currently selected receive endpoint.

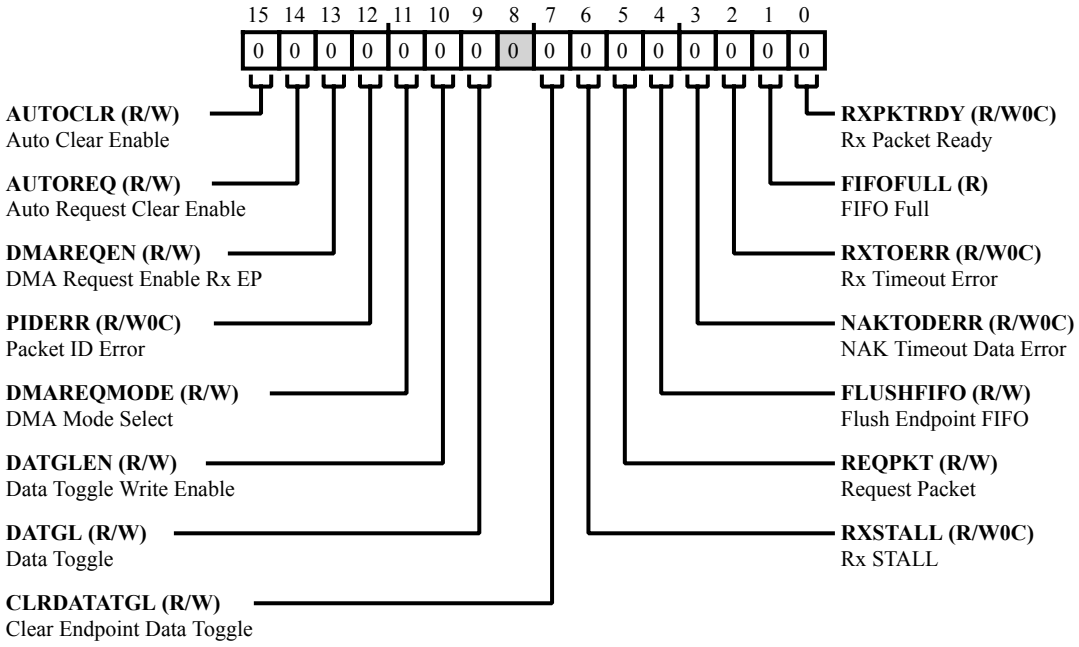

Figure 22-51: USB_EPI[N]_RXCSR_H Register Diagram

Table 22-29: USB_EPI[N]_RXCSR_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOCLR	Auto Clear Enable. The USB_EPI[N]_RXCSR_H.AUTOCLR bit directs (in host mode) the USB controller to automatically clear the USB_EPI[N]_RXCSR_H.RXPKTRDY bit when a packet of size USB_EP[n]_RXMAXP bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, the processor must clear USB_EPI[N]_RXCSR_H.RXPKTRDY manually. When using the DMA to unload the receive FIFO, data is read from the receive FIFO in four byte chunks, regardless of the USB_EP[n]_RXMAXP value. The USB controller auto clears the USB_EPI[N]_RXCSR_H.RXPKTRDY bit as follows. (In the following: Remainder=(RxMaxP/4), and PktSz-Clearing-RxPktRdy=Actual-Bytes-Read-Packet-Sizes-That-Clear-RxPktRdy.) • Remainder=0, Bytes-Read=RxMaxP, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2, RxMaxP-3 • Remainder=3, Bytes Read=RxMaxP+1, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2 • Remainder=2, Bytes Read=RxMaxP+2, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1 • Remainder=1, Bytes Read=RxMaxP+3, PktSz-Clearing-RxPktRdy=RxMaxP For products supporting high-speed operation, the USB_EPI[N]_RXCSR_H.AUTOCLR bit should not be set for high-bandwidth isochronous endpoints.	
		0	Disable Auto Clear
		1	Enable Auto Clear
14 (R/W)	AUTOREQ	Auto Request Clear Enable. The USB_EPI[N]_RXCSR_H.AUTOREQ bit directs (in host mode) the USB controller to automatically clear the USB_EPI[N]_RXCSR_H.REQPKT bit when USB_EPI[N]_RXCSR_H.RXPKTRDY bit is cleared. This bit is automatically cleared when a short packet is received.	
		0	Disable Auto Request Clear
		1	Enable Auto Request Clear
13 (R/W)	DMAREQEN	DMA Request Enable Rx EP. The USB_EPI[N]_RXCSR_H.DMAREQEN bit enables (in host mode) DMA requests for this receive endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 22-29: USB_EPI[N]_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W0C)	PIDERR	Packet ID Error. The `USB_EPI[N]_RXCSR_H.PIDERR` bit indicates (in host mode) when a PID error occurs for isochronous transactions. This bit is ignored in host mode for bulk or interrupt transactions.	
		0	No Status
		1	PID Error
11 (R/W)	DMAREQMODE	DMA Mode Select. The `USB_EPI[N]_RXCSR_H.DMAREQMODE` bit selects (in host mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the `USB_EPI[N]_RXCSR_H.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
10 (R/W)	DATGLEN	Data Toggle Write Enable. The `USB_EPI[N]_RXCSR_H.DATGLEN` bit enables (in host mode) the USB controller to write the current state of the endpoint `USB_EPI[N]_RXCSR_H.DATGL` bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL
9 (R/W)	DATGL	Data Toggle. The `USB_EPI[N]_RXCSR_H.DATGL` bit indicates (in host mode) the current state of the endpoint data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is Set
		1	DATA1 is Set
7 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EPI[N]_RXCSR_H.CLRDATATGL` bit is set (in host mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0

Table 22-29: USB_EPI[N]_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W0C)	RXSTALL	Rx STALL. The USB_EPI[N]_RXCSR_H.RXSTALL bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
5 (R/W)	REQPKT	Request Packet. The USB_EPI[N]_RXCSR_H.REQPKT bit directs (in host mode) the USB controller to request an IN transaction. This bit is cleared when USB_EPI[N]_RXCSR_H.RXPKTRDY is set.	
		0	No Request
		1	Send IN Tokens to Device
4 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EPI[N]_RXCSR_H.FLUSHFIFO bit directs (in host mode) the USB controller to flush data from the endpoint FIFO and clear the USB_EPI[N]_RXCSR_H.RXPKTRDY bit. The USB_EPI[N]_RXCSR_H.FLUSHFIFO bit should only be set if the USB_EPI[N]_RXCSR_H.RXPKTRDY bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
3 (R/W0C)	NAKTODERR	NAK Timeout Data Error. The USB_EPI[N]_RXCSR_H.NAKTODERR bit indicates (in host mode for isochronous transfers) a NAK timeout data error when the USB_EPI[N]_RXCSR_H.RXPKTRDY bit is set and the data packet has a CRC or bit-stuff error. This bit is cleared when the USB_EPI[N]_RXCSR_H.RXPKTRDY bit is cleared. The USB_EPI[N]_RXCSR_H.NAKTODERR bit indicates (in host mod for bulk transfers) when a receive endpoint is halted following the receipt of NAK responses greater than the limit set in the USB_EP[n]_RXINTERVAL register. The processor should clear this bit to allow the endpoint to continue. If double packet buffering is enabled, the USB_EPI[N]_RXCSR_H.REQPKT bit should also be set in the same cycle as this bit is cleared.	
		0	No Status
		1	NAK Timeout Data Error

Table 22-29: USB_EPI[N]_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W0C)	RXTOERR	Rx Timeout Error. The `USB_EPI[N]_RXCSR_H.RXTOERR` bit indicates (in host mode) when three attempts have been made to receive a packet and no data packet has been received. The USB controller generates an interrupt for this condition. The processor should clear this bit. Note that `USB_EPI[N]_RXCSR_H.RXTOERR` is valid only when the endpoint is operating in bulk or interrupt mode.	
		0	No Status
		1	Rx Timeout Error
1 (R/NW)	FIFOFULL	FIFO Full. The `USB_EPI[N]_RXCSR_H.FIFOFULL` bit indicates (in host mode) when no more packets can be loaded into the receive FIFO.	
		0	No Status
		1	FIFO Full
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EPI[N]_RXCSR_H.RXPKTRDY` is set (in host mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EPn Receive Configuration and Status (Peripheral) Register

The USB_EPI[N]_RXCSR_P register provides (in peripheral mode) control and status bits for transfers through the currently selected receive endpoint.

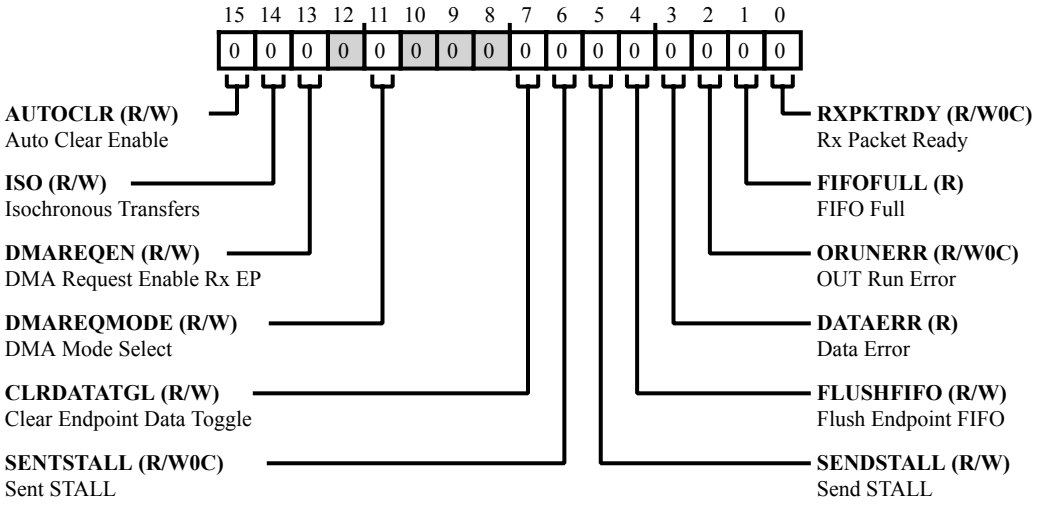

Figure 22-52: USB_EPI[N]_RXCSR_P Register Diagram

Table 22-30: USB_EPI[N]_RXCSR_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOCLR	Auto Clear Enable. The USB_EPI[N]_RXCSR_P.AUTOCLR bit directs (in peripheral mode) the USB controller to automatically clear the USB_EPI[N]_RXCSR_P.RXPKTRDY bit when a packet of size USB_EP[n]_RXMAXP bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, the processor must clear USB_EPI[N]_RXCSR_P.RXPKTRDY manually. When using the DMA to unload the receive FIFO, data is read from the receive FIFO in four byte chunks, regardless of the USB_EP[n]_RXMAXP value. The USB controller auto clears the USB_EPI[N]_RXCSR_P.RXPKTRDY bit as follows. (In the following: Remainder=(RxMaxP/4), and PktSz-Clearing-RxPktRdy=Actual-Bytes-Read-Packet-Sizes-That-Clear-RxPktRdy.) • Remainder=0, Bytes-Read=RxMaxP, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2, RxMaxP-3 • Remainder=3, Bytes Read=RxMaxP+1, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2 • Remainder=2, Bytes Read=RxMaxP+2, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1 • Remainder=1, Bytes Read=RxMaxP+3, PktSz-Clearing-RxPktRdy=RxMaxP For products supporting high-speed operation, the USB_EPI[N]_RXCSR_P.AUTOCLR bit should not be set for high-bandwidth isochronous endpoints.	
		0	Disable Auto Clear
		1	Enable Auto Clear
14 (R/W)	ISO	Isochronous Transfers. The USB_EPI[N]_RXCSR_P.ISO bit selects (in peripheral mode) between isochronous transfers and bulk/interrupt transfers.	
		0	This bit should be cleared for bulk or interrupt transfers.
		1	This bit should be set for isochronous transfers.
13 (R/W)	DMAREQEN	DMA Request Enable Rx EP. The USB_EPI[N]_RXCSR_P.DMAREQEN bit enables (in peripheral mode) DMA requests for this receive endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 22-30: USB_EPI[N]_RXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	DMAREQMODE	DMA Mode Select. The USB_EPI[N]_RXCSR_P.DMAREQMODE bit selects (in peripheral mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the USB_EPI[N]_RXCSR_P.DMAREQEN bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
7 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The USB_EPI[N]_RXCSR_P.CLRDATATGL bit is set (in peripheral mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
6 (R/W0C)	SENTSTALL	Sent STALL. The USB_EPI[N]_RXCSR_P.SENTSTALL bit indicates (in peripheral mode) when a STALL handshake is transmitted. The processor should clear this bit.	
		0	No Status
		1	STALL Handshake Transmitted
5 (R/W)	SENDSTALL	Send STALL. The USB_EPI[N]_RXCSR_P.SENDSTALL bit is set (in peripheral mode) by the processor to send a STALL handshake. The processor clears this bit to terminate the stall condition. This bit has no effect for isochronous transfers.	
		0	No Action
		1	Request STALL Handshake
4 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EPI[N]_RXCSR_P.FLUSHFIFO bit directs (in peripheral mode) the USB controller to flush data from the endpoint FIFO and clear the USB_EPI[N]_RXCSR_P.RXPKTRDY bit. The USB_EPI[N]_RXCSR_P.FLUSHFIFO bit should only be set if the USB_EPI[N]_RXCSR_P.RXPKTRDY bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO

Table 22-30: USB_EPI[N]_RXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/NW)	DATAERR	Data Error. The USB_EPI[N]_RXCSR_P.DATAERR bit indicates (in peripheral mode for isochronous transfers) when the USB_EPI[N]_RXCSR_P.RXPKTRDY bit is set and the data packet has a CRC or bit-stuff error. This bit is cleared when USB_EPI[N]_RXCSR_P.RXPKTRDY is cleared. The USB_EPI[N]_RXCSR_P.DATAERR bit is always zero for bulk endpoints in peripheral mode.	
		0	No Status
		1	Data Error
2 (R/W0C)	ORUNERR	OUT Run Error. The USB_EPI[N]_RXCSR_P.ORUNERR bit indicates (in peripheral mode for isochronous transfers) when an OUT packet cannot be loaded into the receive FIFO. The processor should clear this bit. The USB_EPI[N]_RXCSR_P.ORUNERR bit always returns zero in bulk mode.	
		0	No Status
		1	OUT Run Error
1 (R/NW)	FIFOFULL	FIFO Full. The USB_EPI[N]_RXCSR_P.FIFOFULL bit indicates (in peripheral mode) when no more packets can be loaded into the receive FIFO.	
		0	No Status
		1	FIFO Full
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The USB_EPI[N]_RXCSR_P.RXPKTRDY is set (in peripheral mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EPn Receive Polling Interval Register

The USB_EPI[N]_RXINTERVAL register defines the polling interval for the currently selected receive endpoint for interrupt, isochronous, and bulk transfers. There is a USB_EPI[N]_RXINTERVAL register for each configured receive endpoint, except endpoint 0. The transfer types relate to speedInterval value, and interval operation is as follows:

- Interrupt: Speed=Low Speed or Full Speed, USB_EPI[N]_RXINTERVAL=1-255, and Operation=Polling interval is m frames.
- Interrupt: Speed=High Speed, USB_EPI[N]_RXINTERVAL=1-16, and Operation=Polling interval is $2^{(m-1)}$ micro-frames.
- Isochronous: Speed=Full Speed or High Speed, USB_EPI[N]_RXINTERVAL=1-16, and Operation=Polling interval is $2^{(m-1)}$ frames or micro-frames.
- Bulk: Speed=Full Speed or High Speed, USB_EPI[N]_RXINTERVAL=2-16, and Operation=NAK Limit is $2^{(m-1)}$ frames or micro-frames.

Note that a USB_EPI[N]_RXINTERVAL value of 0 or 1 disables the NAK timeout function.

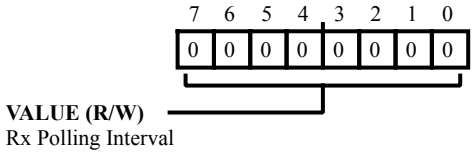

Figure 22-53: USB_EPI[N]_RXINTERVAL Register Diagram

Table 22-31: USB_EPI[N]_RXINTERVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Rx Polling Interval. The USB_EPI[N]_RXINTERVAL.VALUE bits define the polling interval value for interrupt and isochronous transfers and select the number of frames (or microframes, if the processor supports high-speed operation) after which the endpoint should timeout on receiving a stream of NAK responses for bulk and control endpoints. Note that the USB controller halts transfers to control endpoints if the host receives NAK responses for more frames than the limit set by this register.

EPn Receive Maximum Packet Length Register

The USB_EPI[N]_RXMAXP register defines the maximum amount of data that can be transferred through the selected receive endpoint in a single frame.

Note that a value greater than the maximum allowed of 1023 for full-speed USB operation produces unpredictable results. Also note that the total amount of data represented by the value written to this register must not exceed the receive FIFO size, and should not exceed half the FIFO size if double-buffering is required.

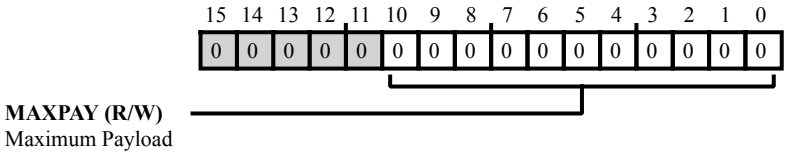

Figure 22-54: USB_EPI[N]_RXMAXP Register Diagram

Table 22-32: USB_EPI[N]_RXMAXP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	MAXPAY	Maximum Payload. The USB_EPI[N]_RXMAXP.MAXPAY bits select the maximum number of bytes that may be transferred per transaction. This field can be up to 1024 but is subject to constraints by the USB specification based on endpoint mode and speed. This field should not exceed the FIFO size for the endpoint, or half the FIFO size if double buffering is used. This value should match the wMaxPacketSize field of the standard endpoint descriptor (USB 2.0 spec, section 9). The USB_EPI[N]_RXMAXP.MAXPAY bits must be set to an even number of bytes for proper interrupt generation in DMA mode 1.

EPn Receive Type Register

The USB_EPI[N]_RXTYPE register selects the endpoint number and transaction protocol to use for the currently selected receive endpoint. There is a USB_EPI[N]_RXTYPE register for each receive endpoint. Note that this register is only used in host mode.

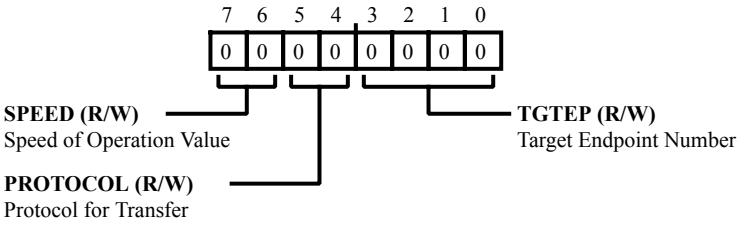

Figure 22-55: USB_EPI[N]_RXTYPE Register Diagram

Table 22-33: USB_EPI[N]_RXTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/W)	SPEED	Speed of Operation Value. The USB_EPI[N]_RXTYPE.SPEED bits select the USB controller operating speed for the endpoint when acting as a host connected to devices through a hub. In these instances. the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When not connected to devices through a hub, program this field with 00.	
		0	Same Speed as the Core
		1	High Speed
		2	Full Speed
		3	Low Speed
5:4 (R/W)	PROTOCOL	Protocol for Transfer. The USB_EPI[N]_RXTYPE.PROTOCOL bits select the transfer protocol for the endpoint.	
		0	Control
		1	Isochronous
		2	Bulk
		3	Interrupt
3:0 (R/W)	TGTEP	Target Endpoint Number. The USB_EPI[N]_RXTYPE.TGTEP bits select (for endpoints 1-11) the target endpoint. This value should be set to the endpoint number contained in the receive endpoint descriptor returned during device enumeration. Endpoint 0 always uses target endpoint number 0. (Enumeration values not shown are reserved.)	

Table 22-33: USB_EPI[N]_RXTYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	Endpoint 0
		1	Endpoint 1
		2	Endpoint 2
		3	Endpoint 3
		4	Endpoint 4
		5	Endpoint 5
		6	Endpoint 6
		7	Endpoint 7
		8	Endpoint 8
		9	Endpoint 9
		10	Endpoint 10
		11	Endpoint 11
		12	Endpoint 12
		13	Endpoint 13
		14	Endpoint 14
		15	Endpoint 15

EPn Transmit Configuration and Status (Host) Register

The USB_EPI[N]_TXCSR_H register provides (in host mode) control and status bits for transfers through the currently selected transmit endpoint.

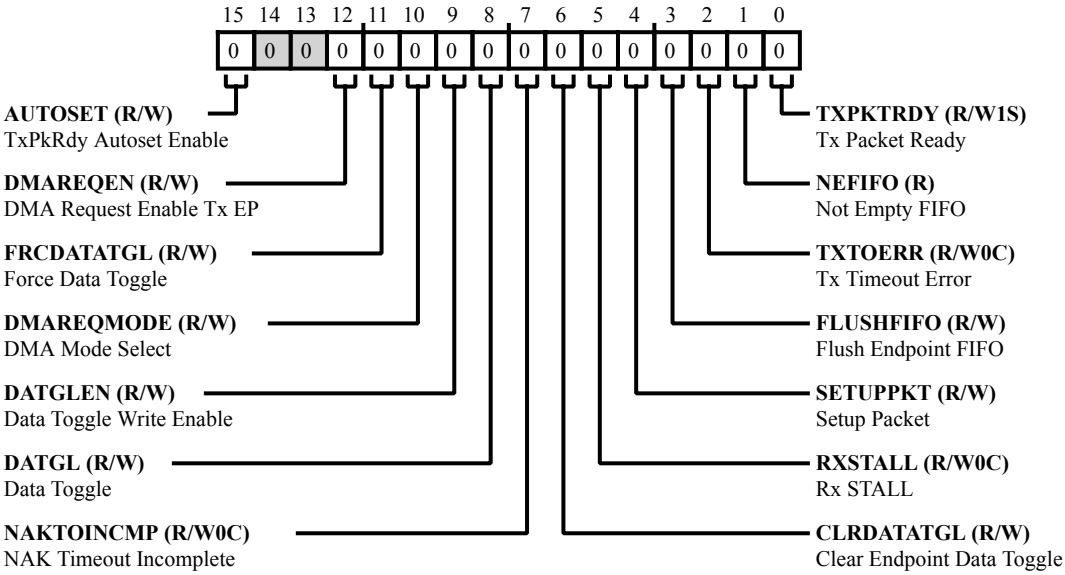

Figure 22-56: USB_EPI[N]_TXCSR_H Register Diagram

Table 22-34: USB_EPI[N]_TXCSR_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOSET	TxPkRdy Autoset Enable. The USB_EPI[N]_TXCSR_H.AUTOSET bit enables (in host mode) automatic setting of the USB_EPI[N]_TXCSR_H.TXPKTRDY bit when the maximum data packet size (USB_EP[n]_TXMAXP) is loaded into the transmit FIFO. The USB_EP[n]_TXMAXP value must be a word (4-byte) multiple. If a packet less than the maximum packet size is loaded, the USB_EPI[N]_TXCSR_H.TXPKTRDY bit needs to be set manually. For products supporting high-speed operation, this USB_EPI[N]_TXCSR_H.AUTOSET bit should not be set for high bandwidth endpoints (endpoints with USB_EP[n]_TXMAXP value greater than 1).	
		0	Disable Autoset
		1	Enable Autoset
12 (R/W)	DMAREQEN	DMA Request Enable Tx EP. The USB_EPI[N]_TXCSR_H.DMAREQEN bit enables (in host mode) DMA requests for this transmit endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 22-34: USB_EPI[N]_TXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	FRCDATATGL	Force Data Toggle. The `USB_EPI[N]_TXCSR_H.FRCDATATGL` bit forces (in host mode) the endpoint data toggle to switch and clears the data packet from the FIFO, regardless of whether an ACK was received. This feature can be used by interrupt transmit endpoints that are used to communicate rate feedback for isochronous endpoints.	
		0	No Action
		1	Toggle Endpoint Data
10 (R/W)	DMAREQMODE	DMA Mode Select. The `USB_EPI[N]_TXCSR_H.DMAREQMODE` bit selects (in host mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the `USB_EPI[N]_TXCSR_H.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
9 (R/W)	DATGLEN	Data Toggle Write Enable. The `USB_EPI[N]_TXCSR_H.DATGLEN` bit enables (in host mode) the USB controller to write the current state of the endpoint `USB_EPI[N]_TXCSR_H.DATGL` bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL
8 (R/W)	DATGL	Data Toggle. The `USB_EPI[N]_TXCSR_H.DATGL` bit indicates (in host mode) the current state of the endpoint data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is set
		1	DATA1 is set

Table 22-34: USB_EPI[N]_TXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W0C)	NAKTOINCMP	NAK Timeout Incomplete. The `USB_EPI[N]_TXCSR_H.NAKTOINCMP` bit indicates (for bulk endpoints in host mode) when the transmit endpoint is halted following the receipt of NAK responses for longer than the time set in the `USB_EP[n]_TXINTERVAL` register. The processor should clear this bit, allowing the endpoint to continue. For products supporting high-speed operation, for high-bandwidth isochronous endpoints in host mode, this bit indicates when no response is received from the device to which the packet is being sent.	
		0	No Status
		1	NAK Timeout Over Maximum
6 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EPI[N]_TXCSR_H.CLRDATATGL` bit is set (in host mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
5 (R/W0C)	RXSTALL	Rx STALL. The `USB_EPI[N]_TXCSR_H.RXSTALL` bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
4 (R/W)	SETUPPKT	Setup Packet. The `USB_EPI[N]_TXCSR_H.SETUPPKT` bit directs (in host mode) the USB controller to send a SETUP token instead of an OUT token for the transaction. This bit is set at the same time as the `USB_EPI[N]_TXCSR_H.TXPKTRDY` bit is set.	
		0	No Request
		1	Send SETUP Token
3 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EPI[N]_TXCSR_H.FLUSHFIFO` bit directs (in host mode) the USB controller to flush data from the endpoint FIFO and clear the `USB_EPI[N]_TXCSR_H.TXPKTRDY` bit. The `USB_EPI[N]_TXCSR_H.FLUSHFIFO` bit should only be set if the `USB_EPI[N]_TXCSR_H.TXPKTRDY` bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush endpoint FIFO

Table 22-34: USB_EPI[N]_TXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W0C)	TXTOERR	Tx Timeout Error. The `USB_EPI[N]_TXCSR_H.TXTOERR` bit indicates (in host mode) when three attempts have been made to send a packet and no handshake packet has been received. The USB controller generates an interrupt for this condition, clears the `USB_EPI[N]_TXCSR_H.TXPKTRDY` bit, and flushes the FIFO. The processor should clear this bit. Note that `USB_EPI[N]_TXCSR_H.TXTOERR` is valid only when the endpoint is operating in bulk or interrupt mode.	
		0	No Status
		1	Tx Timeout Error
1 (R/NW)	NEFIFO	Not Empty FIFO. The `USB_EPI[N]_TXCSR_H.NEFIFO` bit indicates (in host mode) when there is at least one packet in the transmit FIFO. This bit is cleared automatically when a data packet has been transmitted. If the endpoints transmit interrupt is enabled (in `USB_INTRTXE`), the USB controller generates an interrupt for this condition. Note that the `USB_EPI[N]_TXCSR_H.TXPKTRDY` bit is also automatically cleared prior to loading a second packet into a double-buffered FIFO.	
		0	FIFO Empty
		1	FIFO Not Empty
0 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EPI[N]_TXCSR_H.TXPKTRDY` bit should be set (in host mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO

EPn Transmit Configuration and Status (Peripheral) Register

The USB_EPI[N]_TXCSR_P register provides (in peripheral mode) control and status bits for transfers through the currently selected transmit endpoint.

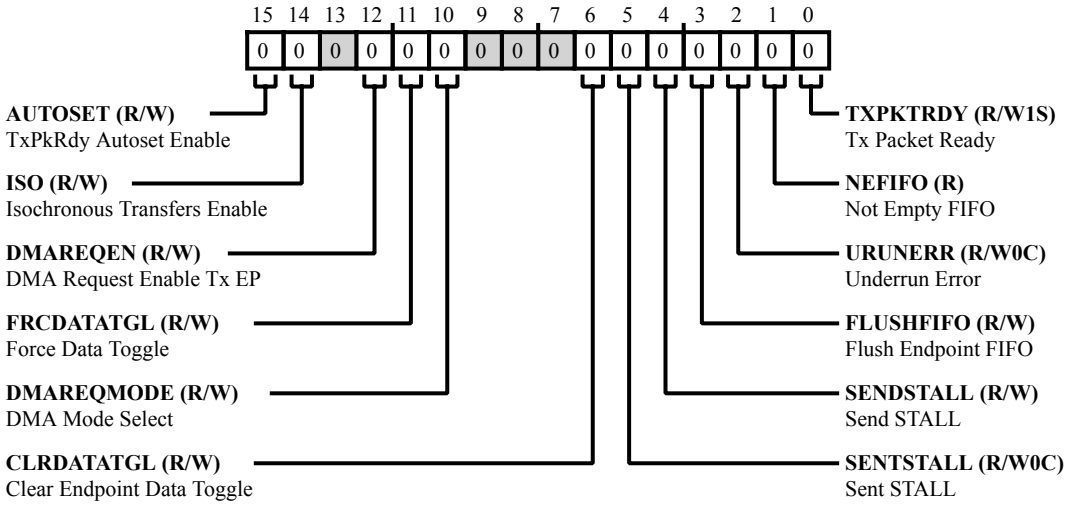

Figure 22-57: USB_EPI[N]_TXCSR_P Register Diagram

Table 22-35: USB_EPI[N]_TXCSR_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOSET	TxPkRdy Autoset Enable. The USB_EPI[N]_TXCSR_P.AUTOSET bit enables (in peripheral mode) automatic setting of the USB_EPI[N]_TXCSR_P.TXPKTRDY bit when the maximum data packet size (USB_EP[n]_TXMAXP) is loaded into the transmit FIFO. The USB_EP[n]_TXMAXP value must be a word (4-byte) multiple. If a packet less than the maximum packet size is loaded, the USB_EPI[N]_TXCSR_P.TXPKTRDY bit needs to be set manually. For products supporting high-speed operation, this USB_EPI[N]_TXCSR_P.AUTOSET bit should not be set for high bandwidth endpoints (endpoints with USB_EP[n]_TXMAXP value greater than 1).	
		0	Disable Autoset
		1	Enable Autoset
14 (R/W)	ISO	Isochronous Transfers Enable. The USB_EPI[N]_TXCSR_P.ISO bit enables (in peripheral mode) the transmit endpoint for isochronous transfers. This bit should be disabled for bulk or interrupt endpoints.	
		0	Disable Tx EP Isochronous Transfers
		1	Enable Tx EP Isochronous Transfers

Table 22-35: USB_EPI[N]_TXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	DMAREQEN	DMA Request Enable Tx EP. The `USB_EPI[N]_TXCSR_P.DMAREQEN` bit enables (in peripheral mode) DMA requests for this transmit endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request
11 (R/W)	FRCDATATGL	Force Data Toggle. The `USB_EPI[N]_TXCSR_P.FRCDATATGL` bit forces (in peripheral mode) the endpoint data toggle to switch and clears the data packet from the FIFO, regardless of whether an ACK was received. This feature can be used by interrupt transmit endpoints that are used to communicate rate feedback for isochronous endpoints.	
		0	No Action
		1	Toggle Endpoint Data
10 (R/W)	DMAREQMODE	DMA Mode Select. The `USB_EPI[N]_TXCSR_P.DMAREQMODE` bit selects (in peripheral mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the `USB_EPI[N]_TXCSR_P.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
6 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EPI[N]_TXCSR_P.CLRDATATGL` bit is set (in peripheral mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
5 (R/W0C)	SENTSTALL	Sent STALL. The `USB_EPI[N]_TXCSR_P.SENTSTALL` bit indicates (in peripheral mode) when the USB controller transmits a STALL handshake. When this condition occurs, the USB controller flushes the FIFO and clears the `USB_EPI[N]_TXCSR_P.TXPKTRDY` bit. The processor should clear this bit.	
		0	No Status
		1	STALL Handshake Transmitted

Table 22-35: USB_EPI[N]_TXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	SENDSTALL	Send STALL. The USB_EPI[N]_TXCSR_P.SENDSTALL bit (in peripheral mode) is set by the processor to issue a STALL handshake to an IN token. The processor clears this bit to terminate the stall condition. This bit has no effect for isochronous transfers.	
		0	No Request
		1	Request STALL Handshake Transmission
3 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EPI[N]_TXCSR_P.FLUSHFIFO bit directs (in peripheral mode) the USB controller to flush data from the endpoint FIFO and clear the USB_EPI[N]_TXCSR_P.TXPKTRDY bit. The USB_EPI[N]_TXCSR_P.FLUSHFIFO bit should only be set if the USB_EPI[N]_TXCSR_P.TXPKTRDY bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush endpoint FIFO
2 (R/W0C)	URUNERR	Underrun Error. The USB_EPI[N]_TXCSR_P.URUNERR bit indicates (in peripheral mode) when an IN token is received while the USB_EPI[N]_TXCSR_P.TXPKTRDY bit is not set. The processor should clear this bit.	
		0	No Status
		1	Underrun Error
1 (R/NW)	NEFIFO	Not Empty FIFO. The USB_EPI[N]_TXCSR_P.NEFIFO bit indicates (in peripheral mode) when there is at least one packet in the transmit FIFO. This bit is cleared automatically when a data packet has been transmitted. If the endpoints transmit interrupt is enabled (in USB_INTRTXE), the USB controller generates an interrupt for this condition. Note that the USB_EPI[N]_TXCSR_P.TXPKTRDY bit is also automatically cleared prior to loading a second packet into a double-buffered FIFO.	
		0	FIFO Empty
		1	FIFO Not Empty
0 (R/W1S)	TXPKTRDY	Tx Packet Ready. The USB_EPI[N]_TXCSR_P.TXPKTRDY bit should be set (in peripheral mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO

EPn Transmit Polling Interval Register

The USB_EPI[N]_TXINTERVAL register defines the polling interval for the currently selected transmit endpoint for interrupt, isochronous, and bulk transfers. There is a USB_EPI[N]_TXINTERVAL register for each configured transmit endpoint, except endpoint 0. The transfer types relate to speedInterval value, and interval operation is as follows:

- Interrupt: Speed=Low Speed or Full Speed, USB_EPI[N]_TXINTERVAL=1-255, and Operation=Polling interval is m frames.
- Interrupt: Speed=High Speed, USB_EPI[N]_TXINTERVAL=1-16, and Operation=Polling interval is $2^{(m-1)}$ micro-frames.
- Isochronous: Speed=Full Speed or High Speed, USB_EPI[N]_TXINTERVAL=1-16, and Operation=Polling interval is $2^{(m-1)}$ frames or micro-frames.
- Bulk: Speed=Full Speed or High Speed, USB_EPI[N]_TXINTERVAL=2-16, and Operation=NAK Limit is $2^{(m-1)}$ frames or micro-frames.

Note that a USB_EPI[N]_TXINTERVAL value of 0 or 1 disables the NAK timeout function.

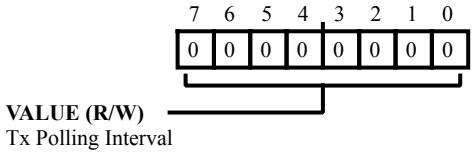

Figure 22-58: USB_EPI[N]_TXINTERVAL Register Diagram

Table 22-36: USB_EPI[N]_TXINTERVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Tx Polling Interval. The USB_EPI[N]_TXINTERVAL.VALUE bits define the polling interval value for interrupt and isochronous transfers and select the number of frames (or microframes, if the processor supports high-speed operation) after which the endpoint should timeout on receiving a stream of NAK responses for bulk and control endpoints. Note that the USB controller halts transfers to control endpoints if the host receives NAK responses for more frames than the limit set by this register.

EPn Transmit Maximum Packet Length Register

The USB_EPI[N]_TXMAXP register defines the maximum amount of data that can be transferred through the selected transmit endpoint in a single frame. When setting this value, you must consider the constraints placed by the USB specification on packet sizes for bulk, interrupt and isochronous transactions in full-speed operations. The USB_EPI[N]_TXMAXP register provides indexed access to the maximum packet length register for each Tx endpoint, except endpoint 0.

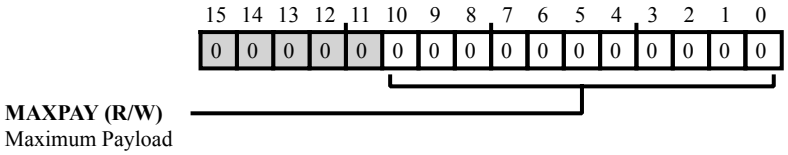

Figure 22-59: USB_EPI[N]_TXMAXP Register Diagram

Table 22-37: USB_EPI[N]_TXMAXP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	MAXPAY	Maximum Payload. The USB_EPI[N]_TXMAXP.MAXPAY bits select the maximum number of bytes that may be transferred per transaction. This field can be up to 1024 but is subject to constraints by the USB specification based on endpoint mode and speed. This field should not exceed the FIFO size for the endpoint, or half the FIFO size if double buffering is used. This value should match the wMaxPacketSize field of the standard endpoint descriptor (USB 2.0 spec, section 9). The USB_EPI[N]_TXMAXP.MAXPAY bits must be set to an even number of bytes for proper interrupt generation in DMA mode 1.

EPn Transmit Type Register

The USB_EPI[N]_TXTYPE register selects the endpoint number and transaction protocol to use for the currently selected transmit endpoint. There is a USB_EPI[N]_TXTYPE register for each transmit endpoint. Note that this register is only used in host mode.

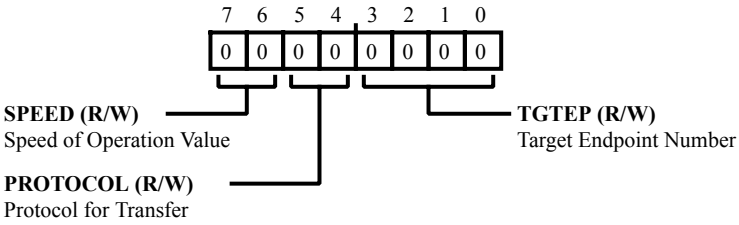

Figure 22-60: USB_EPI[N]_TXTYPE Register Diagram

Table 22-38: USB_EPI[N]_TXTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/W)	SPEED	Speed of Operation Value. The USB_EPI[N]_TXTYPE.SPEED bits select the USB controller operating speed for the endpoint when acting as a host connected to devices through a hub. In these instances. the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When not connected to devices through a hub, program this field with 00.	
		0	Same Speed as the Core
		1	High Speed
		2	Full Speed
		3	Low Speed
5:4 (R/W)	PROTOCOL	Protocol for Transfer. The USB_EPI[N]_TXTYPE.PROTOCOL bits select the transfer protocol for the endpoint.	
		0	Control
		1	Isochronous
		2	Bulk
		3	Interrupt
3:0 (R/W)	TGTEP	Target Endpoint Number. The USB_EPI[N]_TXTYPE.TGTEP bits select (for endpoints 1-11) the target endpoint. This value should be set to the endpoint number contained in the transmit endpoint descriptor returned during device enumeration. Endpoint 0 always uses target endpoint number 0. (Enumeration values not shown are reserved.)	

Table 22-38: USB_EPI[N]_TXTYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	Endpoint 0
		1	Endpoint 1
		2	Endpoint 2
		3	Endpoint 3
		4	Endpoint 4
		5	Endpoint 5
		6	Endpoint 6
		7	Endpoint 7
		8	Endpoint 8
		9	Endpoint 9
		10	Endpoint 10
		11	Endpoint 11
		12	Endpoint 12
		13	Endpoint 13
		14	Endpoint 14
		15	Endpoint 15

FIFO Size

When configured for fixed FIFO sizes, the USB_EPI_FIFOSZ register reports the size for the given endpoint. This register is only valid for endpoints greater than 0 (endpoint zero has a fixed size of 64 bytes). The USB_EPI_FIFOSZ register is not present if the USB controller is configured for dynamic FIFO sizing.

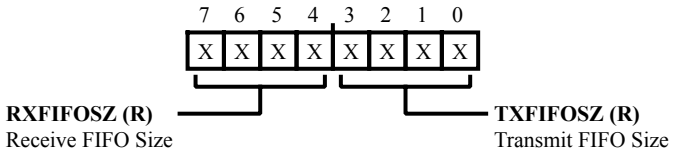

Figure 22-61: USB_EPI_FIFOSZ Register Diagram

Table 22-39: USB_EPI_FIFOSZ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	RXFIFOSZ	Receive FIFO Size. The USB_EPI_FIFOSZ.RXFIFOSZ bit encodes the log2 size of the receive FIFO. Values of 313 correspond to FIFO sizes 8-8192 (2^n). If an endpoint has not been configured, a value of 0 is returned. If the RX and TX endpoints share the same FIFO, a value of 0xF is returned.
3:0 (R/NW)	TXFIFOSZ	Transmit FIFO Size. The USB_EPI_FIFOSZ.TXFIFOSZ bit encodes the log2 size of the transmit FIFO. Values of 313 correspond to FIFO sizes 8-8192 (2^n). If an endpoint has not been configured, a value of 0 is returned.

FIFO Size

When configured for fixed FIFO sizes, the USB_EP[n]_FIFOSZ register reports the size for the given endpoint. This register is only valid for endpoints greater than 0 (endpoint zero has a fixed size of 64 bytes). The USB_EP[n]_FIFOSZ register is not present if the USB controller is configured for dynamic FIFO sizing.

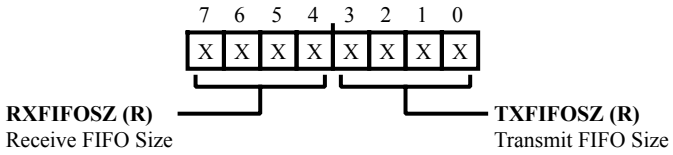

Figure 22-62: USB_EP[n]_FIFOSZ Register Diagram

Table 22-40: USB_EP[n]_FIFOSZ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	RXFIFOSZ	Receive FIFO Size. The USB_EP[n]_FIFOSZ.RXFIFOSZ bit encodes the log2 size of the receive FIFO. Values of 313 correspond to FIFO sizes 8-8192 (2^n). If an endpoint has not been configured, a value of 0 is returned. If the RX and TX endpoints share the same FIFO, a value of 0xF is returned.
3:0 (R/NW)	TXFIFOSZ	Transmit FIFO Size. The USB_EP[n]_FIFOSZ.TXFIFOSZ bit encodes the log2 size of the transmit FIFO. Values of 313 correspond to FIFO sizes 8-8192 (2^n). If an endpoint has not been configured, a value of 0 is returned.

EPn Number of Bytes Received Register

The USB_EP[n]_RXCNT register indicates the number of received data bytes in the endpoint receive FIFO. The value returned changes as the contents of the FIFO change and is only valid while the USB_EP[n]_RXCSR_H.RXPKTRDY bit or USB_EP[n]_RXCSR_P.RXPKTRDY bit is set.

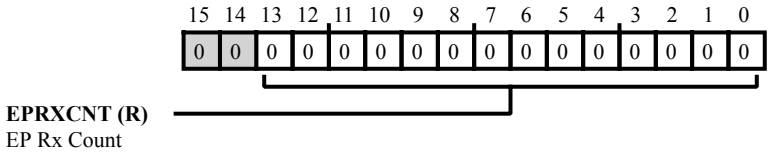

Figure 22-63: USB_EP[n]_RXCNT Register Diagram

Table 22-41: USB_EP[n]_RXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
13:0 (R/NW)	EPRXCNT	EP Rx Count. The USB_EP[n]_RXCNT.EPRXCNT bits hold the number of data bytes ready to be read from the receive FIFO.

EPn Receive Configuration and Status (Host) Register

The USB_EP[n]_RXCSR_H register provides (in host mode) control and status bits for transfers through the currently selected receive endpoint.

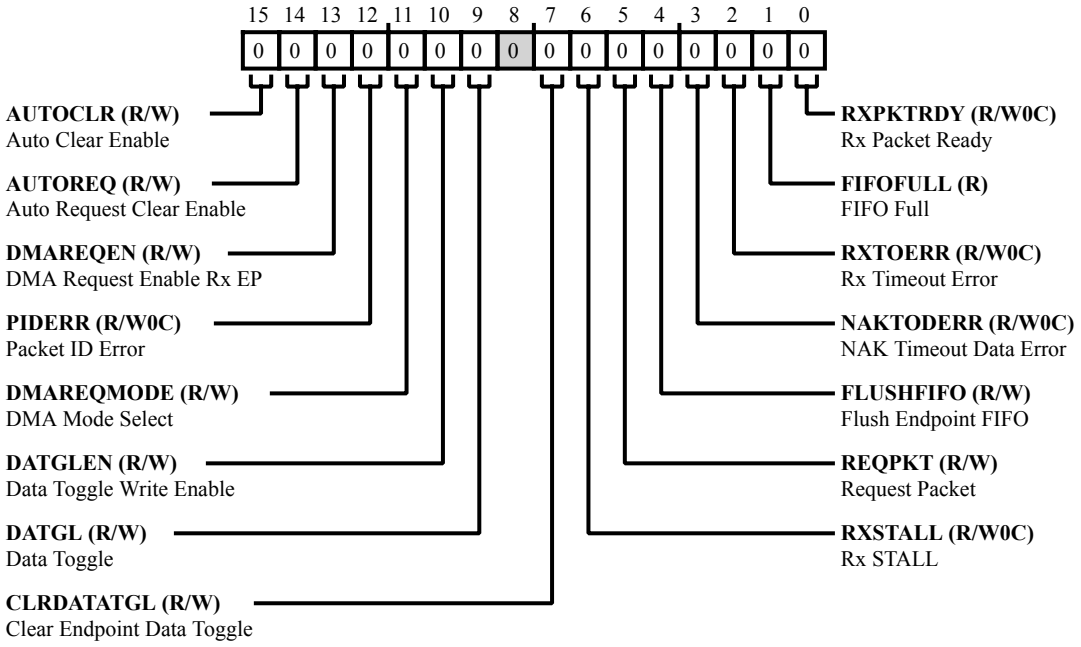

Figure 22-64: USB_EP[n]_RXCSR_H Register Diagram

Table 22-42: USB_EP[n]_RXCSR_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOCLR	Auto Clear Enable. The USB_EP[n]_RXCSR_H.AUTOCLR bit directs (in host mode) the USB controller to automatically clear the USB_EP[n]_RXCSR_H.RXPKTRDY bit when a packet of size USB_EP[n]_RXMAXP bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, the processor must clear USB_EP[n]_RXCSR_H.RXPKTRDY manually. When using the DMA to unload the receive FIFO, data is read from the receive FIFO in four byte chunks, regardless of the USB_EP[n]_RXMAXP value. The USB controller auto clears the USB_EP[n]_RXCSR_H.RXPKTRDY bit as follows. (In the following: Remainder=(RxMaxP/4), and PktSz-Clearing-RxPktRdy=Actual-Bytes-Read-Packet-Sizes-That-Clear-RxPktRdy.) • Remainder=0, Bytes-Read=RxMaxP, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2, RxMaxP-3 • Remainder=3, Bytes Read=RxMaxP+1, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2 • Remainder=2, Bytes Read=RxMaxP+2, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1 • Remainder=1, Bytes Read=RxMaxP+3, PktSz-Clearing-RxPktRdy=RxMaxP For products supporting high-speed operation, the USB_EP[n]_RXCSR_H.AUTOCLR bit should not be set for high-bandwidth isochronous endpoints.	
		0	Disable Auto Clear
		1	Enable Auto Clear
14 (R/W)	AUTOREQ	Auto Request Clear Enable. The USB_EP[n]_RXCSR_H.AUTOREQ bit directs (in host mode) the USB controller to automatically clear the USB_EP[n]_RXCSR_H.REQPKT bit when USB_EP[n]_RXCSR_H.RXPKTRDY bit is cleared. This bit is automatically cleared when a short packet is received.	
		0	Disable Auto Request Clear
		1	Enable Auto Request Clear
13 (R/W)	DMAREQEN	DMA Request Enable Rx EP. The USB_EP[n]_RXCSR_H.DMAREQEN bit enables (in host mode) DMA requests for this receive endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 22-42: USB_EP[n]_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W0C)	PIDERR	Packet ID Error. The `USB_EP[n]_RXCSR_H.PIDERR` bit indicates (in host mode) when a PID error occurs for isochronous transactions. This bit is ignored in host mode for bulk or interrupt transactions.	
		0	No Status
		1	PID Error
11 (R/W)	DMAREQMODE	DMA Mode Select. The `USB_EP[n]_RXCSR_H.DMAREQMODE` bit selects (in host mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the `USB_EP[n]_RXCSR_H.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
10 (R/W)	DATGLEN	Data Toggle Write Enable. The `USB_EP[n]_RXCSR_H.DATGLEN` bit enables (in host mode) the USB controller to write the current state of the endpoint `USB_EP[n]_RXCSR_H.DATGL` bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL
9 (R/W)	DATGL	Data Toggle. The `USB_EP[n]_RXCSR_H.DATGL` bit indicates (in host mode) the current state of the endpoint data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is Set
		1	DATA1 is Set
7 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EP[n]_RXCSR_H.CLRDATATGL` bit is set (in host mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0

Table 22-42: USB_EP[n]_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W0C)	RXSTALL	Rx STALL. The USB_EP[n]_RXCSR_H.RXSTALL bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
5 (R/W)	REQPKT	Request Packet. The USB_EP[n]_RXCSR_H.REQPKT bit directs (in host mode) the USB controller to request an IN transaction. This bit is cleared when USB_EP[n]_RXCSR_H.RXPKTRDY is set.	
		0	No Request
		1	Send IN Tokens to Device
4 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EP[n]_RXCSR_H.FLUSHFIFO bit directs (in host mode) the USB controller to flush data from the endpoint FIFO and clear the USB_EP[n]_RXCSR_H.RXPKTRDY bit. The USB_EP[n]_RXCSR_H.FLUSHFIFO bit should only be set if the USB_EP[n]_RXCSR_H.RXPKTRDY bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
3 (R/W0C)	NAKTODERR	NAK Timeout Data Error. The USB_EP[n]_RXCSR_H.NAKTODERR bit indicates (in host mode for isochronous transfers) a NAK timeout data error when the USB_EP[n]_RXCSR_H.RXPKTRDY bit is set and the data packet has a CRC or bit-stuff error. This bit is cleared when the USB_EP[n]_RXCSR_H.RXPKTRDY bit is cleared. The USB_EP[n]_RXCSR_H.NAKTODERR bit indicates (in host mod for bulk transfers) when a receive endpoint is halted following the receipt of NAK responses greater than the limit set in the USB_EP[n]_RXINTERVAL register. The processor should clear this bit to allow the endpoint to continue. If double packet buffering is enabled, the USB_EP[n]_RXCSR_H.REQPKT bit should also be set in the same cycle as this bit is cleared.	
		0	No Status
		1	NAK Timeout Data Error

Table 22-42: USB_EP[n]_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W0C)	RXTOERR	Rx Timeout Error. The `USB_EP[n]_RXCSR_H.RXTOERR` bit indicates (in host mode) when three attempts have been made to receive a packet and no data packet has been received. The USB controller generates an interrupt for this condition. The processor should clear this bit. Note that `USB_EP[n]_RXCSR_H.RXTOERR` is valid only when the endpoint is operating in bulk or interrupt mode.	
		0	No Status
		1	Rx Timeout Error
1 (R/NW)	FIFOFULL	FIFO Full. The `USB_EP[n]_RXCSR_H.FIFOFULL` bit indicates (in host mode) when no more packets can be loaded into the receive FIFO.	
		0	No Status
		1	FIFO Full
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EP[n]_RXCSR_H.RXPKTRDY` is set (in host mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EPn Receive Configuration and Status (Peripheral) Register

The `USB_EP[n]_RXCSR_P` register provides (in peripheral mode) control and status bits for transfers through the currently selected receive endpoint.

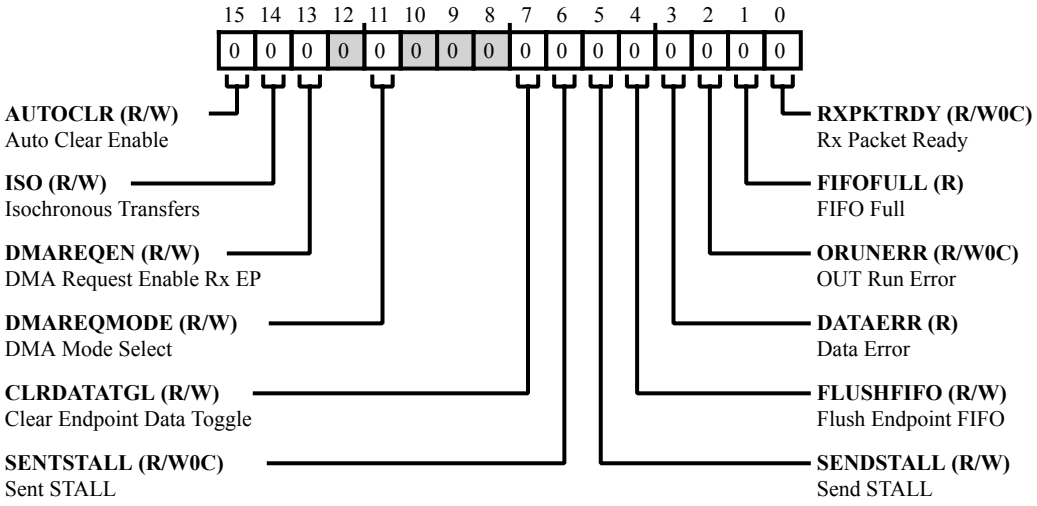

Figure 22-65: USB_EP[n]_RXCSR_P Register Diagram

Table 22-43: USB_EP[n]_RXCSR_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOCLR	Auto Clear Enable. The USB_EP[n]_RXCSR_P.AUTOCLR bit directs (in peripheral mode) the USB controller to automatically clear the USB_EP[n]_RXCSR_P.RXPKTRDY bit when a packet of size USB_EP[n]_RXMAXP bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, the processor must clear USB_EP[n]_RXCSR_P.RXPKTRDY manually. When using the DMA to unload the receive FIFO, data is read from the receive FIFO in four byte chunks, regardless of the USB_EP[n]_RXMAXP value. The USB controller auto clears the USB_EP[n]_RXCSR_P.RXPKTRDY bit as follows. (In the following: Remainder=(RxMaxP/4), and PktSz-Clearing-RxPktRdy=Actual-Bytes-Read-Packet-Sizes-That-Clear-RxPktRdy.) • Remainder=0, Bytes-Read=RxMaxP, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2, RxMaxP-3 • Remainder=3, Bytes Read=RxMaxP+1, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2 • Remainder=2, Bytes Read=RxMaxP+2, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1 • Remainder=1, Bytes Read=RxMaxP+3, PktSz-Clearing-RxPktRdy=RxMaxP For products supporting high-speed operation, the USB_EP[n]_RXCSR_P.AUTOCLR bit should not be set for high-bandwidth isochronous endpoints.	
		0	Disable Auto Clear
		1	Enable Auto Clear
14 (R/W)	ISO	Isochronous Transfers. The USB_EP[n]_RXCSR_P.ISO bit selects (in peripheral mode) between isochronous transfers and bulk/interrupt transfers.	
		0	This bit should be cleared for bulk or interrupt transfers.
		1	This bit should be set for isochronous transfers.
13 (R/W)	DMAREQEN	DMA Request Enable Rx EP. The USB_EP[n]_RXCSR_P.DMAREQEN bit enables (in peripheral mode) DMA requests for this receive endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 22-43: USB_EP[n]_RXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	DMAREQMODE	DMA Mode Select. The USB_EP[n]_RXCSR_P.DMAREQMODE bit selects (in peripheral mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the USB_EP[n]_RXCSR_P.DMAREQEN bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
7 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The USB_EP[n]_RXCSR_P.CLRDATATGL bit is set (in peripheral mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
6 (R/W0C)	SENTSTALL	Sent STALL. The USB_EP[n]_RXCSR_P.SENTSTALL bit indicates (in peripheral mode) when a STALL handshake is transmitted. The processor should clear this bit.	
		0	No Status
		1	STALL Handshake Transmitted
5 (R/W)	SENDSTALL	Send STALL. The USB_EP[n]_RXCSR_P.SENDSTALL bit is set (in peripheral mode) by the processor to send a STALL handshake. The processor clears this bit to terminate the stall condition. This bit has no effect for isochronous transfers.	
		0	No Action
		1	Request STALL Handshake
4 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EP[n]_RXCSR_P.FLUSHFIFO bit directs (in peripheral mode) the USB controller to flush data from the endpoint FIFO and clear the USB_EP[n]_RXCSR_P.RXPKTRDY bit. The USB_EP[n]_RXCSR_P.FLUSHFIFO bit should only be set if the USB_EP[n]_RXCSR_P.RXPKTRDY bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO

Table 22-43: USB_EP[n]_RXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/NW)	DATAERR	Data Error. The USB_EP[n]_RXCSR_P.DATAERR bit indicates (in peripheral mode for isochronous transfers) when the USB_EP[n]_RXCSR_P.RXPKTRDY bit is set and the data packet has a CRC or bit-stuff error. This bit is cleared when USB_EP[n]_RXCSR_P.RXPKTRDY is cleared. The USB_EP[n]_RXCSR_P.DATAERR bit is always zero for bulk endpoints in peripheral mode.	
		0	No Status
		1	Data Error
2 (R/W0C)	ORUNERR	OUT Run Error. The USB_EP[n]_RXCSR_P.ORUNERR bit indicates (in peripheral mode for isochronous transfers) when an OUT packet cannot be loaded into the receive FIFO. The processor should clear this bit. The USB_EP[n]_RXCSR_P.ORUNERR bit always returns zero in bulk mode.	
		0	No Status
		1	OUT Run Error
1 (R/NW)	FIFOFULL	FIFO Full. The USB_EP[n]_RXCSR_P.FIFOFULL bit indicates (in peripheral mode) when no more packets can be loaded into the receive FIFO.	
		0	No Status
		1	FIFO Full
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The USB_EP[n]_RXCSR_P.RXPKTRDY is set (in peripheral mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EPn Receive Polling Interval Register

The USB_EP[n]_RXINTERVAL register defines the polling interval for the currently selected receive endpoint for interrupt, isochronous, and bulk transfers. There is a USB_EP[n]_RXINTERVAL register for each configured receive endpoint, except endpoint 0. The transfer types relate to speedInterval value, and interval operation is as follows:

- Interrupt: Speed=Low Speed or Full Speed, USB_EP[n]_RXINTERVAL=1-255, and Operation=Polling interval is m frames.
- Interrupt: Speed=High Speed, USB_EP[n]_RXINTERVAL=1-16, and Operation=Polling interval is $2^{(m-1)}$ micro-frames.
- Isochronous: Speed=Full Speed or High Speed, USB_EP[n]_RXINTERVAL=1-16, and Operation=Polling interval is $2^{(m-1)}$ frames or micro-frames.
- Bulk: Speed=Full Speed or High Speed, USB_EP[n]_RXINTERVAL=2-16, and Operation=NAK Limit is $2^{(m-1)}$ frames or micro-frames.

Note that a USB_EP[n]_RXINTERVAL value of 0 or 1 disables the NAK timeout function.

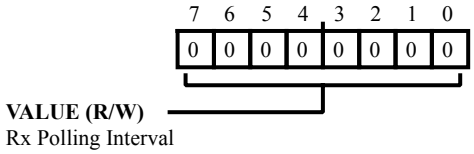

Figure 22-66: USB_EP[n]_RXINTERVAL Register Diagram

Table 22-44: USB_EP[n]_RXINTERVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Rx Polling Interval. The USB_EP[n]_RXINTERVAL.VALUE bits define the polling interval value for interrupt and isochronous transfers and select the number of frames (or microframes, if the processor supports high-speed operation) after which the endpoint should timeout on receiving a stream of NAK responses for bulk and control endpoints. Note that the USB controller halts transfers to control endpoints if the host receives NAK responses for more frames than the limit set by this register.

EPn Receive Maximum Packet Length Register

The USB_EP[n]_RXMAXP register defines the maximum amount of data that can be transferred through the selected receive endpoint in a single frame.

Note that a value greater than the maximum allowed of 1023 for full-speed USB operation produces unpredictable results. Also note that the total amount of data represented by the value written to this register must not exceed the receive FIFO size, and should not exceed half the FIFO size if double-buffering is required.

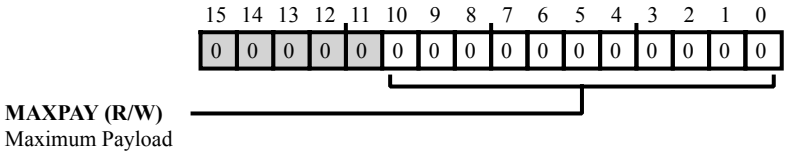

Figure 22-67: USB_EP[n]_RXMAXP Register Diagram

Table 22-45: USB_EP[n]_RXMAXP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	MAXPAY	Maximum Payload. The USB_EP[n]_RXMAXP.MAXPAY bits select the maximum number of bytes that may be transferred per transaction. This field can be up to 1024 but is subject to constraints by the USB specification based on endpoint mode and speed. This field should not exceed the FIFO size for the endpoint, or half the FIFO size if double buffering is used. This value should match the wMaxPacketSize field of the standard endpoint descriptor (USB 2.0 spec, section 9). The USB_EP[n]_RXMAXP.MAXPAY bits must be set to an even number of bytes for proper interrupt generation in DMA mode 1.

EPn Receive Type Register

The USB_EP[n]_RXTYPE register selects the endpoint number and transaction protocol to use for the currently selected receive endpoint. There is a USB_EP[n]_RXTYPE register for each receive endpoint. Note that this register is only used in host mode.

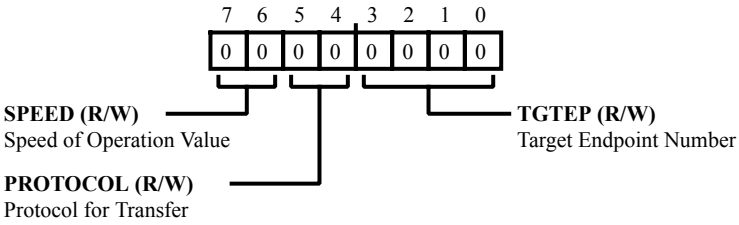

Figure 22-68: USB_EP[n]_RXTYPE Register Diagram

Table 22-46: USB_EP[n]_RXTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/W)	SPEED	Speed of Operation Value. The USB_EP[n]_RXTYPE.SPEED bits select the USB controller operating speed for the endpoint when acting as a host connected to devices through a hub. In these instances. the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When not connected to devices through a hub, program this field with 00.	
		0	Same Speed as the Core
		1	High Speed
		2	Full Speed
		3	Low Speed
5:4 (R/W)	PROTOCOL	Protocol for Transfer. The USB_EP[n]_RXTYPE.PROTOCOL bits select the transfer protocol for the endpoint.	
		0	Control
		1	Isochronous
		2	Bulk
		3	Interrupt
3:0 (R/W)	TGTEP	Target Endpoint Number. The USB_EP[n]_RXTYPE.TGTEP bits select (for endpoints 1-11) the target endpoint. This value should be set to the endpoint number contained in the receive endpoint descriptor returned during device enumeration. Endpoint 0 always uses target endpoint number 0. (Enumeration values not shown are reserved.)	

Table 22-46: USB_EP[n]_RXTYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	Endpoint 0
		1	Endpoint 1
		2	Endpoint 2
		3	Endpoint 3
		4	Endpoint 4
		5	Endpoint 5
		6	Endpoint 6
		7	Endpoint 7
		8	Endpoint 8
		9	Endpoint 9
		10	Endpoint 10
		11	Endpoint 11
		12	Endpoint 12
		13	Endpoint 13
		14	Endpoint 14
		15	Endpoint 15

EPn Transmit Configuration and Status (Host) Register

The USB_EP[n]_TXCSR_H register provides (in host mode) control and status bits for transfers through the currently selected transmit endpoint.

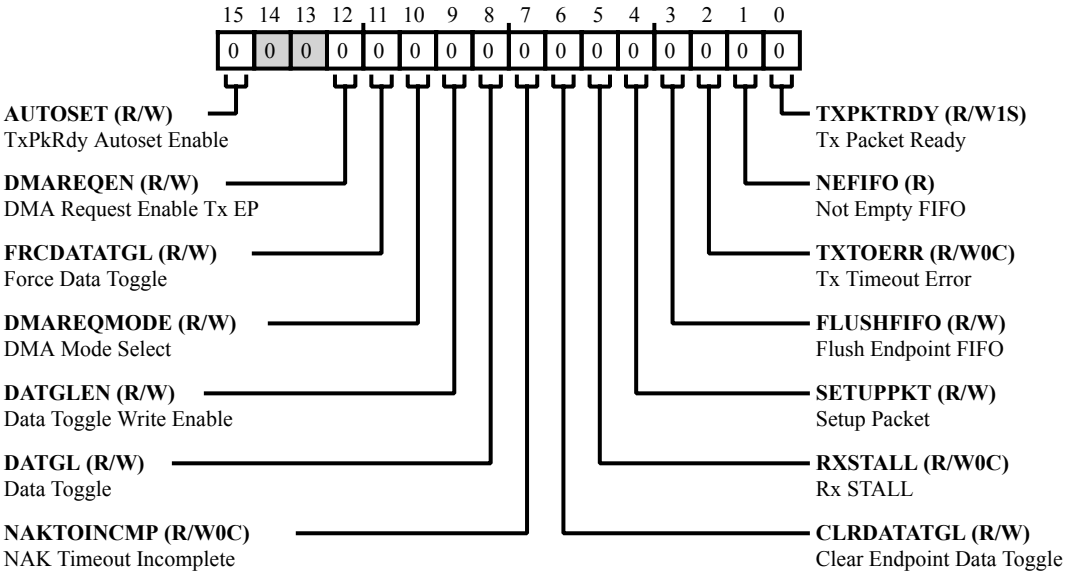

Figure 22-69: USB_EP[n]_TXCSR_H Register Diagram

Table 22-47: USB_EP[n]_TXCSR_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOSET	TxPkRdy Autoset Enable. The USB_EP[n]_TXCSR_H.AUTOSET bit enables (in host mode) automatic setting of the USB_EP[n]_TXCSR_H.TXPKTRDY bit when the maximum data packet size (USB_EP[n]_TXMAXP) is loaded into the transmit FIFO. The USB_EP[n]_TXMAXP value must be a word (4-byte) multiple. If a packet less than the maximum packet size is loaded, the USB_EP[n]_TXCSR_H.TXPKTRDY bit needs to be set manually. For products supporting high-speed operation, this USB_EP[n]_TXCSR_H.AUTOSET bit should not be set for high bandwidth endpoints (endpoints with USB_EP[n]_TXMAXP value greater than 1).	
		0	Disable Autoset
		1	Enable Autoset
12 (R/W)	DMAREQEN	DMA Request Enable Tx EP. The USB_EP[n]_TXCSR_H.DMAREQEN bit enables (in host mode) DMA requests for this transmit endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 22-47: USB_EP[n]_TXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	FRCDATATGL	Force Data Toggle. The `USB_EP[n]_TXCSR_H.FRCDATATGL` bit forces (in host mode) the endpoint data toggle to switch and clears the data packet from the FIFO, regardless of whether an ACK was received. This feature can be used by interrupt transmit endpoints that are used to communicate rate feedback for isochronous endpoints.	
		0	No Action
		1	Toggle Endpoint Data
10 (R/W)	DMAREQMODE	DMA Mode Select. The `USB_EP[n]_TXCSR_H.DMAREQMODE` bit selects (in host mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the `USB_EP[n]_TXCSR_H.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
9 (R/W)	DATGLEN	Data Toggle Write Enable. The `USB_EP[n]_TXCSR_H.DATGLEN` bit enables (in host mode) the USB controller to write the current state of the endpoint `USB_EP[n]_TXCSR_H.DATGL` bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL
8 (R/W)	DATGL	Data Toggle. The `USB_EP[n]_TXCSR_H.DATGL` bit indicates (in host mode) the current state of the endpoint data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is set
		1	DATA1 is set

Table 22-47: USB_EP[n]_TXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W0C)	NAKTOINCMP	NAK Timeout Incomplete. The `USB_EP[n]_TXCSR_H.NAKTOINCMP` bit indicates (for bulk endpoints in host mode) when the transmit endpoint is halted following the receipt of NAK responses for longer than the time set in the `USB_EP[n]_TXINTERVAL` register. The processor should clear this bit, allowing the endpoint to continue. For products supporting high-speed operation, for high-bandwidth isochronous endpoints in host mode, this bit indicates when no response is received from the device to which the packet is being sent.	
		0	No Status
		1	NAK Timeout Over Maximum
6 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EP[n]_TXCSR_H.CLRDATATGL` bit is set (in host mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
5 (R/W0C)	RXSTALL	Rx STALL. The `USB_EP[n]_TXCSR_H.RXSTALL` bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
4 (R/W)	SETUPPKT	Setup Packet. The `USB_EP[n]_TXCSR_H.SETUPPKT` bit directs (in host mode) the USB controller to send a SETUP token instead of an OUT token for the transaction. This bit is set at the same time as the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit is set.	
		0	No Request
		1	Send SETUP Token
3 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EP[n]_TXCSR_H.FLUSHFIFO` bit directs (in host mode) the USB controller to flush data from the endpoint FIFO and clear the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit. The `USB_EP[n]_TXCSR_H.FLUSHFIFO` bit should only be set if the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush endpoint FIFO

Table 22-47: USB_EP[n]_TXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W0C)	TXTOERR	Tx Timeout Error. The `USB_EP[n]_TXCSR_H.TXTOERR` bit indicates (in host mode) when three attempts have been made to send a packet and no handshake packet has been received. The USB controller generates an interrupt for this condition, clears the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit, and flushes the FIFO. The processor should clear this bit. Note that `USB_EP[n]_TXCSR_H.TXTOERR` is valid only when the endpoint is operating in bulk or interrupt mode.	
		0	No Status
		1	Tx Timeout Error
1 (R/NW)	NEFIFO	Not Empty FIFO. The `USB_EP[n]_TXCSR_H.NEFIFO` bit indicates (in host mode) when there is at least one packet in the transmit FIFO. This bit is cleared automatically when a data packet has been transmitted. If the endpoints transmit interrupt is enabled (in `USB_INTRTXE`), the USB controller generates an interrupt for this condition. Note that the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit is also automatically cleared prior to loading a second packet into a double-buffered FIFO.	
		0	FIFO Empty
		1	FIFO Not Empty
0 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EP[n]_TXCSR_H.TXPKTRDY` bit should be set (in host mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO

EPn Transmit Configuration and Status (Peripheral) Register

The USB_EP[n]_TXCSR_P register provides (in peripheral mode) control and status bits for transfers through the currently selected transmit endpoint.

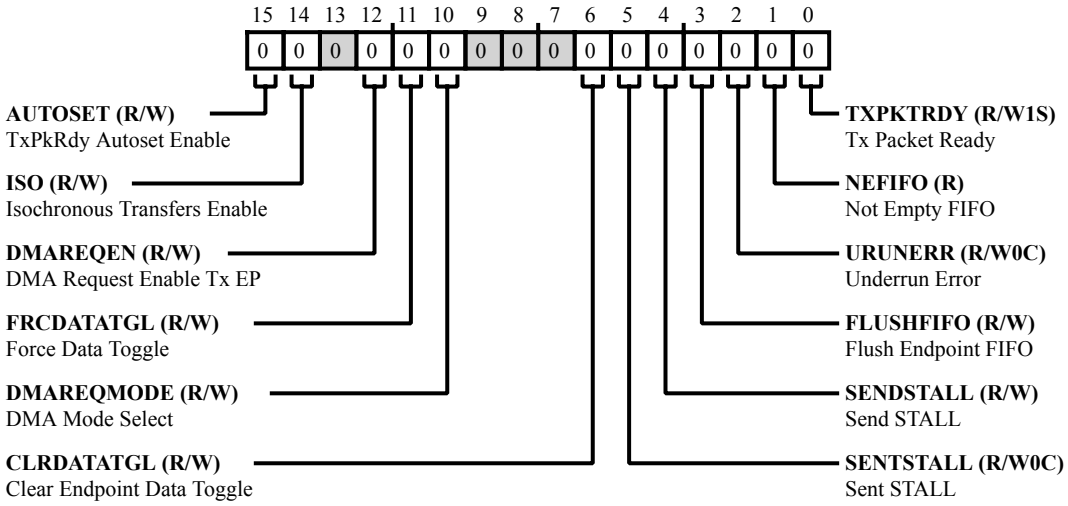

Figure 22-70: USB_EP[n]_TXCSR_P Register Diagram

Table 22-48: USB_EP[n]_TXCSR_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOSET	TxPkRdy Autoset Enable. The USB_EP[n]_TXCSR_P.AUTOSET bit enables (in peripheral mode) automatic setting of the USB_EP[n]_TXCSR_P.TXPKTRDY bit when the maximum data packet size (USB_EP[n]_TXMAXP) is loaded into the transmit FIFO. The USB_EP[n]_TXMAXP value must be a word (4-byte) multiple. If a packet less than the maximum packet size is loaded, the USB_EP[n]_TXCSR_P.TXPKTRDY bit needs to be set manually. For products supporting high-speed operation, this USB_EP[n]_TXCSR_P.AUTOSET bit should not be set for high bandwidth endpoints (endpoints with USB_EP[n]_TXMAXP value greater than 1).	
		0	Disable Autoset
		1	Enable Autoset
14 (R/W)	ISO	Isochronous Transfers Enable. The USB_EP[n]_TXCSR_P.ISO bit enables (in peripheral mode) the transmit endpoint for isochronous transfers. This bit should be disabled for bulk or interrupt endpoints.	
		0	Disable Tx EP Isochronous Transfers
		1	Enable Tx EP Isochronous Transfers

Table 22-48: USB_EP[n]_TXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	DMAREQEN	DMA Request Enable Tx EP. The `USB_EP[n]_TXCSR_P.DMAREQEN` bit enables (in peripheral mode) DMA requests for this transmit endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request
11 (R/W)	FRCDATATGL	Force Data Toggle. The `USB_EP[n]_TXCSR_P.FRCDATATGL` bit forces (in peripheral mode) the endpoint data toggle to switch and clears the data packet from the FIFO, regardless of whether an ACK was received. This feature can be used by interrupt transmit endpoints that are used to communicate rate feedback for isochronous endpoints.	
		0	No Action
		1	Toggle Endpoint Data
10 (R/W)	DMAREQMODE	DMA Mode Select. The `USB_EP[n]_TXCSR_P.DMAREQMODE` bit selects (in peripheral mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the `USB_EP[n]_TXCSR_P.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
6 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EP[n]_TXCSR_P.CLRDATATGL` bit is set (in peripheral mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
5 (R/W0C)	SENTSTALL	Sent STALL. The `USB_EP[n]_TXCSR_P.SENTSTALL` bit indicates (in peripheral mode) when the USB controller transmits a STALL handshake. When this condition occurs, the USB controller flushes the FIFO and clears the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit. The processor should clear this bit.	
		0	No Status
		1	STALL Handshake Transmitted

Table 22-48: USB_EP[n]_TXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	SENDSTALL	Send STALL. The USB_EP[n]_TXCSR_P.SENDSTALL bit (in peripheral mode) is set by the processor to issue a STALL handshake to an IN token. The processor clears this bit to terminate the stall condition. This bit has no effect for isochronous transfers.	
		0	No Request
		1	Request STALL Handshake Transmission
3 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EP[n]_TXCSR_P.FLUSHFIFO bit directs (in peripheral mode) the USB controller to flush data from the endpoint FIFO and clear the USB_EP[n]_TXCSR_P.TXPKTRDY bit. The USB_EP[n]_TXCSR_P.FLUSHFIFO bit should only be set if the USB_EP[n]_TXCSR_P.TXPKTRDY bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush endpoint FIFO
2 (R/W0C)	URUNERR	Underrun Error. The USB_EP[n]_TXCSR_P.URUNERR bit indicates (in peripheral mode) when an IN token is received while the USB_EP[n]_TXCSR_P.TXPKTRDY bit is not set. The processor should clear this bit.	
		0	No Status
		1	Underrun Error
1 (R/NW)	NEFIFO	Not Empty FIFO. The USB_EP[n]_TXCSR_P.NEFIFO bit indicates (in peripheral mode) when there is at least one packet in the transmit FIFO. This bit is cleared automatically when a data packet has been transmitted. If the endpoints transmit interrupt is enabled (in USB_INTRTXE), the USB controller generates an interrupt for this condition. Note that the USB_EP[n]_TXCSR_P.TXPKTRDY bit is also automatically cleared prior to loading a second packet into a double-buffered FIFO.	
		0	FIFO Empty
		1	FIFO Not Empty
0 (R/W1S)	TXPKTRDY	Tx Packet Ready. The USB_EP[n]_TXCSR_P.TXPKTRDY bit should be set (in peripheral mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO

EPn Transmit Polling Interval Register

The USB_EP[n]_TXINTERVAL register defines the polling interval for the currently selected transmit endpoint for interrupt, isochronous, and bulk transfers. There is a USB_EP[n]_TXINTERVAL register for each configured transmit endpoint, except endpoint 0. The transfer types relate to speedInterval value, and interval operation is as follows:

- Interrupt: Speed=Low Speed or Full Speed, USB_EP[n]_TXINTERVAL=1-255, and Operation=Polling interval is m frames.
- Interrupt: Speed=High Speed, USB_EP[n]_TXINTERVAL=1-16, and Operation=Polling interval is $2^{(m-1)}$ micro-frames.
- Isochronous: Speed=Full Speed or High Speed, USB_EP[n]_TXINTERVAL=1-16, and Operation=Polling interval is $2^{(m-1)}$ frames or micro-frames.
- Bulk: Speed=Full Speed or High Speed, USB_EP[n]_TXINTERVAL=2-16, and Operation=NAK Limit is $2^{(m-1)}$ frames or micro-frames.

Note that a USB_EP[n]_TXINTERVAL value of 0 or 1 disables the NAK timeout function.

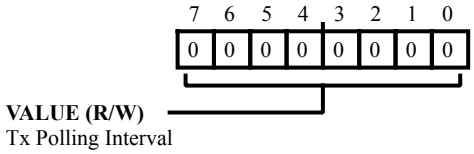

Figure 22-71: USB_EP[n]_TXINTERVAL Register Diagram

Table 22-49: USB_EP[n]_TXINTERVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Tx Polling Interval. The USB_EP[n]_TXINTERVAL.VALUE bits define the polling interval value for interrupt and isochronous transfers and select the number of frames (or microframes, if the processor supports high-speed operation) after which the endpoint should timeout on receiving a stream of NAK responses for bulk and control endpoints. Note that the USB controller halts transfers to control endpoints if the host receives NAK responses for more frames than the limit set by this register.

EPn Transmit Maximum Packet Length Register

The USB_EP[n]_TXMAXP register defines the maximum amount of data that can be transferred through the selected transmit endpoint in a single frame. When setting this value, you must consider the constraints placed by the USB specification on packet sizes for bulk, interrupt and isochronous transactions in full-speed operations. The USB_EP[n]_TXMAXP register provides indexed access to the maximum packet length register for each Tx endpoint, except endpoint 0.

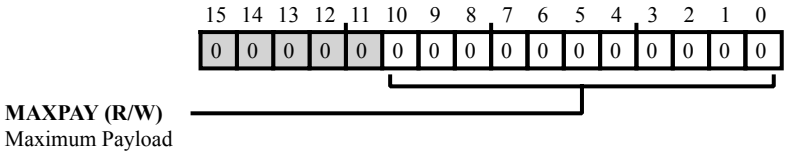

Figure 22-72: USB_EP[n]_TXMAXP Register Diagram

Table 22-50: USB_EP[n]_TXMAXP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	MAXPAY	Maximum Payload. The USB_EP[n]_TXMAXP.MAXPAY bits select the maximum number of bytes that may be transferred per transaction. This field can be up to 1024 but is subject to constraints by the USB specification based on endpoint mode and speed. This field should not exceed the FIFO size for the endpoint, or half the FIFO size if double buffering is used. This value should match the wMaxPacketSize field of the standard endpoint descriptor (USB 2.0 spec, section 9). The USB_EP[n]_TXMAXP.MAXPAY bits must be set to an even number of bytes for proper interrupt generation in DMA mode 1.

EPn Transmit Type Register

The USB_EP[n]_TXTYPE register selects the endpoint number and transaction protocol to use for the currently selected transmit endpoint. There is a USB_EP[n]_TXTYPE register for each transmit endpoint. Note that this register is only used in host mode.

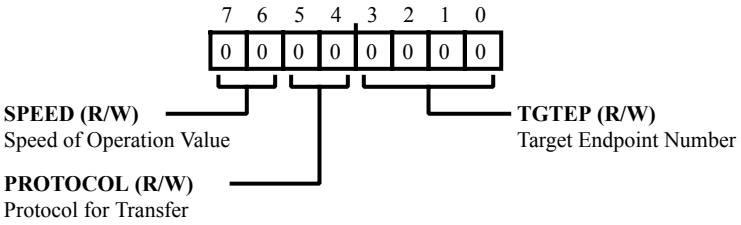

Figure 22-73: USB_EP[n]_TXTYPE Register Diagram

Table 22-51: USB_EP[n]_TXTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/W)	SPEED	Speed of Operation Value. The USB_EP[n]_TXTYPE.SPEED bits select the USB controller operating speed for the endpoint when acting as a host connected to devices through a hub. In these instances. the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When not connected to devices through a hub, program this field with 00.	
		0	Same Speed as the Core
		1	High Speed
		2	Full Speed
		3	Low Speed
5:4 (R/W)	PROTOCOL	Protocol for Transfer. The USB_EP[n]_TXTYPE.PROTOCOL bits select the transfer protocol for the endpoint.	
		0	Control
		1	Isochronous
		2	Bulk
		3	Interrupt
3:0 (R/W)	TGTEP	Target Endpoint Number. The USB_EP[n]_TXTYPE.TGTEP bits select (for endpoints 1-11) the target endpoint. This value should be set to the endpoint number contained in the transmit endpoint descriptor returned during device enumeration. Endpoint 0 always uses target endpoint number 0. (Enumeration values not shown are reserved.)	

Table 22-51: USB_EP[n]_TXTYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	Endpoint 0
		1	Endpoint 1
		2	Endpoint 2
		3	Endpoint 3
		4	Endpoint 4
		5	Endpoint 5
		6	Endpoint 6
		7	Endpoint 7
		8	Endpoint 8
		9	Endpoint 9
		10	Endpoint 10
		11	Endpoint 11
		12	Endpoint 12
		13	Endpoint 13
		14	Endpoint 14
		15	Endpoint 15

Function Address Register

The USB_FADDR register contains the device address used in peripheral mode. The processor writes this register with the address received through a SET_ADDRESS command from the host.

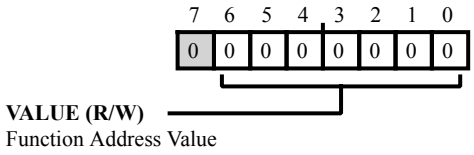

Figure 22-74: USB_FADDR Register Diagram

Table 22-52: USB_FADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Function Address Value. The USB_FADDR.VALUE bits contain the address of the peripheral part of the transaction.

FIFO Byte (8-Bit) Register

Writes to the USB_FIFOB[n] register go to the endpoint Tx FIFO and reads from the USB_FIFOB[n] register come from the endpoint Rx FIFO. The USB_FIFOB[n], USB_FIFOH[n], and USB_FIFO[n] registers are one and the same. These registers exist at the same address. Typically, programs should load and unload the FIFO using word (USB_FIFO[n] register) writes and reads, which are more efficient. Only if the USB packet is a non-word (4-byte) size should the program use a half-word (USB_FIFOH[n] register) or byte (USB_FIFOB[n] register) read or write at the end when loading or unloading the FIFO.

Note that (for correct USB controller operation) programs should not mix byte, half-word, or word accesses, except for the last few bytes if the size of the packet is odd (not a multiple of the size they were using).

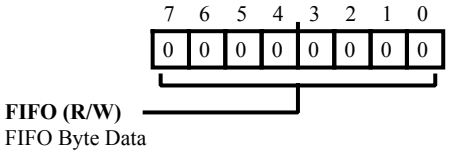

Figure 22-75: USB_FIFOB[n] Register Diagram

Table 22-53: USB_FIFOB[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	FIFO	FIFO Byte Data. The USB_FIFOB[n].FIFO bits provide byte access to the USB Tx and Rx endpoint FIFOs.

FIFO Half-Word (16-Bit) Register

Writes to the USB_FIFOH[n] register go to the endpoint Tx FIFO and reads from the USB_FIFOH[n] register come from the endpoint Rx FIFO. The USB_FIFOB[n], USB_FIFOH[n], and USB_FIFO[n] registers are one and the same. These registers exist at the same address. Typically, programs should load and unload the FIFO using word (USB_FIFO[n] register) writes and reads, which are more efficient. Only if the USB packet is a non-word (4-byte) size should the program use a half-word (USB_FIFOH[n] register) or byte (USB_FIFOB[n] register) read or write at the end when loading or unloading the FIFO.

Note that (for correct USB controller operation) programs should not mix byte, half-word, or word accesses, except for the last few bytes if the size of the packet is odd (not a multiple of the size they were using).

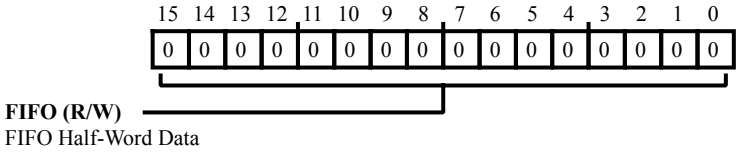

Figure 22-76: USB_FIFOH[n] Register Diagram

Table 22-54: USB_FIFOH[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	FIFO	FIFO Half-Word Data. The USB_FIFOH[n].FIFO bits provide half-word access to the USB Tx and Rx endpoint FIFOs.

FIFO Word (32-Bit) Register

Writes to the USB_FIFO[n] register go to the endpoint Tx FIFO and reads from the USB_FIFO[n] register come from the endpoint Rx FIFO. The USB_FIFOB[n], USB_FIFOH[n], and USB_FIFO[n] registers are one and the same. These registers exist at the same address. Typically, programs should load and unload the FIFO using word (USB_FIFO[n] register) writes and reads, which are more efficient. Only if the USB packet is a non-word (4-byte) size should the program use a half-word (USB_FIFOH[n] register) or byte (USB_FIFOB[n] register) read or write at the end when loading or unloading the FIFO.

Note that (for correct USB controller operation) programs should not mix byte, half-word, or word accesses, except for the last few bytes if the size of the packet is odd (not a multiple of the size they were using).

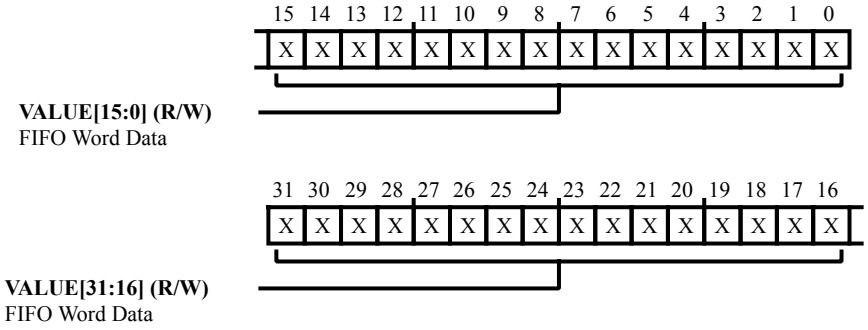

Figure 22-77: USB_FIFO[n] Register Diagram

Table 22-55: USB_FIFO[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	FIFO Word Data. The USB_FIFO[n].VALUE bits provide word access to the USB Tx and Rx endpoint FIFOs.

Frame Number Register

The USB_FRAME register contains the frame number of the last received frame. The data in this register has bit 10 as the MSB and bit 0 as the LSB.

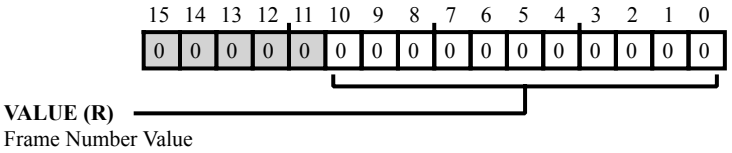

Figure 22-78: USB_FRAME Register Diagram

Table 22-56: USB_FRAME Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/NW)	VALUE	Frame Number Value. The USB_FRAME.VALUE bits contains the frame number of the last received frame. The data in this field has bit 10 as the MSB and bit 0 as the LSB.

Full-Speed EOF 1 Register

The USB_FS_EOF1 register defines the minimum time gap allowed between the start of the last transaction and the end of frame for full-speed transactions.

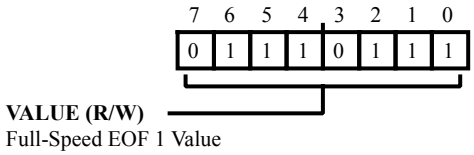

Figure 22-79: USB_FS_EOF1 Register Diagram

Table 22-57: USB_FS_EOF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Full-Speed EOF 1 Value. The USB_FS_EOF1.VALUE bits set the time before end of frame to stop beginning new transactions (in units of 533.3ns) for full-speed transactions. The default setting corresponds to 63.46us.

ID Control

The USB_IDCTL register can be used to override the ID pin and force the controller to act as an A-device or B-device.

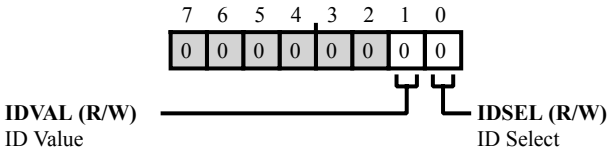

Figure 22-80: USB_IDCTL Register Diagram

Table 22-58: USB_IDCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	IDVAL	ID Value. When the USB_IDCTL.IDSEL bit =1, the USB_IDCTL.IDVAL bit sets the value of the ID input to the controller. This bit has no effect if USB_IDCTL.IDSEL=0.	
		0	A-Device
		1	B-Device
0 (R/W)	IDSEL	ID Select. The USB_IDCTL.IDSEL bit selects the source of the ID input to the controller. This can be used to bypass the ID input pin and force the controller to act as an A-device or B-device.	
		0	ID pin selected for controller input
		1	IDCTL[1] selected for controller input

Common Interrupts Enable Register

The USB_IEN register enables interrupts for USB controller system sources. Enabling an interrupt in this register directs the USB controller to generate an interrupt if the corresponding interrupt pending bit in the USB_IRQ register is set.

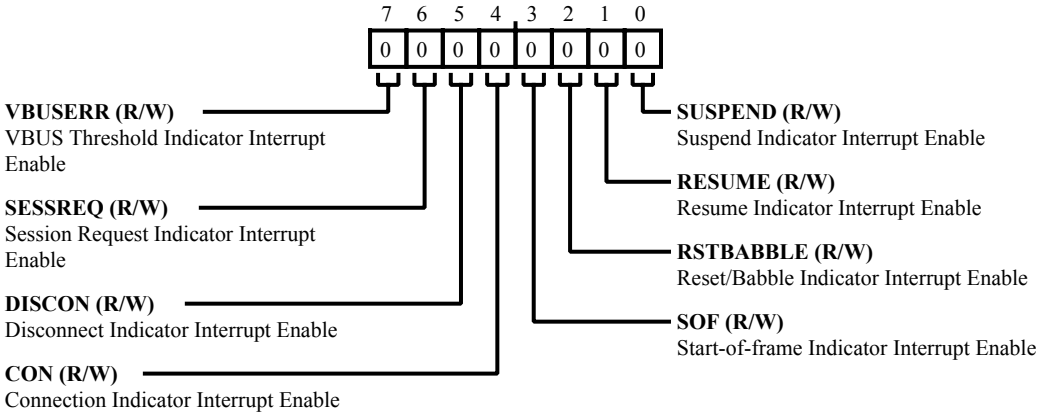

Figure 22-81: USB_IEN Register Diagram

Table 22-59: USB_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	VBUSERR	VBUS Threshold Indicator Interrupt Enable. The USB_IEN.VBUSERR bit enables the USB_IRQ.VBUSERR interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
6 (R/W)	SESSREQ	Session Request Indicator Interrupt Enable. The USB_IEN.SESSREQ bit enables the USB_IRQ.SESSREQ interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
5 (R/W)	DISCON	Disconnect Indicator Interrupt Enable. The USB_IEN.DISCON bit enables the USB_IRQ.DISCON interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
4 (R/W)	CON	Connection Indicator Interrupt Enable. The USB_IEN.CON bit enables the USB_IRQ.CON interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt

Table 22-59: USB_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	SOF	Start-of-frame Indicator Interrupt Enable. The USB_IEN.SOF bit enables the USB_IRQ.SOF interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
2 (R/W)	RSTBABBLE	Reset/Babble Indicator Interrupt Enable. The USB_IEN.RSTBABBLE bit enables the USB_IRQ.RSTBABBLE interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
1 (R/W)	RESUME	Resume Indicator Interrupt Enable. The USB_IEN.RESUME bit enables the USB_IRQ.RESUME interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
0 (R/W)	SUSPEND	Suspend Indicator Interrupt Enable. The USB_IEN.SUSPEND bit enables the USB_IRQ.SUSPEND interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt

Index Register

The USB_INDEX register contains an index value for mirrored addressing of USB controller endpoint control and status registers.

There is one set of registers, but they are mirrored at two address locations if the endpoint is selected by the USB_INDEX register. An endpoint's register set only appears in the indexed location if the USB_INDEX register is written with that endpoint number. You can read/write an endpoint's register in either the directly mapped location which is always visible, or in the indexed location which is only visible if the USB_INDEX register is written with the endpoint number. The USB_INDEX register and indexed address locations only affect address decoding. For example, loading a 0 into the USB_INDEX register selects endpoint 0 access.

The USB_INDEX register can be used for indexed access of the directly mapped control/status registers from USB controller address offset 0x100-0x1FF. For products supporting the dynamic FIFO size feature, the endpoint Tx/Rx size and address registers always use the USB_INDEX register, there is no direct mapping for these endpoint specific registers. The multipoint USB_MP[n]_TXFUNCADDR, USB_MP[n]_TXHUBADDR, USB_MP[n]_TXHUBPORT, USB_MP[n]_RXFUNCADDR, USB_MP[n]_RXHUBADDR, and USB_MP[n]_RXHUBPORT registers only have direct mapping, no indexed mapping.

Before accessing an endpoint's control/status registers using the indexed range, write the endpoint number to the USB_INDEX register to ensure that the correct control/status registers appear in the indexed range of the memory map.

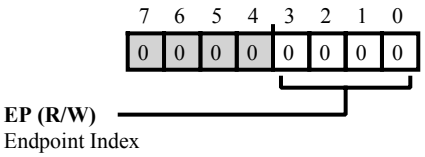

Figure 22-82: USB_INDEX Register Diagram

Table 22-60: USB_INDEX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	EP	Endpoint Index. The USB_INDEX.EP bits selects mirrored access for an endpoints indexed control and status registers. Valid values for this bit field are 0-11.

Receive Interrupt Register

The USB_INTRRX register indicates which interrupts are currently active for the receive (Rx) endpoints. Note that the USB controller automatically clears this register when it is read.

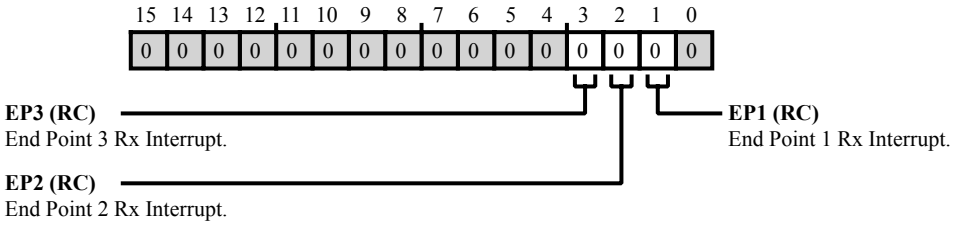

Figure 22-83: USB_INTRRX Register Diagram

Table 22-61: USB_INTRRX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (RC/NW)	EP3	End Point 3 Rx Interrupt.. The USB_INTRRX.EP3 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
2 (RC/NW)	EP2	End Point 2 Rx Interrupt.. The USB_INTRRX.EP2 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
1 (RC/NW)	EP1	End Point 1 Rx Interrupt.. The USB_INTRRX.EP1 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Receive Interrupt Enable Register

The USB_INTRRXE register enables interrupts for the receive (Rx) endpoints. Enabling an interrupt in this register directs the USB controller to generate an interrupt if the corresponding interrupt pending bit in the USB_INTRRX register is set.

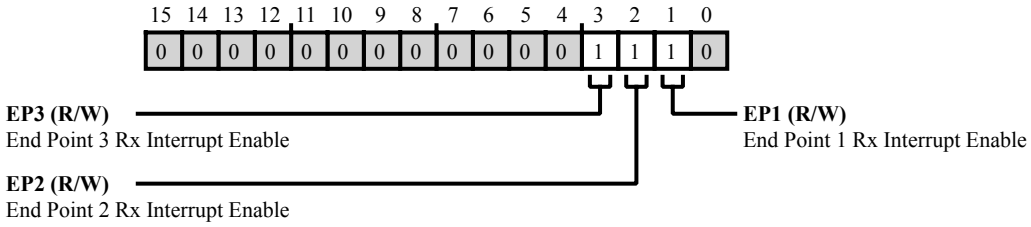

Figure 22-84: USB_INTRRXE Register Diagram

Table 22-62: USB_INTRRXE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	EP3	End Point 3 Rx Interrupt Enable. The USB_INTRRXE.EP3 bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
2 (R/W)	EP2	End Point 2 Rx Interrupt Enable. The USB_INTRRXE.EP2 bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
1 (R/W)	EP1	End Point 1 Rx Interrupt Enable. The USB_INTRRXE.EP1 bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt

Transmit Interrupt Register

The USB_INTRTX register indicates which interrupts are currently active for endpoint 0 and the transmit (Tx) endpoints. Note that the USB controller automatically clears this register when it is read.

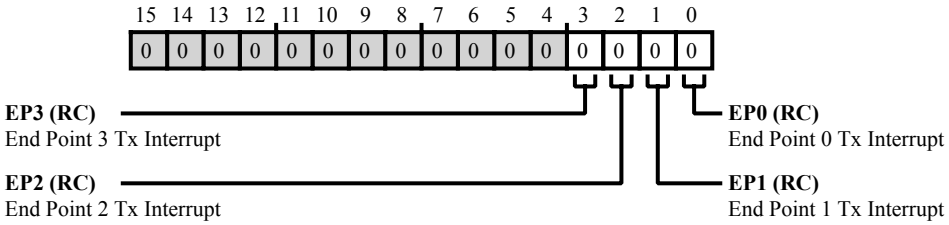

Figure 22-85: USB_INTRTX Register Diagram

Table 22-63: USB_INTRTX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (RC/NW)	EP3	End Point 3 Tx Interrupt. The USB_INTRTX.EP3 bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
2 (RC/NW)	EP2	End Point 2 Tx Interrupt. The USB_INTRTX.EP2 bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
1 (RC/NW)	EP1	End Point 1 Tx Interrupt. The USB_INTRTX.EP1 bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
0 (RC/NW)	EP0	End Point 0 Tx Interrupt. The USB_INTRTX.EP0 bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Transmit Interrupt Enable Register

The USB_INTRTXE register enables interrupts for endpoint 0 and the transmit (Tx) endpoints. Enabling an interrupt in this register directs the USB controller to generate an interrupt if the corresponding interrupt pending bit in the USB_INTRTX register is set.

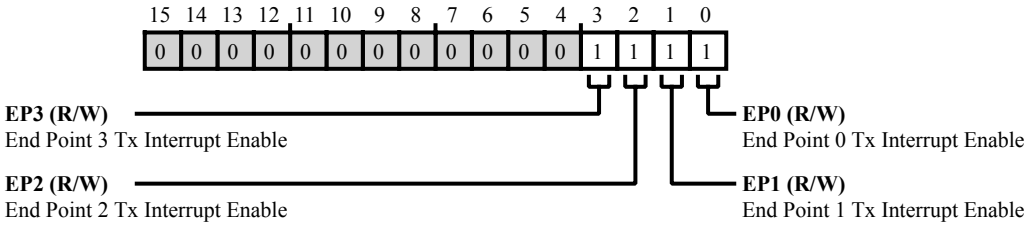

Figure 22-86: USB_INTRTXE Register Diagram

Table 22-64: USB_INTRTXE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	EP3	End Point 3 Tx Interrupt Enable. The USB_INTRTXE.EP3 bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
2 (R/W)	EP2	End Point 2 Tx Interrupt Enable. The USB_INTRTXE.EP2 bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
1 (R/W)	EP1	End Point 1 Tx Interrupt Enable. The USB_INTRTXE.EP1 bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
0 (R/W)	EP0	End Point 0 Tx Interrupt Enable. The USB_INTRTXE.EP0 bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt

Common Interrupts Register

The USB_IRQ register indicates which interrupts are currently active for USB controller system sources. Note that the USB controller automatically clears this register when it is read.

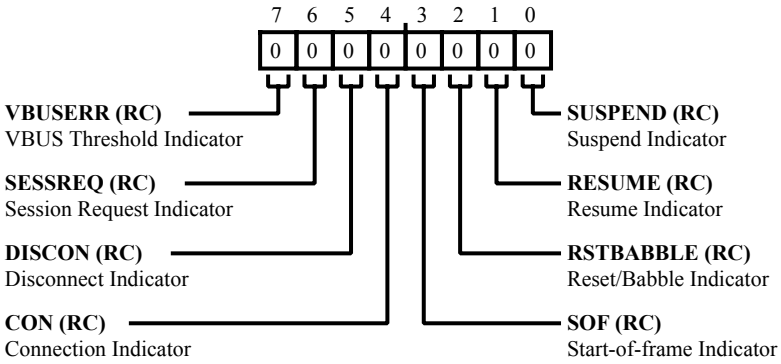

Figure 22-87: USB_IRQ Register Diagram

Table 22-65: USB_IRQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (RC/NW)	VBUSERR	VBUS Threshold Indicator. The USB_IRQ.VBUSERR bit indicates whether the USB controller has detected that the VBUS is below the VBUS valid threshold. This bit is valid only when the USB controller is an A device. Note that the USB_IRQ.VBUSERR bit and the USB_VBUS_CTL.DRVINT bit share an interrupt source line.	
		0	No Interrupt
		1	Interrupt Pending
6 (RC/NW)	SESSREQ	Session Request Indicator. The USB_IRQ.SESSREQ bit indicates whether the USB controller has detected a session request signal. This bit is valid only when the USB controller is an A device.	
		0	No Interrupt
		1	Interrupt Pending
5 (RC/NW)	DISCON	Disconnect Indicator. The USB_IRQ.DISCON bit indicates whether the USB controller has detected a device disconnect (host mode) or has detected a session end (peripheral mode).	
		0	No Interrupt
		1	Interrupt Pending

Table 22-65: USB_IRQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (RC/NW)	CON	Connection Indicator. The USB_IRQ.CON bit indicates whether the USB controller has detected a device connection. This bit is valid only in host mode.	
		0	No Interrupt
		1	Interrupt Pending
3 (RC/NW)	SOF	Start-of-frame Indicator. The USB_IRQ.SOF bit indicates whether the USB controller has detected a start of frame.	
		0	No Interrupt
		1	Interrupt Pending
2 (RC/NW)	RSTBABBLE	Reset/Babble Indicator. The USB_IRQ.RSTBABBLE bit indicates whether the USB controller has detected reset signalling on the bus. In host mode, the USB controller also indicates when the USB controller detects babble. Note that the USB_IRQ.RSTBABBLE bit is only active after the first SOF has been sent.	
		0	No Interrupt
		1	Interrupt Pending
1 (RC/NW)	RESUME	Resume Indicator. The USB_IRQ.RESUME bit indicates whether the USB controller has detected resume signaling on the bus while the USB controller is in suspend mode.	
		0	No Interrupt
		1	Interrupt Pending
0 (RC/NW)	SUSPEND	Suspend Indicator. The USB_IRQ.SUSPEND bit indicates whether the USB controller has detected suspend signalling on the bus. This bit is valid only in peripheral mode.	
		0	No Interrupt
		1	Interrupt Pending

Link Information Register

The USB_LINKINFO register specifies the PHY-related delays.

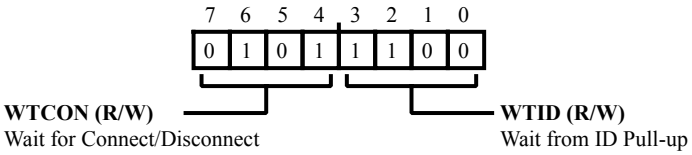

Figure 22-88: USB_LINKINFO Register Diagram

Table 22-66: USB_LINKINFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/W)	WTCON	Wait for Connect/Disconnect. The USB_LINKINFO.WTCON bits set the wait to be applied to allow for the users connect or disconnect filter in units of 533.3ns. The default settings corresponds to 2.667us
3:0 (R/W)	WTID	Wait from ID Pull-up. The USB_LINKINFO.WTID bits set the delay to be applied from IDPULLUP being asserted to IDDIG being considered valid in units of 4.3690ms. The default corresponds to 52.43ms UTMI+ spec says 50ms min. OTG spec does not have timing requirements (it doesn't assume a programmable pull-up that is only sampled during session start). Micro-USB cable spec says that the ID pin is greater than 10 Ohms when shorted and less than 100k Ohms when open.

LPM Attribute Register

The USB_LPM_ATTR register defines the link power management (LPM) attributes for LPM transactions and sleep/wake operation. In peripheral mode, the USB_LPM_ATTR register contains values received in the most recent, accepted (ACK'd) LPM transaction. In host mode, the USB_LPM_ATTR register contains values (loaded by software) that set up the next LPM transaction. The USB controller inserts the LPM values within the next LPM transaction.

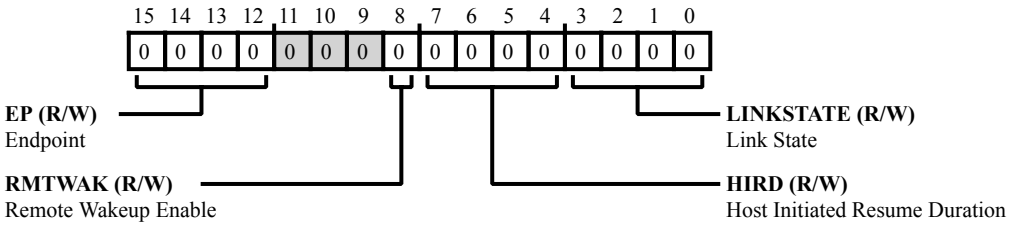

Figure 22-89: USB_LPM_ATTR Register Diagram

Table 22-67: USB_LPM_ATTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:12 (R/W)	EP	Endpoint. The USB_LPM_ATTR.EP bits select the endpoint in the token packet of the LPM transaction.	
8 (R/W)	RMTWAK	Remote Wakeup Enable. The USB_LPM_ATTR.RMTWAK bit enables remote wakeup. This bit is applied on a temporary basis only and is only applied to the current suspend state. After the current suspend cycle, the remote wakeup capability that was negotiated during enumeration applies.	
		0	Disable Remote Wakeup
		1	Enable Remote Wakeup
7:4 (R/W)	HIRD	Host Initiated Resume Duration. The USB_LPM_ATTR.HIRD bits select the host initiated resume duration. This value is the minimum time that the host drives resume on the bus. The value in this register corresponds to an actual resume time of: Resume Time = 50us + HIRD*75us. This equation produces results in a range of 50us to 1200us.	
3:0 (R/W)	LINKSTATE	Link State. The USB_LPM_ATTR.LINKSTATE bits is value is provided by the host to the peripheral to indicate what state the peripheral must transition to after the receipt and acceptance of a LPM transaction. (Enumerations not shown are reserved.)	
		1	Sleep State (L1)

LPM Control Register

The USB_LPM_CTL register controls link power management (LPM) operations, including LPM enable, NAK, resume, and mode transition.

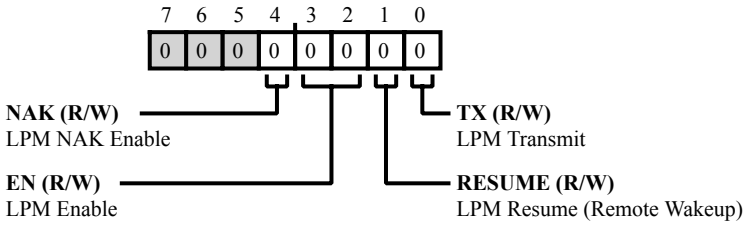

Figure 22-90: USB_LPM_CTL Register Diagram

Table 22-68: USB_LPM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	NAK	LPM NAK Enable. The USB_LPM_CTL.NAK bit enables (in peripheral mode) a NAK-all-non-LPM transactions mode for all end points, forcing a NAK response to all transactions other than an LPM transaction. This bit only takes effect after the controller has been LPM suspended. In this case, the USB controller continues to NAK, until this bit has been cleared by software.	
		0	Disable LPM NAK
		1	Enable LPM NAK
3:2 (R/W)	EN	LPM Enable. The USB_LPM_CTL.EN bits enable (In peripheral mode) LPM operations. The LPM operation may be enabled at different levels, which determine the response of the USB controller to LPM transactions.	
		0	Disable LPM LPM and extended transactions are not supported. The USB controller does not respond to LPM transactions, and these transaction timeout.
		1	Disable LPM LPM and extended transactions are not supported. The USB controller does not respond to LPM transactions, and these transaction timeout.
		2	Enable Extended Transactions LPM is not supported, but extended transactions are supported. The USB controller responds to an LPM transaction with a STALL.
		3	Enable LPM and Extended Transactions Both LPM and extended transactions are supported. The USB controller responds with a NYET or an ACK as determined by the value of LPMXMT and other conditions.

Table 22-68: USB_LPM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	RESUME	LPM Resume (Remote Wakeup). The USB_LPM_CTL.RESUME bit initiates resume (remote wakeup). This bits operation differs from the USB_POWER.RESUME bit in that the LPM resume signal timing is controlled by hardware. When set, the USB controller asserts resume signaling for 50us in host mode or asserts resume signaling for the time specified by the USB_LPM_ATTR.HIRD field in device mode. The USB_LPM_CTL.RESUME bit is self clearing.	
		0	No Action
		1	LPM Resume
0 (R/W)	TX	LPM Transmit. The USB_LPM_CTL.TX bit puts the USB controller in LPM transmit mode, but this mode has differing operations in host mode versus peripheral mode. In peripheral mode, this bit is set by software to instruct the controller to transition to the L1 state upon receipt of the next LPM transaction. This bit is only effective if LPM enable (USB_LPM_CTL.EN) is set to 0x3. The LPM transmit enable bit can be set in the same cycle as LPM enable. If USB_LPM_CTL.TX and USB_LPM_CTL.EN are enabled, the USB controller can respond in the following ways: • If no data is pending (all transmit FIFOs are empty), the USB controller responds with an ACK, clears the USB_LPM_CTL.TX bit, and generates a software interrupt. • If data is pending (data resides in at least one transmit FIFO), the USB controller responds with a NYET, does not clear the USB_LPM_CTL.TX bit, and generates a software interrupt. In host mode, this bit is set by software to transmit an LPM transaction. This bit is self clearing. The USB controller clears this bit immediately on receipt of any token or after three timeouts have occurred.	
		0	Disable LPM Tx
		1	Enable LPM Tx

LPM Function Address Register

The USB_LPM_FADDR register selects the link power management (LPM) function address.

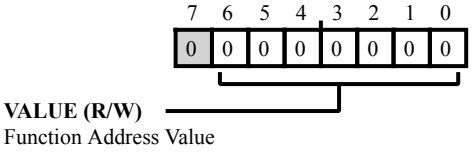

Figure 22-91: USB_LPM_FADDR Register Diagram

Table 22-69: USB_LPM_FADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Function Address Value. The USB_LPM_FADDR.VALUE bits hold the LPM function address value that the USB controller places in the LPM payload.

LPM Interrupt Enable Register

The USB_LPM_IEN register enables the link power management (LPM) related interrupts. When an interrupt is enabled in this register and the corresponding interrupt is pending in USB_LPM_IRQ, the USB controller generates the interrupt. When an interrupt is disabled in this register, the corresponding interrupt may be pending in USB_LPM_IRQ, but the USB controller does not generate an interrupt.

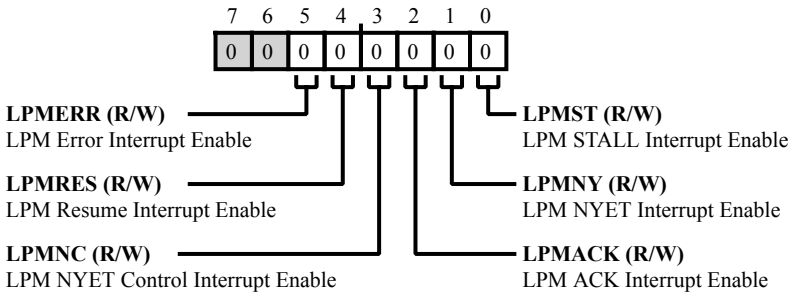

Figure 22-92: USB_LPM_IEN Register Diagram

Table 22-70: USB_LPM_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	LPMERR	LPM Error Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt
4 (R/W)	LPMRES	LPM Resume Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt
3 (R/W)	LPMNC	LPM NYET Control Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt
2 (R/W)	LPMACK	LPM ACK Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt
1 (R/W)	LPMNY	LPM NYET Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt

Table 22-70: USB_LPM_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	LPMST	LPM STALL Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt

LPM Interrupt Status Register

The USB_LPM_IRQ register indicates link power management (LPM) related interrupt status. The USB controller clears this register when it is read.

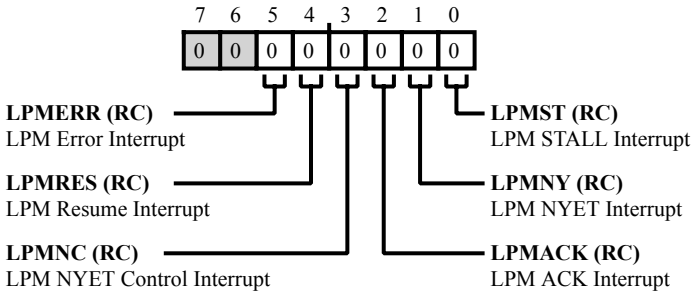

Figure 22-93: USB_LPM_IRQ Register Diagram

Table 22-71: USB_LPM_IRQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (RC/NW)	LPMERR	LPM Error Interrupt. The USB_LPM_IRQ.LPMERR bit indicates an LPM error interrupt condition, but this interrupt has differing conditions for host mode versus peripheral mode. In peripheral mode, this bit is set if an LPM transaction is received that has a LinkState field that is not supported. The USB controller responds to the transaction with a STALL. Note that the USB controller updates the USB_LPM_ATTR register, so software can observe the non compliant LPM packet payload. In host mode, this bit is set if the response to a LPM transaction is received with a bit stuff or PID error. No suspend occurs and the state of the device is now unknown.	
		0	No Interrupt Pending
		1	Interrupt Pending
4 (RC/NW)	LPMRES	LPM Resume Interrupt. The USB_LPM_IRQ.LPMRES bit indicates that the USB controller has been resumed for any reason. This bit is mutually exclusive from the USB_POWER.RESUME bit.	
		0	No Interrupt Pending
		1	Interrupt Pending

Table 22-71: USB_LPM_IRQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (RC/NW)	LPMNC	LPM NYET Control Interrupt. The USB_LPM_IRQ.LPMNC bit indicates an LPM NYET control interrupt condition, but this interrupt has differing conditions for host mode versus peripheral mode. In peripheral mode, this bit is set when an LPM transaction is received, and the USB controller responds with a NYET due to data pending in the transmit FIFOs. This interrupt may only occur when the USB_LPM_CTL.EN field is set to 11, the USB_LPM_CTL.TX field is set to 1, and there is data pending in the transmit FIFOs. In host mode, this bit is set when an LPM transaction has been transmitted, but has failed to complete. The transaction failure must be because a timeout occurred or be because there were bit errors in the response for three attempts.	
		0	No Interrupt Pending
		1	Interrupt Pending
2 (RC/NW)	LPMACK	LPM ACK Interrupt. The USB_LPM_IRQ.LPMACK bit indicates an LPM ACK interrupt condition, but this interrupt has differing conditions for host mode versus peripheral mode. In peripheral mode, this bit is set when an LPM transaction is received, and the USB controller responds with an ACK. This interrupt may only occur when the USB_LPM_CTL.EN field is set to 11, the USB_LPM_CTL.TX field is set to 1, and there is no data pending in the controller transmit FIFOs. In host mode, this bit is set when an LPM transaction is transmitted, and the device responds with an ACK.	
		0	No Interrupt Pending
		1	Interrupt Pending
1 (RC/NW)	LPMNY	LPM NYET Interrupt. The USB_LPM_IRQ.LPMNY bit indicates an LPM NYET interrupt condition, but this interrupt has differing conditions for host mode versus peripheral mode. In peripheral mode, this bit is set when an LPM transaction is received, and the USB controller responds with a NYET. This interrupt may only occur when the USB_LPM_CTL.EN field is set to 11, and the USB_LPM_CTL.TX field is set to 0. In host mode, this bit is set when an LPM transaction is transmitted and the device responds with a NYET.	
		0	No Interrupt Pending
		1	Interrupt Pending

Table 22-71: USB_LPM_IRQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (RC/NW)	LPMST	LPM STALL Interrupt. The `USB_LPM_IRQ.LPMST` bit indicates an LPM STALL interrupt condition, but this interrupt has differing conditions for host mode versus peripheral mode. This bit is set when an LPM transaction is received, and the USB controller responds with a STALL. This interrupt may only occur when the `USB_LPM_CTL.EN` field is set to 01. In host mode, this bit is set when an LPM transaction is transmitted, and the device responds with a STALL.	
		0	No Interrupt Pending
		1	Interrupt Pending

Low-Speed EOF 1 Register

The USB_LS_EOF1 register defines the minimum time gap allowed between the start of the last transaction and the end of frame for low-speed transactions.

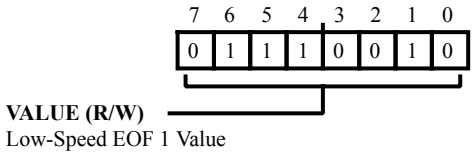

Figure 22-94: USB_LS_EOF1 Register Diagram

Table 22-72: USB_LS_EOF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Low-Speed EOF 1 Value. The USB_LS_EOF1.VALUE bits set the time before end of frame to stop beginning new transactions (in units of 1.067us) for low-speed transactions. The default setting corresponds to 121.6us.

MPn Receive Function Address Register

The USB_MP[n]_RXFUNCADDR register specifies the receive endpoint's target address in host mode. This register is not used in device mode. Note that the USB_MP[n]_RXFUNCADDR register does not exist for EP0.

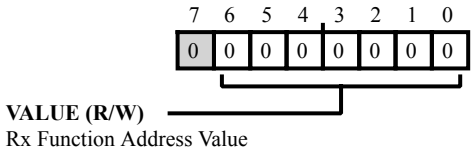

Figure 22-95: USB_MP[n]_RXFUNCADDR Register Diagram

Table 22-73: USB_MP[n]_RXFUNCADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Rx Function Address Value. The USB_MP[n]_RXFUNCADDR.VALUE bits hold the address of the target device for this endpoint.

MPn Receive Hub Address Register

The USB_MP[n]_RXHUBADDR register specifies the hub address of the endpoint in host mode. This register is not used in device mode. Note that this register only needs to be programmed when a full-speed or low-speed device is connected to a high-speed hub, which carries out the necessary transaction translation. Note that the USB_MP[n]_RXHUBADDR register does not exist for EP0.

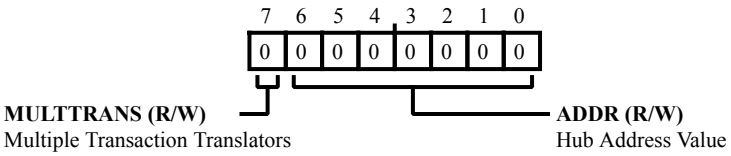

Figure 22-96: USB_MP[n]_RXHUBADDR Register Diagram

Table 22-74: USB_MP[n]_RXHUBADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	MULTTRANS	Multiple Transaction Translators. The USB_MP[n]_RXHUBADDR.MULTTRANS bit should be set if the hub has multiple transaction translators.	
		0	Single Transaction Translator
		1	Multiple Transaction Translators
6:0 (R/W)	ADDR	Hub Address Value. The USB_MP[n]_RXHUBADDR.ADDR bits hold the address of the hub to which this device is connected.	

MPn Receive Hub Port Register

The USB_MP[n]_RXHUBPORT register specifies the hub port for full-speed and low-speed endpoints in host mode. This register is not used in device mode. The USB_MP[n]_RXHUBPORT register lets the USB controller support SPLIT transactions. Note that the USB_MP[n]_RXHUBPORT register does not exist for EP0.

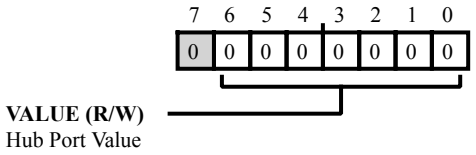

Figure 22-97: USB_MP[n]_RXHUBPORT Register Diagram

Table 22-75: USB_MP[n]_RXHUBPORT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Hub Port Value. The USB_MP[n]_RXHUBPORT.VALUE bits hold the hub port value of the target device for this endpoint.

MPn Transmit Function Address Register

The USB_MP[n]_TXFUNCADDR register specifies the transmit endpoint's target address in host mode. This register is not used in device mode. Note that the USB_MP[n]_TXFUNCADDR register must be setup for EP0. (The USB_MP[n]_RXFUNCADDR register does not exist for EP0.)

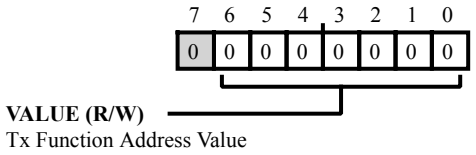

Figure 22-98: USB_MP[n]_TXFUNCADDR Register Diagram

Table 22-76: USB_MP[n]_TXFUNCADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Tx Function Address Value. The USB_MP[n]_TXFUNCADDR.VALUE bits hold the address of the target device for this endpoint.

MPn Transmit Hub Address Register

The USB_MP[n]_TXHUBADDR register specifies the hub address of the endpoint in host mode. This register is not used in device mode. Note that this register only needs to be programmed when a full-speed or low-speed device is connected to a high-speed hub, which carries out the necessary transaction translation. Also note that EP0 only uses the USB_MP[n]_TXHUBADDR register. (The USB_MP[n]_RXHUBADDR register does not exist for EP0.)

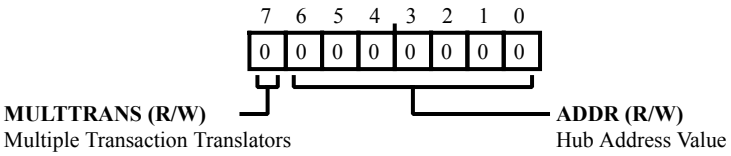

Figure 22-99: USB_MP[n]_TXHUBADDR Register Diagram

Table 22-77: USB_MP[n]_TXHUBADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	MULTTRANS	Multiple Transaction Translators. The USB_MP[n]_TXHUBADDR.MULTTRANS bit should be set if the hub has multiple transaction translators.	
		0	Single Transaction Translator
		1	Multiple Transaction Translators
6:0 (R/W)	ADDR	Hub Address Value. The USB_MP[n]_TXHUBADDR.ADDR bits hold the address of the hub to which this device is connected.	

MPn Transmit Hub Port Register

The USB_MP[n]_TXHUBPORT register specifies the hub port for full-speed and low-speed endpoints in host mode. This register is not used in device mode. The USB_MP[n]_TXHUBPORT register lets the USB controller support SPLIT transactions. EP0 only uses the USB_MP[n]_TXHUBPORT register. (The USB_MP[n]_RXHUBPORT register does not exist for EP0.)

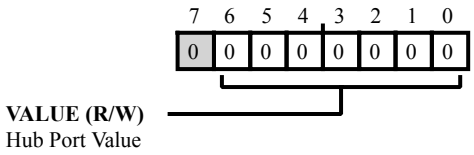

Figure 22-100: USB_MP[n]_TXHUBPORT Register Diagram

Table 22-78: USB_MP[n]_TXHUBPORT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Hub Port Value. The USB_MP[n]_TXHUBPORT.VALUE bits hold the hub port value of the target device for this endpoint.

FS PHY Control

The USB_PHY_CTL register provides configuration options for USB full speed operations.

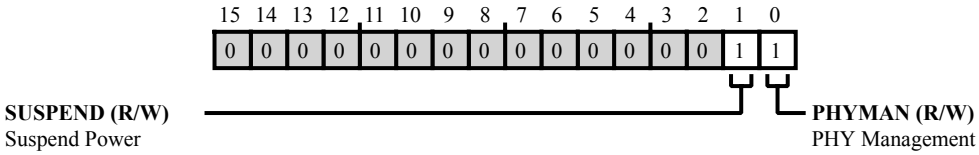

Figure 22-101: USB_PHY_CTL Register Diagram

Table 22-79: USB_PHY_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	SUSPEND	Suspend Power. The USB_PHY_CTL.SUSPEND bit powers down the output drivers, differential input comparator, ID input comparator and pull-up.	
		0	normal operation (power enabled)
		1	SUSPEND enabled
0 (R/W)	PHYMAN	PHY Management. The USB_PHY_CTL.PHYMAN bit manages inputs.	
		0	PHY inputs come from controller
		1	PHY inputs come from this register

FS PHY Status

The USB_PHY_STAT register is used to directly observe PHY inputs.

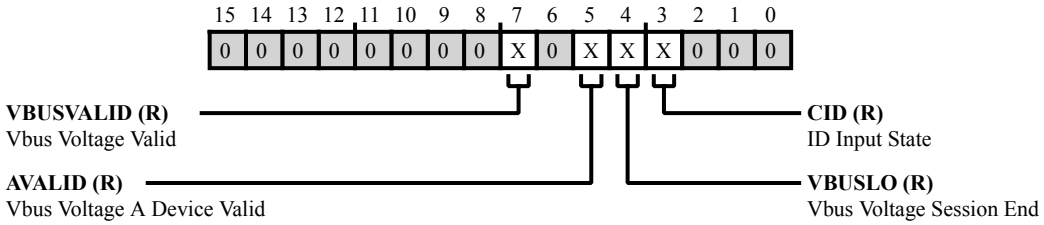

Figure 22-102: USB_PHY_STAT Register Diagram

Table 22-80: USB_PHY_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	VBUSVALID	Vbus Voltage Valid. The USB_PHY_STAT.VBUSVALID bit indicates if the voltage on Vbus is at a valid level for operation.	
		0	Vbus < 4.4V
		1	Vbus > 4.75V
5 (R/NW)	AVALID	Vbus Voltage A Device Valid. The USB_PHY_STAT.AVALID bit indicates if the voltage on Vbus is above the A-device session valid threshold.	
		0	Vbus < 0.8V
		1	Vbus > 2V
4 (R/NW)	VBUSLO	Vbus Voltage Session End. The USB_PHY_STAT.VBUSLO bit is used to determine if Vbus is above the session end voltage.	
		0	Vbus > 0.8V
		1	Vbus < 0.2V
3 (R/NW)	CID	ID Input State. The USB_PHY_STAT.CID bit reflects the state of the ID input. The ID input comparator is disabled when the PHY is in SUSPEND (references are powered down) and the USB_PHY_STAT.CID bit is forced to 0. There is a small current source enabled on the ID pin when the PHY is enabled to sense whether this pin is floating or shorted to GND.	

Power and Device Control Register

The USB_POWER register controls suspend and resume signaling and controls some operational aspects of the USB controller.

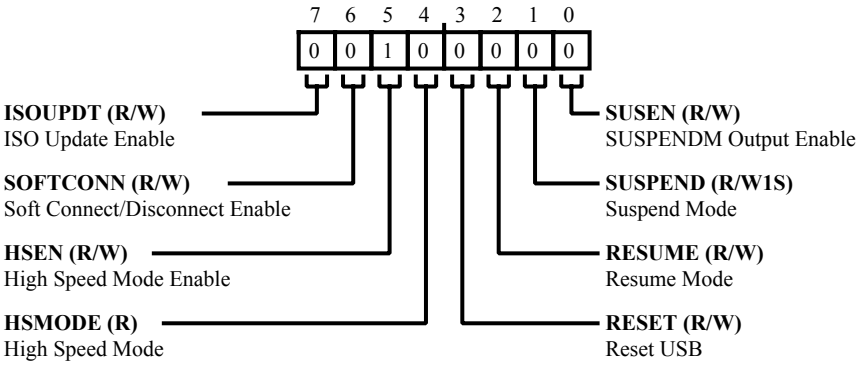

Figure 22-103: USB_POWER Register Diagram

Table 22-81: USB_POWER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	ISOUPDT	ISO Update Enable. The USB_POWER.ISOUPDT bit directs the USB controller to waits for an SOF token from the time TXPKTRDY is set before sending the packet. If an IN token is received before an SOF token, the USB controller send a zero length data packet. This USB_POWER.ISOUPDT bit only affects endpoints performing isochronous transfers. This bit only is valid in peripheral mode (USB_DEV_CTL.HOSTMODE = 0).	
		0	Disable ISO Update
		1	Enable ISO Update
6 (R/W)	SOFTCONN	Soft Connect/Disconnect Enable. In peripheral mode, the D+/- lines default to disconnected. Setting this bit will enable the D+/- termination resistors. This bit is automatically set when the DevCtl.Session bit is written with '1'. The USB_POWER.SOFTCONN bit enables USB controller soft connect/disconnect, enabling the termination resistors for USB_DP (Data +) and USB_DM (Data -) pins. When disabled, these pins are three-stated. Note that USB_POWER.SOFTCONN only is valid in peripheral mode (USB_DEV_CTL.HOSTMODE = 0).	
		0	Disable Soft Connect/Disconnect
		1	Enable Soft Connect/Disconnect

Table 22-81: USB_POWER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	HSEN	High Speed Mode Enable. The USB_POWER.HSEN bit enables USB controller negotiation for high speed (on devices supporting high-speed mode) when the device is reset by the hub/host. If disabled, the USB controller only operates in full-speed mode. When operating in full-speed mode, this bit should be cleared.	
		0	Disable Negotiation for HS Mode
		1	Enable Negotiation for HS Mode
4 (R/NW)	HSMODE	High Speed Mode. The USB_POWER.HSMODE bit indicates whether or not the USB controller successfully negotiated high-speed mode during a USB controller reset. In peripheral mode (USB_DEV_CTL.HOSTMODE = 0), this bit has valid data when the USB controller completes reset. In host mode (USB_DEV_CTL.HOSTMODE = 1), this bit has valid data when the USB_IRQ.RSTBABBLE bit is cleared, remaining valid for the duration of the session.	
		0	Full Speed Mode (HS fail during reset)
		1	High Speed Mode (HS success during reset)
3 (R/W)	RESET	Reset USB. The USB_POWER.RESET bit indicates (in both host and peripheral modes) that the USB controller has detected that reset signaling is present on the bus. In peripheral mode (USB_DEV_CTL.HOSTMODE = 0), this bit is read only, but in host mode (USB_DEV_CTL.HOSTMODE = 1), this bit is read/write, permitting the processor core to set the bit and initiate a USB controller reset.	
		0	No Reset
		1	Reset USB
2 (R/W)	RESUME	Resume Mode. The USB_POWER.RESUME bit directs the USB controller to generate resume signaling when the function is in suspend mode (USB_POWER.SUSPEND =1). The processor core should clear this bit after 10 ms (a maximum of 15 ms) to end resume signaling. When the USB controller is in host mode (USB_DEV_CTL.HOSTMODE = 1), the USB controller automatically sets the USB_POWER.RESUME bit when resume signaling from the target is detected while the USB controller is suspended.	
		0	Disable Resume Signaling
		1	Enable Resume Signaling

Table 22-81: USB_POWER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1S)	SUSPEND	Suspend Mode. When the USB controller is in host mode (USB_DEV_CTL.HOSTMODE = 1), the USB_POWER.SUSPEND bit enables suspend mode. When the USB controller is in peripheral mode (USB_DEV_CTL.HOSTMODE = 0), the USB controller sets the USB_POWER.SUSPEND bit on entry to suspend mode and clears the bit when the processor reads the USB_IRQ register. Note that the USB controller automatically clears this bit if the USB_POWER.RESUME bit is set.	
		0	Disable Suspend Mode (Host)
		1	Enable Suspend Mode (Host)
0 (R/W)	SUSEN	SUSPENDM Output Enable. The USB_POWER.SUSEN bit enables the SUSPENDM output (internal USB controller signal). When enabled, the SUSPENDM output signal is used by the USB controller PHY to power-down its drivers when the USB controller is not active.	
		0	Disable SUSPENDM Output
		1	Enable SUSPENDM Output

RAM Information Register

The USB_RAMINFO register provides information about the width of the USB controller RAM.

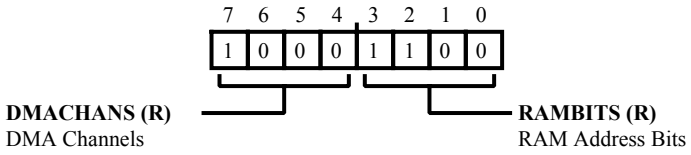

Figure 22-104: USB_RAMINFO Register Diagram

Table 22-82: USB_RAMINFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	DMACHANS	DMA Channels. The USB_RAMINFO.DMACHANS bits indicate the number of DMA channels.
3:0 (R/NW)	RAMBITS	RAM Address Bits. The USB_RAMINFO.RAMBITS bits indicate the number of RAM address bits. The USB controller FIFO RAM is 32-bits wide. The number of bytes in the FIFO RAM may be calculated from the formula: RAM_bytes = $2^{(RAM\_Bits+2)}$

EPn Request Packet Count Register

The USB_RQPKTCNT[n] register specifies (in host mode) the number of packets to request in a block transfer of one or more bulk packets of size USB_EP[n]_RXMAXP from a receive endpoint. This register only applies for receive endpoints 1 through 11 in host mode.

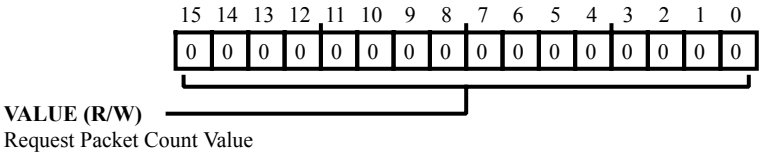

Figure 22-105: USB_RQPKTCNT[n] Register Diagram

Table 22-83: USB_RQPKTCNT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Request Packet Count Value. The USB_RQPKTCNT[n].VALUE bits specify the number of bulk packets to request in a block transfer from a receive endpoint. This field is used in conjunction with Auto Request feature (USB_EP[n]_RXCSR_H.AUTOREQ).

RX Double Packet Buffer Disable for Endpoints 1 to 3

When the bits in the USB_RXDPKTBUFDIS register =1, double packet buffering for the corresponding endpoint is disabled regardless of the end point FIFO size and the maximum packet size (MAXP) relationship. When the bits in the USB_RXDPKTBUFDIS register =0, they do not necessarily enable double packet buffering but rather allow double packet buffering to be determined based upon the end point FIFO size and MAXP size relationship.

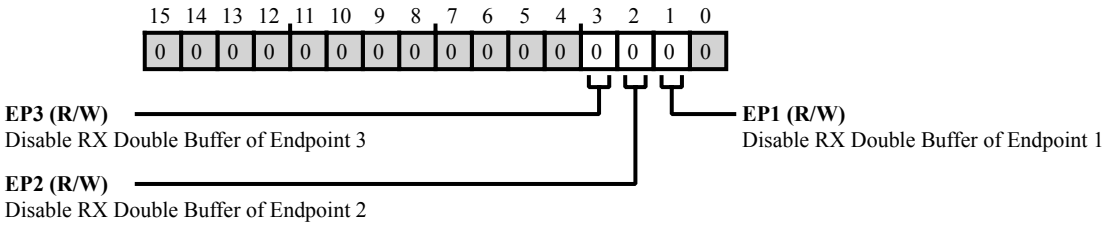

Figure 22-106: USB_RXDPKTBUFDIS Register Diagram

Table 22-84: USB_RXDPKTBUFDIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	EP3	Disable RX Double Buffer of Endpoint 3. The USB_RXDPKTBUFDIS.EP3 bit either allows or disables double packed buffering based on endpoint 3 FIFO size and MAXP size relationship.	
		0	Allow double packet buffering based on endpoint FIFO size and MAXP size relationship
		1	Disable double packet buffering
2 (R/W)	EP2	Disable RX Double Buffer of Endpoint 2. The USB_RXDPKTBUFDIS.EP2 bit either allows or disables double packed buffering based on endpoint 2 FIFO size and MAXP size relationship.	
		0	Allow double packet buffering based on endpoint FIFO size and MAXP size relationship
		1	Disable double packet buffering
1 (R/W)	EP1	Disable RX Double Buffer of Endpoint 1. The USB_RXDPKTBUFDIS.EP1 bit either allows or disables double packed buffering based on endpoint 1 FIFO size and MAXP size relationship.	
		0	Allow double packet buffering based on endpoint FIFO size and MAXP size relationship
		1	Disable double packet buffering

Software Reset Register

The USB_SOFT_RST register provides reset controls for the USB controller CLK domain and XCLK domain. The USB controller PHY operates in the controller's XCLK domain, and the USB controller interface to the processor core operates in the controller's CLK domain. Note that for correct operation, both of the reset control bits (USB_SOFT_RST.RST and USB_SOFT_RST.RSTX) should always be asserted simultaneously.

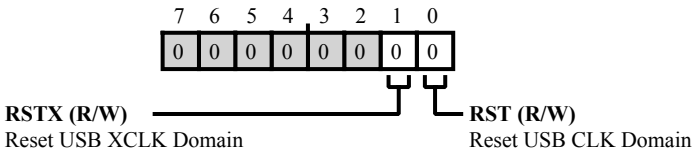

Figure 22-107: USB_SOFT_RST Register Diagram

Table 22-85: USB_SOFT_RST Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	RSTX	Reset USB XCLK Domain. The USB_SOFT_RST.RSTX bit resets logic in the USB XCLK domain. This bit is self clearing. Note that this bit should always be asserted simultaneously with the USB_SOFT_RST.RST bit.	
		0	No Reset
		1	Reset USB XCLK Domain
0 (R/W)	RST	Reset USB CLK Domain. The USB_SOFT_RST.RST bit resets logic in the USB CLK domain. This bit is self clearing. Note that this bit should always be asserted simultaneously with the USB_SOFT_RST.RSTX bit.	
		0	No Reset
		1	Reset USB CLK Domain

Testmode Register

The USB_TESTMODE register places the USB controller into test mode state and can also put the USB controller into one of the test modes for high-speed operation. For more information about these modes, see the USB 2.0 specification.

Note that the USB_TESTMODE register is not used in normal operation. Only one of the test mode (bits 0-6) selection bits may be set at a time.

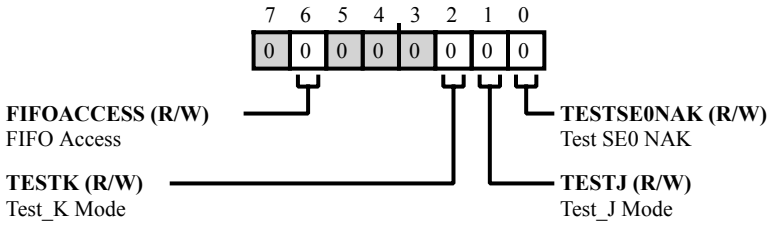

Figure 22-108: USB_TESTMODE Register Diagram

Table 22-86: USB_TESTMODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W)	FIFOACCESS	FIFO Access. The USB_TESTMODE bit directs the USB controller to transfer the packet in the endpoint 0 Tx FIFO to the endpoint 0 Rx FIFO. The bit is cleared automatically.
2 (R/W)	TESTK	Test_K Mode. The USB_TESTMODE.TESTK bit selects Test_K test mode. In this mode, the USB controller transmits a continuous K on the bus.
1 (R/W)	TESTJ	Test_J Mode. The USB_TESTMODE.TESTJ bit selects Test_J test mode. In this mode, the USB controller transmits a continuous J on the bus.
0 (R/W)	TESTSE0NAK	Test SE0 NAK. The USB_TESTMODE.TESTSE0NAK bit selects Test_SE0_NAK test mode, which applies only when the USB controller in high speed mode. In this mode, the USB controller remains in high-speed mode, but responds to any valid IN token with a NAK.

TX Double Packet Buffer Disable for Endpoints 1 to 3

When the bits in the USB_TXDPKTBUFDIS register =1, double packet buffering for the corresponding endpoint is disabled regardless of the end point FIFO size and the maximum packet size (MAXP) size relationship. When the bits in the USB_TXDPKTBUFDIS register =0, they do not necessarily enable double packet buffering but rather allow double packet buffering to be determined based upon the end point FIFO size and MAXP size relationship.

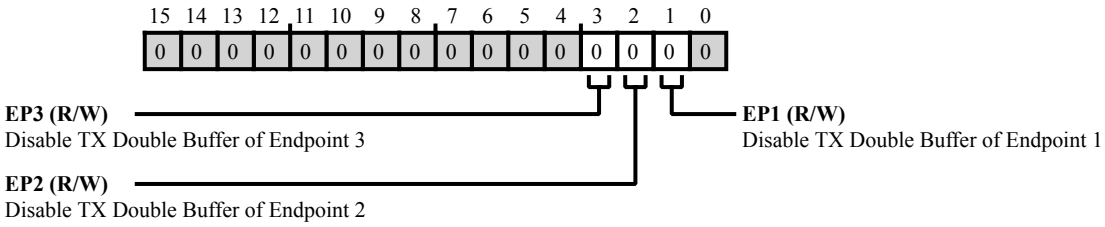

Figure 22-109: USB_TXDPKTBUFDIS Register Diagram

Table 22-87: USB_TXDPKTBUFDIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	EP3	Disable TX Double Buffer of Endpoint 3. The USB_TXDPKTBUFDIS.EP3 bit either allows or disables double packed buffering based on endpoint 3 FIFO size and MAXP size relationship.	
		0	Allow double packet buffering based on endpoint FIFO size and MAXP size relationship
		1	Disable double packet buffering
2 (R/W)	EP2	Disable TX Double Buffer of Endpoint 2. The USB_TXDPKTBUFDIS.EP2 bit either allows or disables double packed buffering based on endpoint 2 FIFO size and MAXP size relationship.	
		0	Allow double packet buffering based on endpoint FIFO size and MAXP size relationship
		1	Disable double packet buffering
1 (R/W)	EP1	Disable TX Double Buffer of Endpoint 1. The USB_TXDPKTBUFDIS.EP1 bit either allows or disables double packed buffering based on endpoint 1 FIFO size and MAXP size relationship.	
		0	Allow double packet buffering based on endpoint FIFO size and MAXP size relationship
		1	Disable double packet buffering

VBUS Control Register

The USB_VBUS_CTL controls USB controller VBUS related features.

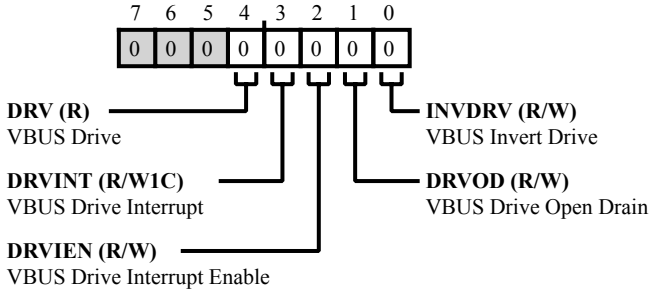

Figure 22-110: USB_VBUS_CTL Register Diagram

Table 22-88: USB_VBUS_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/NW)	DRV	VBUS Drive. The USB_VBUS_CTL.DRV bit indicates the state of the UTMI+ DrvVBUS signal from the USB controller.
3 (R/W1C)	DRVINT	VBUS Drive Interrupt. The USB_VBUS_CTL.DRVINT bit indicates the state of the DrvVBUSInt interrupt.
2 (R/W)	DRVIEN	VBUS Drive Interrupt Enable. The USB_VBUS_CTL.DRVIEN bit enables the DrvVBUS interrupt.
1 (R/W)	DRVOD	VBUS Drive Open Drain. The USB_VBUS_CTL.DRVOD selects whether the DrvVBUS output is open drain.
0 (R/W)	INVDRV	VBUS Invert Drive. The USB_VBUS_CTL.INVDRV bit selects whether the DrvVBUS output is inverted.

VBUS Pulse Length Register

The USB_VPLEN register defines the duration of the VBUS pulsing charge for SRP initiation.

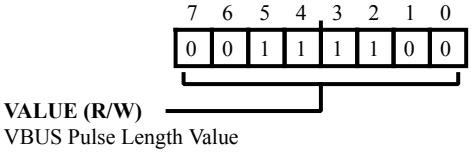

Figure 22-111: USB_VPLEN Register Diagram

Table 22-89: USB_VPLEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	VBUS Pulse Length Value. The USB_VPLEN.VALUE bits sets the duration of the VBUS pulsing charge in units of 546.1us. The default setting corresponds to 32.77ms. Note that VBUS pulsing was removed in the OTG specification v2.0, section 5.1.4.

23 Ethernet Media Access Controller (EMAC)

The EMAC peripheral in the processor enables network connectivity to applications through an Ethernet interface. The module is fully compliant to the following standards:

- Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications, Standard 802.3-2005, Institute of Electrical and Electronics Engineers (IEEE).
- Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, Standard 1588–2008, Institute of Electrical and Electronics Engineers (IEEE).
- Reduced Media Independent Interface Specification, Revision 1.2, RMII Consortium.

EMAC Features

The Ethernet features include the following:

- Supports 10/100 Mbps data transfer rates with external PHY interfaced through RMII
- Full-duplex and half-duplex support for Ethernet
- Supports dual-buffer (ring) or linked-list (chained) descriptor chaining
- Direct interface with the system crossbar bus
- Provides support for CSMA/CD protocol for half-duplex operation
- IEEE 802.3x flow control for full-duplex and half-duplex
- Automatic network monitoring statistics with management counters
- Flexible address filtering options for uni-cast, multi-cast, and broadcast addresses
- Support for promiscuous mode in reception
- Supports IEEE 802.1Q VLAN tag detection
- Supports programmable Inter-frame Gap (IFG)

- Checksum offload engine for checking IPv4 header checksum and TCP/UDP/ICMP checksum encapsulated in IPv4 or IPv6 datagrams
- Station management interface for PHY device configuration and management
- Includes FIFOs for buffering: 256 bytes for transmit FIFO and 128 bytes for receive FIFO
- Automatic CRC and pad generation controllable on a per-frame basis
- CRC replacement, source address field insertion or replacement, and VLAN insertion, replacement, and deletion in transmitted frames with per-frame control

EMAC Functional Description

This section provides information on the function of Ethernet MAC peripheral.

Hardware for the Media Access Control protocol

This function allows applications to support TCP/IP based network communication. At the system end, the module supports direct connection with the system crossbar bus for memory or MMR transactions. It supports RMII (Reduced Media Independent Interface), and SMI (Station Management Interface) for interfacing with the external PHY chip.

Dedicated DMA Controller with independent read/write channels

Performs both data and status transfers between the application and the media independent interfaces. Internal transmit and receive FIFOs are used to buffer and regulate the frames. A dedicated interrupt line connects the EMAC interrupt sources to the System Event Controller (SEC).

MAC Management Counters (MMC) block

An extended set of registers that collect various statistics compliant with IEEE 802.3 definitions regarding the operation of the interface. The registers are updated for each new transmitted or received frame when the condition to update the counter is met. The EMAC provides a set of such counters, along with extended usage control.

PTP (Precision Time Protocol) engine

Provides hardware assistance for the implementation of the IEEE 1588 version 1 and version 2 standards, which allows time synchronization between systems.

ADSP-CM40x EMAC Register List

The Ethernet MAC (EMAC) module provides an Ethernet interface to the processor. The interface is compliant to IEEE Standard 802.3-2005. A set of registers govern EMAC operations. For more information on EMAC functionality, see the EMAC register descriptions.

Table 23-1: ADSP-CM40x EMAC Register List

Name	Description
EMAC_ADDR0_HI	MAC Address 0 High Register
EMAC_ADDR0_LO	MAC Address 0 Low Register
EMAC_DBG	Debug Register
EMAC_DMA_BMMODE	DMA SCB Bus Mode Register
EMAC_DMA_BMSTAT	DMA SCB Status Register
EMAC_DMA_BUSMODE	DMA Bus Mode Register
EMAC_DMA_IEN	DMA Interrupt Enable Register
EMAC_DMA_MISS_FRM	DMA Missed Frame Register
EMAC_DMA_OPMODE	DMA Operation Mode Register
EMAC_DMA_RXBUF_CUR	DMA Rx Buffer Current Register
EMAC_DMA_RXDSC_ADDR	DMA Rx Descriptor List Address Register
EMAC_DMA_RXDSC_CUR	DMA Rx Descriptor Current Register
EMAC_DMA_RXIWDOG	DMA Rx Interrupt Watch Dog Register
EMAC_DMA_RXPOLL	DMA Rx Poll Demand register
EMAC_DMA_STAT	DMA Status Register
EMAC_DMA_TXBUF_CUR	DMA Tx Buffer Current Register
EMAC_DMA_TXDSC_ADDR	DMA Tx Descriptor List Address Register
EMAC_DMA_TXDSC_CUR	DMA Tx Descriptor Current Register
EMAC_DMA_TXPOLL	DMA Tx Poll Demand Register
EMAC_FLOWCTL	FLow Control Register
EMAC_HASHTBL_HI	Hash Table High Register
EMAC_HASHTBL_LO	Hash Table Low Register
EMAC_IMSK	Interrupt Mask Register
EMAC_IPC_RXIMSK	MMC IPC Rx Interrupt Mask Register
EMAC_IPC_RXINT	MMC IPC Rx Interrupt Register
EMAC_ISTAT	Interrupt Status Register
EMAC_MACCFG	MAC Configuration Register
EMAC_MACFRMFILT	MAC Rx Frame Filter Register
EMAC_MMC_CTL	MMC Control Register
EMAC_MMC_RXIMSK	MMC Rx Interrupt Mask Register
EMAC_MMC_RXINT	MMC Rx Interrupt Register

Table 23-1: ADSP-CM40x EMAC Register List (Continued)

Name	Description
EMAC_MMC_TXIMSK	MMC TX Interrupt Mask Register
EMAC_MMC_TXINT	MMC Tx Interrupt Register
EMAC_RX1024TOMAX_GB	Rx 1024- to Max-Byte Frames (Good/Bad) Register
EMAC_RX128TO255_GB	Rx 128- to 255-Byte Frames (Good/Bad) Register
EMAC_RX256TO511_GB	Rx 256- to 511-Byte Frames (Good/Bad) Register
EMAC_RX512TO1023_GB	Rx 512- to 1023-Byte Frames (Good/Bad) Register
EMAC_RX64_GB	Rx 64-Byte Frames (Good/Bad) Register
EMAC_RX65TO127_GB	Rx 65- to 127-Byte Frames (Good/Bad) Register
EMAC_RXALIGN_ERR	Rx alignment Error Register
EMAC_RXBCASTFRM_G	Rx Broadcast Frames (Good) Register
EMAC_RXCRC_ERR	Rx CRC Error Register
EMAC_RXFIFO_OVF	Rx FIFO Overflow Register
EMAC_RXFRMCNT_GB	Rx Frame Count (Good/Bad) Register
EMAC_RXICMP_ERR_FRM	Rx ICMP Error Frames Register
EMAC_RXICMP_ERR_OCT	Rx ICMP Error Octets Register
EMAC_RXICMP_GD_FRM	Rx ICMP Good Frames Register
EMAC_RXICMP_GD_OCT	Rx ICMP Good Octets Register
EMAC_RXIPV4_FRAG_FRM	Rx IPv4 Datagrams Fragmented Frames Register
EMAC_RXIPV4_FRAG_OCT	Rx IPv4 Datagrams Fragmented Octets Register
EMAC_RXIPV4_GD_FRM	Rx IPv4 Datagrams (Good) Register
EMAC_RXIPV4_GD_OCT	Rx IPv4 Datagrams Good Octets Register
EMAC_RXIPV4_HDR_ERR_FRM	Rx IPv4 Datagrams Header Errors Register
EMAC_RXIPV4_HDR_ERR_OCT	Rx IPv4 Datagrams Header Errors Register
EMAC_RXIPV4_NOPAY_FRM	Rx IPv4 Datagrams No Payload Frame Register
EMAC_RXIPV4_NOPAY_OCT	Rx IPv4 Datagrams No Payload Octets Register
EMAC_RXIPV4_UDSBL_FRM	Rx IPv4 UDP Disabled Frames Register
EMAC_RXIPV4_UDSBL_OCT	Rx IPv4 UDP Disabled Octets Register
EMAC_RXIPV6_GD_FRM	Rx IPv6 Datagrams Good Frames Register
EMAC_RXIPV6_GD_OCT	Rx IPv6 Good Octets Register
EMAC_RXIPV6_HDR_ERR_FRM	Rx IPv6 Datagrams Header Error Frames Register
EMAC_RXIPV6_HDR_ERR_OCT	Rx IPv6 Header Errors Register

Table 23-1: ADSP-CM40x EMAC Register List (Continued)

Name	Description
EMAC_RXIPV6_NOPAY_FRM	Rx IPv6 Datagrams No Payload Frames Register
EMAC_RXIPV6_NOPAY_OCT	Rx IPv6 No Payload Octets Register
EMAC_RXJAB_ERR	Rx Jab Error Register
EMAC_RXLEN_ERR	Rx Length Error Register
EMAC_RXMCASTFRM_G	Rx Multicast Frames (Good) Register
EMAC_RXOCTCNT_G	Rx Octet Count (Good) Register
EMAC_RXOCTCNT_GB	Rx Octet Count (Good/Bad) Register
EMAC_RXOORTYPE	Rx Out Of Range Type Register
EMAC_RXOSIZE_G	Rx Oversize (Good) Register
EMAC_RXPAUSEFRM	Rx Pause Frames Register
EMAC_RXRUNT_ERR	Rx Runt Error Register
EMAC_RXTCP_ERR_FRM	Rx TCP Error Frames Register
EMAC_RXTCP_ERR_OCT	Rx TCP Error Octets Register
EMAC_RXTCP_GD_FRM	Rx TCP Good Frames Register
EMAC_RXTCP_GD_OCT	Rx TCP Good Octets Register
EMAC_RXUCASTFRM_G	Rx Unicast Frames (Good) Register
EMAC_RXUDP_ERR_FRM	Rx UDP Error Frames Register
EMAC_RXUDP_ERR_OCT	Rx UDP Error Octets Register
EMAC_RXUDP_GD_FRM	Rx UDP Good Frames Register
EMAC_RXUDP_GD_OCT	Rx UDP Good Octets Register
EMAC_RXUSIZE_G	Rx Undersize (Good) Register
EMAC_RXVLANFRM_GB	Rx VLAN Frames (Good/Bad) Register
EMAC_RXWDOG_ERR	Rx Watch Dog Error Register
EMAC_SMI_ADDR	SMI Address Register
EMAC_SMI_DATA	SMI Data Register
EMAC_TM_ADDEND	Time Stamp Addend Register
EMAC_TM_AUXSTMP_NSEC	Time Stamp Auxiliary TS Nano Seconds Register
EMAC_TM_AUXSTMP_SEC	Time Stamp Auxiliary TM Seconds Register
EMAC_TM_CTL	Time Stamp Control Register
EMAC_TM_HISEC	Time Stamp High Second Register
EMAC_TM_NSEC	Time Stamp Nanoseconds Register

Table 23-1: ADSP-CM40x EMAC Register List (Continued)

Name	Description
EMAC_TM_NSECUPDT	Time Stamp Nanoseconds Update Register
EMAC_TM_NTGTM	Time Stamp Target Time Nanoseconds Register
EMAC_TM_PPSCTL	PPS Control Register
EMAC_TM_PPSINTVL	Time Stamp PPS Interval Register
EMAC_TM_PPSWIDTH	PPS Width Register
EMAC_TM_SEC	Time Stamp Low Seconds Register
EMAC_TM_SECUPDT	Time Stamp Seconds Update Register
EMAC_TM_STMPSTAT	Time Stamp Status Register
EMAC_TM_SUBSEC	Time Stamp Sub Second Increment Register
EMAC_TM_TGTM	Time Stamp Target Time Seconds Register
EMAC_TX1024TOMAX_GB	Tx 1024- to Max-Byte Frames (Good/Bad) Register
EMAC_TX128TO255_GB	Tx 128- to 255-Byte Frames (Good/Bad) Register
EMAC_TX256TO511_GB	Tx 256- to 511-Byte Frames (Good/Bad) Register
EMAC_TX512TO1023_GB	Tx 512- to 1023-Byte Frames (Good/Bad) Register
EMAC_TX64_GB	Tx 64-Byte Frames (Good/Bad) Register
EMAC_TX65TO127_GB	Tx 65- to 127-Byte Frames (Good/Bad) Register
EMAC_TXBCASTFRM_G	Tx Broadcast Frames (Good) Register
EMAC_TXBCASTFRM_GB	Tx Broadcast Frames (Good/Bad) Register
EMAC_TXCARR_ERR	Tx Carrier Error Register
EMAC_TXDEFERRED	Tx Deferred Register
EMAC_TXEXCESSCOL	Tx Excess Collision Register
EMAC_TXEXCESSDEF	Tx Excess Deferral Register
EMAC_TXFRMCNT_G	Tx Frame Count (Good) Register
EMAC_TXFRMCNT_GB	Tx Frame Count (Good/Bad) Register
EMAC_TXLATECOL	Tx Late Collision Register
EMAC_TXMCASTFRM_G	Tx Multicast Frames (Good) Register
EMAC_TXMCASTFRM_GB	Tx Multicast Frames (Good/Bad) Register
EMAC_TXMULTCOL_G	Tx Multiple Collision (Good) Register
EMAC_TXOCTCNT_G	Tx Octet Count (Good) Register
EMAC_TXOCTCNT_GB	Tx OCT Count (Good/Bad) Register
EMAC_TXPAUSEFRM	Tx Pause Frame Register

Table 23-1: ADSP-CM40x EMAC Register List (Continued)

Name	Description
EMAC_TXSNGCOL_G	Tx Single Collision (Good) Register
EMAC_TXUCASTFRM_GB	Tx Unicast Frames (Good/Bad) Register
EMAC_TXUNDR_ERR	Tx Underflow Error Register
EMAC_TXVLANFRM_G	Tx VLAN Frames (Good) Register
EMAC_VLANTAG	VLAN Tag Register

ADSP-CM40x EMAC Interrupt List

Table 23-2: ADSP-CM40x EMAC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
86	EMAC0_STAT	EMAC0 Status		

ADSP-CM40x EMAC Trigger List

Table 23-3: ADSP-CM40x EMAC Trigger List Masters

Trigger ID	Name	Description	Sensitivity
32	EMAC0_STAT	EMAC0 Status	

Table 23-4: ADSP-CM40x EMAC Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

EMAC Definitions

The following definitions are helpful for using the EMAC.

EMAC SCB

System Crossbar Interface of EMAC

EMAC DMA

DMA Controller of EMAC

EMAC MFL

MAC FIFO Layer inside EMAC

EMAC CORE

CORE layer inside EMAC which performs the actual Ethernet operations, including interface with PHY through the reduced media interface(s).

MMC

MAC Management Counter

SMI

Station Management Interface that controls PHY through MDIO and MDC signals.

RMII

Reduced Media Independent Interface

MAC

Media Access Control

PTP

Precision Time Protocol

EMAC Block Diagram and Interfaces

The ***EMAC Simplified Block Diagram*** illustrates the overall functional architecture of the Ethernet MAC peripheral. The EMAC module is comprised of four major layers: EMAC SCB, EMAC DMA, EMAC MFL, and EMAC CORE. Each of these layers (subblocks) is explained in depth in their respective sections in this chapter.

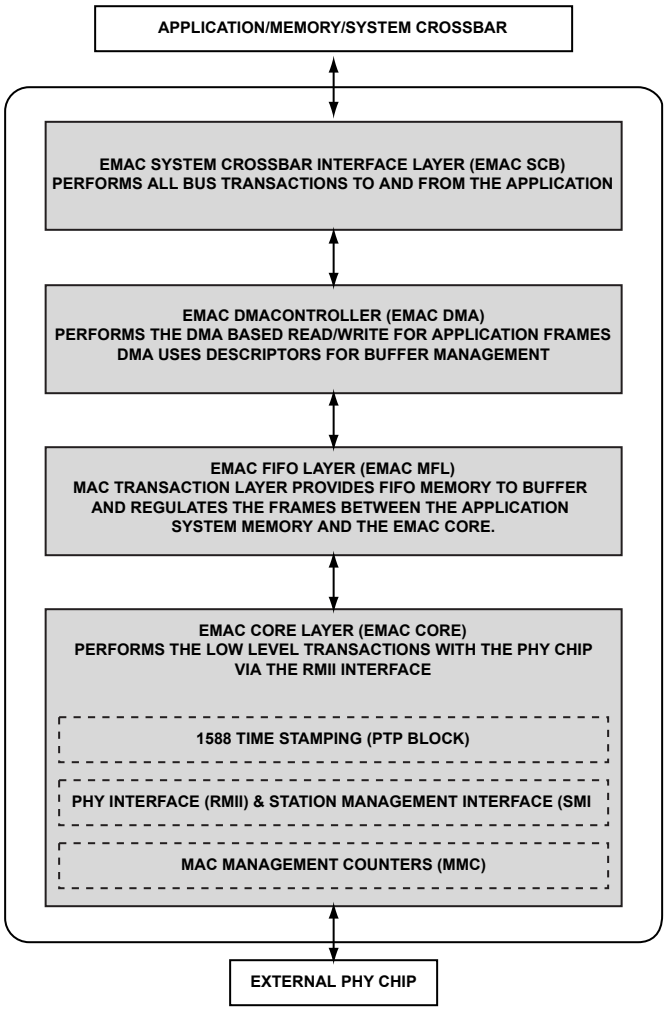

Figure 23-1: EMAC Simplified Block Diagram

A more comprehensive block diagram is shown in the ***EMAC Complete Block Diagram***. It includes most of the important blocks inside the EMAC. The EMAC is connected to processor memory and the system crossbar through the System Crossbar Bus Interface (SCB) and System Peripheral Bus Interface (SPB). These connections are which are part of the SCB layer. The SPB interface is connected to all modules that require MMR programming.

The DMA controller performs application data transfer frame by frame, through well defined descriptor structures. A FIFO layer acts as a buffer between the DMA controller and EMAC core.

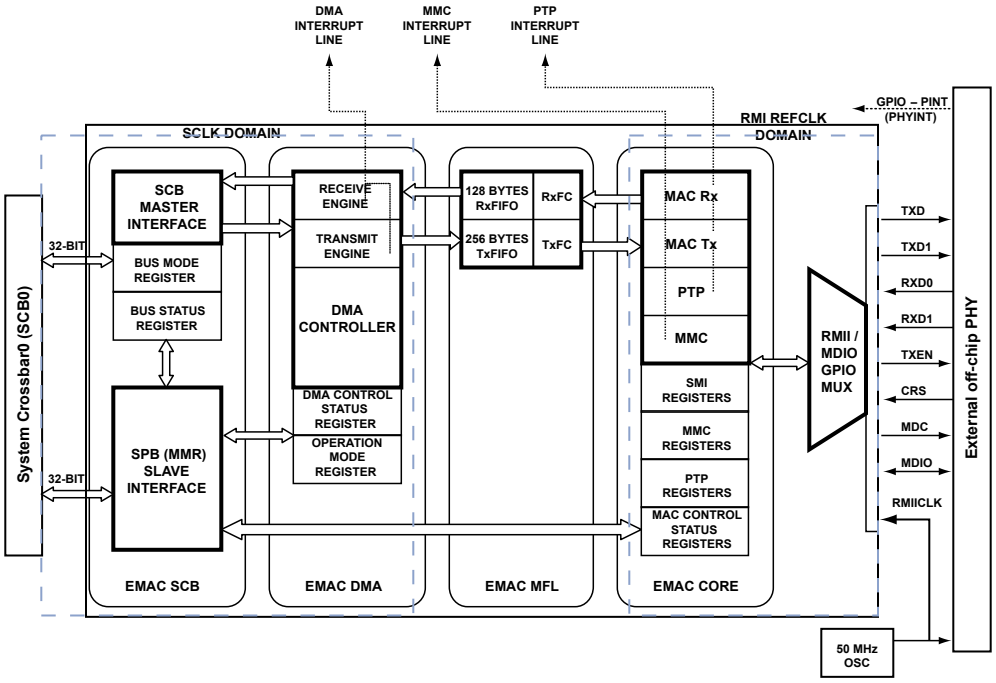

Figure 23-2: EMAC Complete Block Diagram

EMAC CORE Sub-Blocks

The ***Core Transmit Engine Subblocks*** table summarizes the core transmit engine subblocks and their functions. Refer to the EMAC core section for further explanation of each of these subblocks.

Table 23-5: Core Transmit Engine Subblocks

CORE Transmit Engine Sub Block	Function
Transmit Bus Interface	Interface to the FIFO.
Transmit Frame Controller	–Appends Zero-PAD data, if required, for short frames. –Appends CRC for Frame Check-Sum from the CRC generator.
Transmit Protocol Engine	–Generates preamble and SFD, as per 802.3 protocol. –Generates jam pattern in half-duplex mode, for collisions. –Jabber timeout, for excessively large frames. –Flow control for half-duplex mode (back pressure). –Generates transmit frame status.
Transmit Scheduler	–Maintains the inter-frame gap between two transmitted frames. –Follows the Truncated Binary Exponential Back-off algorithm for half-duplex mode.
Transmit CRC Generator	Generate CRC for the Frame Checksum field of the Ethernet frame.
Transmit Flow Control	Receives the pause frame, appends the calculated CRC, and sends the frame to the protocol engine module.

Table 23-5: Core Transmit Engine Subblocks (Continued)

CORE Transmit Engine Sub Block	Function
Transmit Checksum Offload Engine	Supports checksum calculation and insertion in the transmit path, for IPV4/TCP/UDP/ICMP packets.

The ***Core Receive Engine Subblocks*** table summarizes the core receive engine subblocks and their function. Refer to the EMAC core section for more information on each of these subblocks.

Table 23-6: Core Receive Engine Subblocks

CORE Receive Engine sub block	Functionality overview
Receive Protocol Engine	–Strips the incoming preamble and SFD. –Checks for correct length or type field. –Performs internal loopback, if necessary. –Generates receive status. –Supports watchdog of received frames. –Supports jumbo frames.
Receive CRC Module	Checks for CRC error, by comparing with FCS.
Receive Frame Controller Module	–Packs incoming 8-bit input stream to 32-bit data internally. –Performs frame filtering, for uni-cast, multi-cast, and broadcast frames. –Attaches the calculated IP checksum input from Checksum Offload Engine. –Updates the receive status to bus interface.
Receive Flow Control Module	–Detects the receiving pause frame and pauses the frame transmission for the delay specified within the received pause frame. –Works in full duplex mode.
Receive IP Checksum Offload Engine	–Calculates IPv4 header checksums and verify against the received IPv4 header checksums. –Identifies a TCP, UDP, or ICMP payload in the received IP datagrams.
Receive Bus Interface Unit Module	Interface to the FIFO.
Address Filtering Module	Filters destination and source address based on uni-cast, multi-cast, and broadcast frames. –Provides CRC hash filtering.

EMAC PHY Interface

The EMAC can interface to the PHY through the RMII interface standard. The ***RMII Pins*** table shows the RMII pins available in the EMAC in terms of their generic names. Refer to the data sheet for exact pin names.

Table 23-7: RMII Pins

Sl. No.	Generic Signal Name (IEEE Standards)	RMII Pin functionality.
1.	TXD0	RMII transmit data pin D0 (di-bit lower)
2.	TXD1	RMII transmit data pin D1 (di-bit higher)
3.	RXD0	RMII receive data pin D0 (di-bit lower)
4.	RXD1	RMII receive data pin D1 (di-bit higher)
5.	RMII CLK	RMII common clock (for TX and RX), also called reference clock
6.	TXEN	RMII transmit enable pin (TX valid)
7.	CRS	RMII carrier sense / receive data valid
8.	MDC	Serial management clock driven by EMAC
9.	MDIO	Serial management bidirectional data

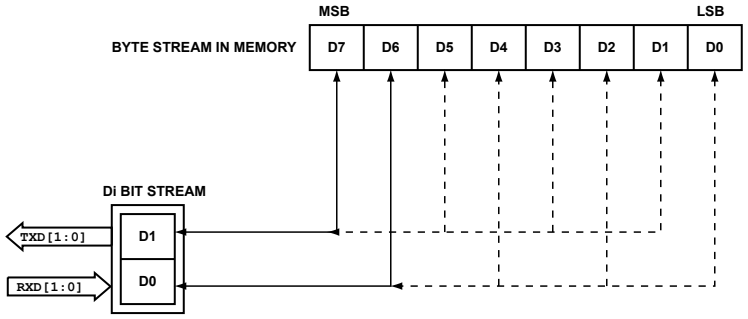

Figure 23-3: RMII Di-bit Data Transfer

Clock Sources

The Ethernet MAC is clocked internally from SYSCLK. Check the processor data sheet for the valid frequency range of the appropriate SYSCLK signal for Ethernet operation.

Source a 50-MHz clock externally to operate the EMAC in RMII mode. This clock is the same for both transmit and receive. The MDC station management clock is derived from the SYSCLK and driven from the MAC to the PHY, when accessing any PHY registers.

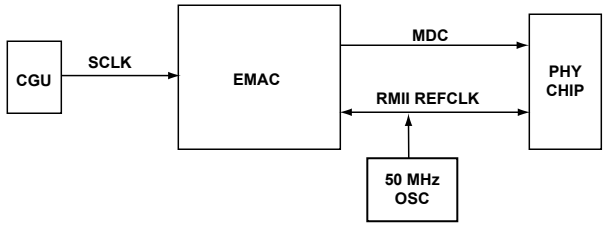

Figure 23-4: EMAC Clock Sources for RMII PHY interface

EMAC Architectural Concepts

This section explains different architectural concepts relevant to EMAC peripheral, such as EMAC SCB, EMAC DMA, EMAC MFL, EMAC CORE, and others.

EMAC System Crossbar Interface (EMAC SCB)

The EMAC SCB bus interface provides the bus connectivity to support highly effective throughput of data traffic. System bus use is maximized by allowing simultaneous read and write transfers initiated from different DMA channels. The EMAC controller connects directly to the SCB0 crossbar. The following interfaces are available with the design.

- A 32-bit SCB master interface for reading and writing to and from the application memory.
- A 32-bit SPB slave interface for register programming.

Refer to the "System Crossbars (SCB)" chapter for more information on how the crossbar operates. This chapter details only the EMAC-specific information.

Table 23-8: EMAC-SCB Interface Data Transfer Specifications with Crossbar

Specification Term	Comments
1 beat in SCB	SINGLE burst
BLEN4 bursts	4 beats in SCB
BLEN8 bursts	8 beats in SCB
BLEN16 bursts	16 beats in SCB
Bus size	32-bit fixed bus size; equals 1 beat
INCR bursts	Incrementing Bursts
INCR ALIGNED bursts	Incrementing aligned bursts
UNDEF bursts	Undefined burst length
PBL	Programmable Burst Length for DMA

The ***EMAC DMA Read/Write channels with System Crossbar*** figure shows DMA write channel and read channel datapaths and their connection to the system crossbar.

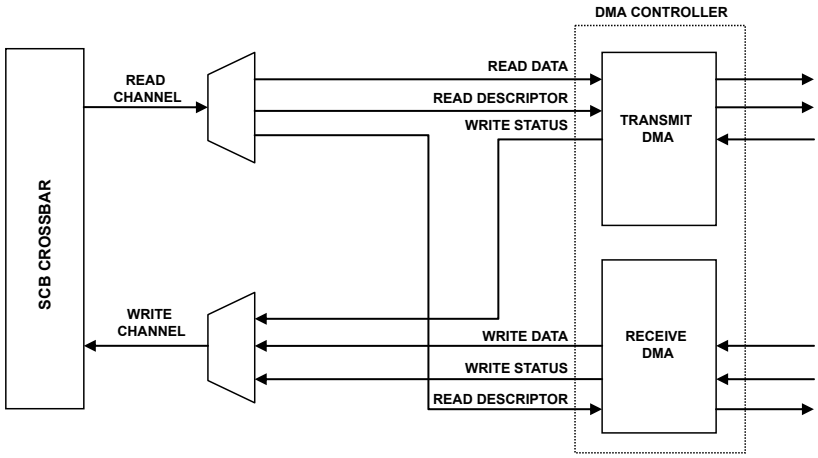

Figure 23-5: EMAC DMA Read/Write channels with System Crossbar

NOTE: Transmit descriptor read and receive descriptor write-back (status update) operations can occur simultaneously. However, transmit descriptor read and write-back operations cannot occur simultaneously. Transmit DMA (or receive DMA) does not initiate the next transfer unless the previous one is complete.

Priority of SCB Requests

The descriptor transfers have higher priority than the data transfers. For example, if there are two bus requests, such as a receive descriptor read and a transmit data read, the receive descriptor read has a higher priority. The next receive data write (subsequent to the receive descriptor read) does not depend on the completion of the transmit data-read transfer.

If there are requests for descriptor reads from both DMA channels, they are serviced based on a first-come first-serve. Receive DMA has higher priority if the descriptor-read requests are generated from both the DMA channels in the same clock cycle. Similarly, in the write channel, descriptor writes from DMA have higher priority than the data-write transfers for the receive DMA.

SCB Interface Programming Options

The SCB bus interface supports the following programmable options for the EMAC module. These options are available using the `EMAC_DMA_BMMODE` register with the `EMAC_DMA_BUSMODE` register.

- ***Outstanding transactions.*** The EMAC-SCB supports up to four outstanding read/write requests on the SCB bus. Software can control these requests by programming the `EMAC_DMA_BMMODE.WROSRLMT` and `EMAC_DMA_BMMODE.RDOSRLMT` bits. Maximum outstanding requests = `EMAC_DMA_BMMODE.WROSRLMT` + 1 (or) `EMAC_DMA_BMMODE.RDOSRLMT` + 1.
- ***Allowed burst sizes.*** The allowed burst sizes are 4 (`EMAC_DMA_BMMODE.BLEN4`), 8 (`EMAC_DMA_BMMODE.BLEN8`), 16 (`EMAC_DMA_BMMODE.BLEN16`) and the SINGLE burst. The EMAC-SCB uses only those burst sizes configured by the program (through the `EMAC_DMA_BMMODE` register) for data transfer through the SCB bus. However, SINGLE burst is available by default, when the

EMAC_DMA_BMMODE.UNDEF bit is cleared. Data transfers are restricted to the maximum burst size from this list of programmed burst sizes.

- ***Burst splitting and burst selection.*** The EMAC-SCB splits the DMA requests into multiple bursts on the SCB system bus. Splitting is based on DMA count and software controllable burst enable bits (shown in the allowed burst sizes) as well as burst types (INCR and INCR_ALIGNED). Burst types are also controllable through the software. SINGLE burst is enabled when the EMAC_DMA_BMMODE.UNDEF bit is not set. Burst length select priority is in the sequence: UNDEF, 16, 8, and 4.
- ***INCR burst type***
  - If the EMAC_DMA_BMMODE.UNDEF bit is set, the EMAC-SCB always chooses the maximum allowed burst length based on the EMAC_DMA_BMMODE.BLEN16, EMAC_DMA_BMMODE.BLEN8, EMAC_DMA_BMMODE.BLEN4 bits. When the DMA requests are not multiples of the maximum allowed burst length, the SCB can choose a burst-length of any value less than the maximum enabled. (All lesser burst-length enables are redundant). For example, when length bits are enabled and the DMA requests a burst of 42 beats, the SCB splits it into three bursts of 16, 16 and 10 beats respectively.
  - If EMAC_DMA_BMMODE.UNDEF is not enabled, then the burst length is based on the priority of the enabled bits in the following order EMAC_DMA_BMMODE.BLEN16, EMAC_DMA_BMMODE.BLEN8, EMAC_DMA_BMMODE.BLEN4. When the DMA requests a burst transfer, the SCB interface splits the requested bursts into multiple transfers using only the enabled burst lengths. This splitting can occur when the requested burst is not a multiple of the maximum enabled burst. If it cannot choose any of the enabled burst lengths, then it selects the burst length as 1.

  For example, the DMA requests a burst transfer of 42 beats, the SCB interface splits it into multiple bursts of size 16, 16, 8, 1 and 1 beats respectively. (In this case, the allowed burst sizes are enabled and the sequence is in decreasing burst sizes).
- ***INCR_ALIGNED burst type.*** When the address-aligned burst-type is enabled (EMAC_DMA_BMMODE.AAL), the SCB interface splits the DMA requested bursts. The "INCR Burst Type" section explains burst splitting conditions further. Each burst-size aligns to the least significant bits of the start address. The SCB interface initially generates smaller bursts so that the remaining transfers move with the maximum (enabled) fixed burst lengths.

  For example, in the same setting as explained earlier for EMAC_DMA_BMMODE.UNDEF set, the DMA requests a burst size of 42 beats at the start address of 0x000003A4.(EMAC_DMA_BMMODE.BLEN16, EMAC_DMA_BMMODE.BLEN8, and EMAC_DMA_BMMODE.BLEN4 are enabled). The SCB starts the first transfer with size 3 such that the address of the next burst is aligned (0x000003B0) for a burst of 16. Therefore, the sequence of bursts is 3, 16, 16, and 7, respectively.

  When EMAC_DMA_BMMODE.UNDEF is not set, then (having a start address of 0x000003A4 with 42 beats), the sequence of burst transfers is 1, 1, 1, 16, 16, 4, and 3 respectively. The sequence of smaller bursts at the beginning is used to align the address to the next higher enabled burst-lengths programmed in the register.
- ***Burst operations for DMA transactions.*** The EMAC_DMA_BUSMODE.PBL (programmable burst length) field indicates the maximum number of beats to transfer in one DMA transaction. This value is also the maximum used in a single block read/write. It is shown in the following table.

- For example, if EMAC_DMA_BUSMODE.PBL=32 and if EMAC_DMA_BMMODE.BLEN16 is enabled, the DMA automatically splits 32 bursts in to 2 x 16 bursts. If EMAC_DMA_BUSMODE.PBL=8, and if EMAC_DMA_BMMODE.BLEN16 and EMAC_DMA_BMMODE.BLEN8 are enabled, the maximum burst is limited to EMAC_DMA_BMMODE.BLEN8. If the EMAC_DMA_BUSMODE.PBL8 bit is set, the programmed EMAC_DMA_BUSMODE.PBL value is multiplied by 8 times internally. However, the result cannot be more than the maximum limits specified.
- Set the EMAC_DMA_BUSMODE.USP bit to make the receive DMA burst length configuration independent of the transmit DMA configuration. When this bit is set, the EMAC uses the EMAC_DMA_BUSMODE.RPBL bits to define the burst length of receive DMA. If the EMAC_DMA_BUSMODE.USP bit is not set, the EMAC_DMA_BUSMODE.RPBL bits are used for both transmit and receive. Programs must ensure that the PBL maximum limit is not violated.
- The receive and transmit descriptors are always accessed in the maximum burst-size for the 16-bytes to be read (PBL-max limit is (TX or RX FIFO size/2)/4 words. (PBL maximum for transmit and receive limits burst-size).

DMA Bursts Using the SCB Interface

The transmit DMA initiates a data transfer when sufficient space to accommodate the configured burst is available in the transmit FIFO. Or, the transmit DMA initiates a data transfer when the number of bytes until the end of frame is less than the configured burst-length. The DMA indicates the start address and the number of transfers required to the SCB master interface. When the SCB interface is configured for fixed-length burst, then it transfers data using the best combination of INCR4/8/16 and 1 beat transaction.

The receive DMA initiates a data transfer when sufficient data to accommodate the configured burst is available in the MTL receive FIFO. Or, the receive DMA initiates a data transfer when the end of frame is detected in the receive FIFO. (For example, when the amount is less than the configured burst-length). The DMA indicates the start address and the number of transfers required to the SCB master interface. When the SCB interface is configured for fixed-length burst, then it transfers data using the best combination of INCR 4, 8, 16 or 1 beat transaction. If the end-of frame is reached before the fixed-burst ends on the SCB interface, then dummy transfers are performed to complete the fixed-burst. Otherwise (if EMAC_DMA_BUSMODE.FB is reset), it transfers data using INCR (undefined length) transactions.

When the SCB interface is configured for address-aligned beats, both DMA engines ensure that the first burst transfer is less than or equal to the configured PBL size. (The address-aligned beats configuration uses the EMAC_DMA_BUSMODE.AAL bit). Therefore, all subsequent beats start at an address that is aligned to the configured PBL.

SCB Bus Transaction Status

The SCB uses the EMAC_DMA_BMSTAT.BUSRD and EMAC_DMA_BMSTAT.BUSWR bits to indicate whether the channel is active or not.

Fatal Bus Error

The EMAC SCB asserts the error interrupt (`EMAC_DMA_STAT.FBI`) when the corresponding fatal bus error interrupt is enabled in the DMA interrupt enable register. The application must reset the core to restart the DMA.

DMA Controller (EMAC DMA)

The EMAC has a built-in DMA controller that performs reads and writes of application data and descriptors through the SCB master interface.

The DMA controller has independent transmit and receive engines, and a CSR (control and status register) space. The transmit engine transfers data from system memory to a FIFO, while the receive engine transfers data from the FIFO to the system memory. The controller uses a descriptor chain-based transfer mechanism to move data efficiently from source to destination with minimal processor core intervention. The DMA is specially designed for packet-oriented data transfers such as Ethernet frames. The controller can be programmed to interrupt the application for situations such as frame transmit and receive transfer completion, and other normal or abnormal conditions.

The DMA and the application device driver communicate through two internal data structures:

1. DMA control and status registers (CSR).
2. Data buffers and descriptor lists. Descriptor lists operate in ring mode and chain mode, as shown in the ***EMAC DMA Descriptor Models*** figure.

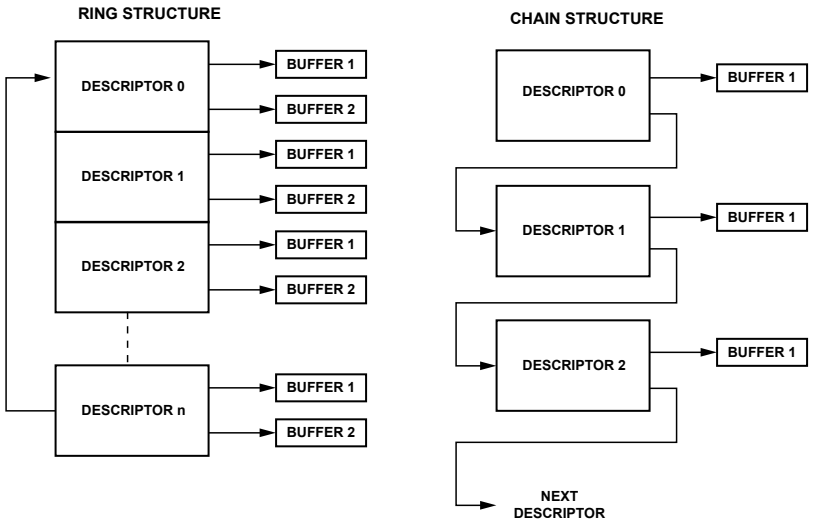

Figure 23-6: EMAC DMA Descriptor Models

Descriptors that reside in the application memory act as pointers to receive and transmit buffers. Descriptors have the following extra attributes.

- There are two descriptor lists, one for receive, and one for transmit. The base address of each list is written into the address registers of the receive and transmit descriptor lists respectively.
- A descriptor list is forward linked (either implicitly or explicitly). The last descriptor can point back to the first entry to create a ring structure.

- Explicit chaining of descriptors is accomplished by setting the second address chained in both receive and transmit descriptors.
- The descriptor lists reside in the address space of the application memory.
- Each descriptor can point to a maximum of two buffers. This attribute enables two buffers, physically addressed, rather than contiguous buffers in memory.

A data buffer resides in the physical memory space of the application. It consists of an entire frame or part of a frame, but cannot exceed a single frame. Buffers can contain only data. The descriptor maintains buffer status. *Data chaining* refers to frames that span multiple data buffers. However, a single descriptor cannot span multiple frames. The DMA skips to the next frame buffer when the end-of-frame is detected. Data chaining is enabled or disabled.

NOTE: It is possible to define a skip length (in terms of N × 32-bit words) between two subsequent descriptors, when using ring mode. Program the `EMAC_DMA_BUSMODE.DSL` field to enable this attribute. With this option available, programs are not always restricted to a contiguous memory location in ring mode.

DMA Related Registers

The ***Summary of DMA Related Registers*** table provides a summary of DMA registers relative to their function. Refer to the "Register Descriptions" sections for complete bit descriptions of each of these registers.

Table 23-9: Summary of DMA Related Registers

Register Name	Description
Bus Mode[*1]	Establishes the bus operating modes for the DMA based on the SCB master interface.
Transmit Poll Demand	Enables the transmit DMA to check whether the DMA owns the current descriptor. The transmit poll demand command wakes up the TxDMA when it is in suspend mode. The TxDMA can go into suspend mode because of an underflow error in a transmitted frame or because of the unavailability of descriptors owned by transmit DMA. Issue this command anytime and the TxDMA resets this command once it starts refetching the current descriptor from host memory.
Receive Poll Demand	Enables the receive DMA to check for new descriptors. This command wakes up the RxDMA from the SUSPEND state. The RxDMA can go into SUSPEND state only because of the unavailability of descriptors owned by it.
Receive Descriptor List Address	Points to the start of the receive descriptor list. The descriptor lists reside in the application memory space and must be word-aligned (32- bit data bus). The DMA internally converts the descriptor list to a bus width aligned address by making the corresponding LSBs low.
Transmit Descriptor List Address	Points to the start of the transmit descriptor list. The descriptor lists reside in the application memory space and must be word-aligned (for 32-bit data bus). The DMA internally converts it to bus width aligned address by making the corresponding LSB to low.
DMA Status	Contains all the status bits that the DMA reports to the application. The software driver reads this register during an interrupt service routine or during polling. Most of the fields in this register interrupt the host.

Table 23-9: Summary of DMA Related Registers (Continued)

Register Name	Description
Operation Mode	Establishes the transmit and receive operating modes and commands. The operation mode register is the last control register written as part of DMA initialization.
Interrupt Enable	Enables the interrupts reported by DMA status register. After a hardware or software reset, all interrupts disable.
Missed Frame and Buffer Overflow Counter	The DMA maintains two counters to track the number of missed frames during reception. This register reports the current value of the counter, which is used for diagnostic purposes.
Receive Interrupt Watchdog Timer	When written with non-zero value, enables the watchdog timer for receive interrupt (RI) in the DMA status register.
SCB Bus Mode	Controls the SCB interface master behavior. It controls the burst splitting and the number of outstanding requests.
SCB Status	Provides the active status of the SCB interface read and write channels.
Current Host Transmit Descriptor	Points to the start address of the current transmit descriptor read by the DMA.
Current Host Receive Descriptor	Points to the start address of the current receive descriptor read by the DMA.
Current Host Transmit Buffer Address	Points to the current transmit buffer address the DMA is reading.
Current Host Receive Buffer Address	Points to the current receive buffer address the DMA is reading.

*1 Do not write to the EMAC_DMA_BUSMODE registers until the first write updates. Otherwise, the second write operation does not update properly. For correct operation, the delay between two writes to the same register location must be at least 8 cycles of 50 MHz RMII REFCLK.

Table 23-10: DMA Registers with Consecutive Writes

Registers with Implications for Consecutive Writes
DMA Bus Mode

DMA Descriptors

The DMA module in the Ethernet subsystem transfers data based on a linked list of descriptors. Align the descriptor addresses to the 32-bit bus width. The descriptors can be either 4 x 32-bit words (16 bytes) or 8 x 32-bit words (32 bytes). Configure the controller for the appropriate word length using the EMAC_DMA_BUSMODE register. Number descriptor words from 0 to 7 for both the transmit and receive engine.

Typical factors for deciding the descriptor word size are as follows.

- When the time-stamping or receive checksum engines are not enabled, the extended descriptors are not required. The software can use descriptors with the default size of 16 bytes (4 words).
- When the time-stamping feature is enabled, the software must allocate 32 bytes (8 words) of memory for every descriptor. (The time-stamping feature is used with the IEEE 1588 PTP engine).

- When only the receive checksum offload is enabled (time-stamping disabled) the software must allocate 32 bytes (8 words) of memory for every descriptor. However, only word 4 of the extended words (descriptors 4–7) contains the required status information. Treat the rest of the extended words as reserved or dummy.

Transmit Descriptor

The ***Transmit Descriptor Words*** figure shows the transmit descriptor structure in memory. The application software must program the TDES0 control bits during descriptor initialization. When the DMA updates the descriptor, it writes back all the control bits except the OWN bit (which it clears) and updates the status bits. The following tables give the contents of the transmitter descriptor word 0 (TDES0) through word 7 (TDES7).

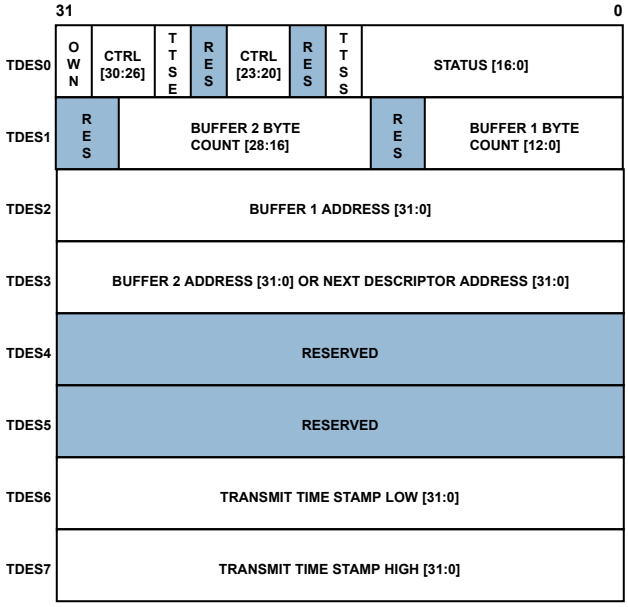

Figure 23-7: Transmit Descriptor Words

Table 23-11: Transmit Descriptor Fields (TDES0)

Bit	Name	Description
31	OWN	When set, this bit indicates that the DMA owns the descriptor. When this bit is reset, it indicates that the application owns the descriptor. The DMA clears this bit either when it completes the frame transmission or when the buffers allocated in the descriptor are read completely. The ownership bit of the first descriptor of the frame must be set after all subsequent descriptors belonging to the same frame have been set. This configuration avoids a possible race condition between fetching a descriptor and the driver setting an ownership bit.
30	IC	Interrupt on Completion. When set, this bit sets the transmit interrupt (DMA status register [0]) after the present frame is transmitted.
29	LS	Last Segment. When set, this bit indicates that the buffer contains the last segment of the frame
28	FS	First Segment. When set, this bit indicates that the buffer contains the first segment of a frame.

Table 23-11: Transmit Descriptor Fields (TDES0) (Continued)

Bit	Name	Description
27	DC	Disable CRC. When this bit is set, the EMAC does not append a cyclic redundancy check (CRC) to the end of the transmitted frame. This functionality is valid only when the first segment (TDES0[28]) is set.
26	DP	Disable Pad. When set, the EMAC does not automatically add padding to a frame shorter than 64 bytes. When this bit is reset, the DMA automatically adds padding and CRC to a frame shorter than 64 bytes. The CRC field is added despite the state of the DC (TDES0[27]) bit. This functionality is valid only when the first segment (TDES0[28]) is set.
25	TTSE	Transmit Time-Stamp Enable. When set, this bit enables IEEE1588 hardware time-stamping for the transmit frame referenced by the descriptor. This field is valid only when the first segment control bit (TDES0[28]) is set.
24	Reserved	
23:22	CIC	Checksum Insertion Control. These bits control the checksum calculation and insertion. Bit encodings are as follows. 00 = Checksum Insertion Disabled. 01 = Only IP header checksum calculation and insertion are enabled. 10 = IP header checksum and payload checksum calculation and insertion are enabled, but pseudo-header checksum is not calculated in hardware. 11 = IP Header checksum and payload checksum calculation and insertion are enabled, and pseudo-header checksum is calculated in hardware.
21	TER	Transmit End of Ring. When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a descriptor ring.
20	TCH	Second Address Chained. When set, this bit indicates that the second address in the descriptor is the next descriptor address rather than the second buffer address. When TDES0[20] bit is set, TBS2 (TDES1[28:16]) is a ***do-not-care*** value. TDES0[21] takes precedence over TDES0[20].
19:18	Reserved	
17	TTSS	Transmit Time Stamp Status. This bit is used as a status bit to indicate that a time-stamp was captured for the described transmit frame. When this bit is set, TDES2 and TDES3 have a time-stamp value captured for the transmit frame. This field is only valid when the last segment control bit of the descriptor (TDES0[29]) is set.
16	IHE	IP Header Error. When set, this bit indicates that the EMAC transmitter detected an error in the IP datagram header. The transmitter checks the header length in the IPv4 packet against the number of header bytes received from the application. It indicates an error status when there is a mismatch. For IPv6 frames, a header error is reported if the main header length is not 40 bytes. Furthermore, the Ethernet length or type field value for an IPv4 or IPv6 frame must match the IP header version received with the packet. For IPv4 frames, an error status is also indicated if the header length field has a value less than 0x5.
15	ES	Error Summary. Indicates the logical OR of the following bits. TDES0[14] = Jabber Timeout TDES0[13] = Frame Flush

Table 23-11: Transmit Descriptor Fields (TDES0) (Continued)

Bit	Name	Description
		TDES0[11] = Loss of Carrier TDES0[10] = No Carrier TDES0[9] = Late Collision TDES0[8] = Excessive Collision TDES0[2] = Excessive Deferral TDES0[1] = Underflow Error TDES0[16] = IP Header Error TDES0[12] = IP Payload Error
14	JT	Jabber Timeout. When set, this bit indicates that the EMAC transmitter has experienced a jabber timeout. This bit is only set when the JD bit of the EMAC configuration register is not set.
13	FF	Frame Flushed. When set, this bit indicates that the DMA or MFL flushed the frame due to a software flush command given by the CPU.
12	IPE	IP Payload Error. When set, this bit indicates that EMAC transmitter detected an error in the TCP, UDP, or ICMP IP datagram payload. The transmitter checks the payload length received in the IPv4 or IPv6 header against the actual number of TCP, UDP, or ICMP packet bytes received from the application. It issues an error status when there is a mismatch.
11	LC	Loss of Carrier. When set, this bit indicates that a loss of carrier occurred during frame transmission. This functionality is valid only for the frames transmitted without collision when the EMAC operates in half-duplex mode.
10	NC	No Carrier. When set, this bit indicates that the carrier sense signal from the PHY did not assert during transmission.
9	LC	Late Collision. When set, this bit indicates that frame transmission was aborted due to a collision occurring after the collision window (64 byte-times, including preamble). This bit is not valid if the underflow error bit is set.
8	EC	Excessive Collision. When set, this bit indicates that the transmission was aborted after 16 successive collisions while attempting to transmit the current frame. If the DR (disable retry) bit in the EMAC configuration register is set, this bit is set after the first collision, and the transmission of the frame aborts.
7	VF	VLAN Frame. When set, this bit indicates that the transmitted frame was a VLAN-type frame.
6:3	CC	Collision Count. This 4-bit counter-value indicates the number of collisions occurring before the frame was transmitted. The count is not valid when the excessive collisions bit (TDES0[8]) is set.
2	ED	Excessive Deferral. When set, this bit indicates that the transmission has ended because of excessive deferral when the deferral check (DC) bit in the EMAC control register is set high. Excessive deferral is over 24,288-bit times.
1	UF	Underflow Error. When set, this bit indicates that the EMAC aborted the frame because data arrived late from the application memory. Underflow error indicates that the DMA encountered an empty transmit buffer while transmitting the frame. The transmission process enters the suspended state and sets both Transmit Underflow (`EMAC_DMA_STAT.UNF`) and Transmit Interrupt (`EMAC_DMA_STAT.TI`).

Table 23-11: Transmit Descriptor Fields (TDES0) (Continued)

Bit	Name	Description
0	DB	Deferred Bit. When set, this bit indicates that the EMAC defers before transmission because of the presence of carrier. This bit is valid only in half-duplex mode.

Table 23-12: Transmit Descriptor Word 1 (TDES1)

Bit	Name	Description
31-29	Reserved	
28–16	TBS2	Transmit Buffer 2 Size. These bits indicate the second data buffer size in bytes. This field is not valid if TDES0[20] is set.
15–13	Reserved	
12–0	TBS1	Transmit Buffer 1 Size. These bits indicate the first data buffer byte size, in bytes. If this field is 0, the DMA ignores this buffer and uses buffer 2 or the next descriptor, depending on the value of TCH (TDES0[20]).

Table 23-13: Transmit Descriptor 2 (TDES2)

Bit	Name	Description
31–0	Buffer 1 Address Pointer	These bits indicate the physical address of buffer 1. There is no limitation on the buffer address alignment

Table 23-14: Transmit Descriptor 3 (TDES3)

Bit	Name	Description
31–0	Buffer 2 Address Pointer (Next Descriptor Address)	Indicates the physical address of buffer 2 when DMA uses a descriptor ring structure. If the second address chained (TDES1[24]) bit is set, this address contains the pointer to the physical memory where the next descriptor is present. The buffer address pointer must align to the bus width only when TDES1[24] is set. (LSBs are ignored internally.)

Table 23-15: Transmit Descriptor 6 (TDES6)

Bit	Name	Description
31–0	TTSL	Transmit Frame Time Stamp Low. The DMA updates this field with the least significant 32 bits of the time-stamp captured for the corresponding transmit frame. This field has the time-stamp only if the last segment bit (LS) in the descriptor is set and time-stamp status (TTSS) bit is set.

Table 23-16: Transmit Descriptor 7 (TDES7)

Bit	Name	Description
31–0	TTSH	Transmit Frame Time Stamp High. The DMA updates this field with the most significant 32 bits of the time-stamp captured for the corresponding receive frame. This field has the time-stamp only if the last segment bit (LS) in the descriptor is set and time-stamp status (TTSS) bit is set.

DMA Transmit Process

The following sections describe how the transmission process works for direct memory access on the EMAC controller.

- Default (Non-OSF) Mode
- OSF Mode Enabled
- Transmit Frame Processing
- Transmit Polling Suspended

Default (Non-OSF) Mode

The default process for DMA transmit works as follows:

1. The application sets up the transmit descriptor (using TDES0- TDES3) and sets the OWN bit (TDES0) after setting up the corresponding data buffers with Ethernet frame data.
2. Once the EMAC_DMA_OPMODE.ST bit is set, the DMA enters the run state.
3. While in the run state, the DMA polls the transmit descriptor list for frames requiring transmission. After polling starts, it continues in either sequential descriptor ring order or chained order. If the DMA detects a descriptor flagged as owned by the application, or if an error condition occurs, transmission suspends. Both the transmit buffer unavailable (EMAC_DMA_STAT.TU) and normal interrupt summary (EMAC_DMA_STAT.NIS) bits are set. The transmit engine proceeds to Step 9.
4. If the acquired descriptor is flagged as owned by DMA (TDES0 [31] = 1#b1), the DMA decodes the transmit data buffer address from the acquired descriptor.
5. The DMA fetches the transmit data from the application memory and transfers the data to the MFL for transmission.
6. If an Ethernet frame is stored over data buffers in multiple descriptors, the DMA closes the intermediate descriptor and fetches the next descriptor. Steps 3, 4, and 5 repeat until the end-of-Ethernet-frame data transfers to the MFL.
7. Frame transmission completes. If IEEE 1588 time-stamping was enabled for the frame, the time-stamp value obtained from MFL is written to the transmit descriptor (TDES2 and TDES3) that contains the end-of-frame buffer. (The transmit status indicates if IEEE 1588 time-stamping enables). The status information is then written to this transmit descriptor (TDES0). Because the OWN bit is cleared during this step, the application now owns this descriptor. If time-stamping was not enabled for this frame, the DMA does not alter the contents of TDES2 and TDES3.
8. Transmit interrupt (EMAC_DMA_STAT.TI) is set after completing transmission of a frame. The frame has interrupt on completion (TDES1 [31]) set in its last descriptor. The DMA engine then returns to Step 3.
9. In the suspend state, the DMA tries to reacquire the descriptor (and returns to Step 3) when it receives a transmit poll demand and the EMAC_DMA_STAT.UNF bit is cleared.

NOTE: If the EMAC_DMA_OPMODE.OSF bit is not set, the actual inter frame gap (IFG) is more than the value programmed in the EMAC_MACCFG register.

OSF Mode Enabled

While in the run state, the transmit process can simultaneously acquire two frames without closing the status descriptor of the first (if the EMAC_DMA_OPMODE.OSF bit is set). As the transmit process finishes transferring the first frame, it immediately polls the transmit descriptor list for the second frame. If the second frame is valid, the transmit process transfers this frame before writing the status information of the first frame.

In OSF mode, the run state transmit DMA operates in the following sequence.

1. The DMA operates as described in steps 1–6 of Default (Non-OSF) Mode.
2. Without closing the previous last descriptor of the frame, the DMA fetches the next descriptor.
3. If the DMA owns the acquired descriptor, the DMA decodes the transmit buffer address in this descriptor. If the DMA does not own the descriptor, the DMA goes into suspend mode and skips to Step 7.
4. The DMA fetches the transmit frame from the application memory and transfers the frame to the MFL until the end-of-frame data is transferred. It closes the intermediate descriptors when this frame splits across multiple descriptors.
5. The DMA waits for the frame transmission status and time-stamp of the previous frame. Once the status is available, the DMA writes the time-stamp to TDES2 and TDES3, if the time-stamp was captured (as indicated by a status bit). The DMA then writes the status, with a cleared OWN bit, to the corresponding TDES0, thus closing the descriptor. If time-stamping was not enabled for the previous frame, the DMA does not alter the contents of TDES2 and TDES3.
6. If enabled, the transmit interrupt is set; the DMA fetches the next descriptor, and then proceeds to Step 3 (when status is normal). If the previous transmission status shows an underflow error, the DMA goes into suspend mode (Step 7).
7. In suspend mode, if a pending status and time-stamp are received from the MFL, the DMA writes the time-stamp (if enabled for the current frame) to TDES2 and TDES3. The DMA then writes the status to the corresponding TDES0. It then sets relevant interrupts and returns to suspend mode.
8. After receiving a transmit poll demand (EMAC_DMA_TXPOLL), the DMA can exit suspend mode and enter the run state. (Go to Step 1 or Step 2 depending on pending status)

NOTE: If the EMAC_DMA_OPMODE.OSF bit is set, the DMA fetches the next descriptor in advance of closing the current descriptor. Therefore, the descriptor chain must have more than two different descriptors for proper operation.

NOTE: If the EMAC_DMA_OPMODE.OSF bit is set, the DMA starts fetching the second frame immediately after completing the transfer of the first frame to the FIFO. It does not wait for the status to update. In the meantime, the MFL receives the second frame into the FIFO while transmitting the first frame. The difference in cycles is not seen for the first descriptor, because the time taken for the complete descriptor

processing remains the same whether EMAC_DMA_OPMODE.OSF is set or not. The difference appears only for the following descriptor because its processing began earlier.

Transmit Frame Processing

The transmit DMA engine expects that the data buffers contain complete Ethernet frames, excluding: preamble, pad bytes, and FCS fields. The destination address, source address, and type or length fields contain valid data. If the transmit descriptor indicates that the EMAC CORE must disable CRC or PAD insertion, the buffer must have complete Ethernet frames (excluding preamble), including the CRC bytes.

Frames can be data-chained and can span several buffers. Frames must be delimited by the first descriptor (TDES0[28]) and the last descriptor (TDES0[29]), respectively.

As transmission starts, the first descriptor must have (TDES0[28]) set. Frame data transfers from the application buffer to the transmit FIFO. Concurrently, if last descriptor (TDES0[29]) of the current frame clears, the transmit process attempts to acquire the next descriptor. The transmit process expects this descriptor to have TDES0[28] clear. If TDES1[29] is clear, it indicates an intermediary buffer. If TDES0[29] is set, it indicates the last buffer of the frame.

After the last buffer of the frame has been transmitted, the DMA writes back the final status information. The DMA writes to the transmit descriptor 0 (TDES0) word of the descriptor that has the last segment set in transmit descriptor 0 (TDES0[29]). Now, if interrupt-on-completion (TDES0[30]) is set, the transmit interrupt (DMA_STAT [0]) is set, the next descriptor is fetched, and the process repeats.

Actual frame transmission begins after the MFL transmit FIFO has reached either a programmable transmit threshold (EMAC_DMA_OPMODE.TTC), or a full frame is contained in the FIFO. There is also an option for store-and-forward mode (EMAC_DMA_OPMODE.TSF). Descriptors are released (OWN bit TDES0 [31] clears) when the DMA finishes transferring the frame.

Transmit Polling Suspended

Either of the following conditions suspends transmit polling:

1. The DMA detects a descriptor owned by the application (TDES0 [31] = 0).
2. A frame transmission aborts when a transmit error due to underflow is detected. The appropriate transmit descriptor 0 (TDES0) bit is set.

If the second condition occurs, both of the abnormal interrupt summary ([15]) and transmit underflow bits ([5]) are set. The information is written to transmit descriptor 0, causing the suspension. If the DMA goes into a SUSPEND state due to the first condition, then both EMAC_DMA_STAT.NIS and EMAC_DMA_STAT.TU are set.

In both cases, the position in the transmit list is retained. The retained position is that of the descriptor following the last descriptor closed by the DMA.

The driver must explicitly issue a transmit poll demand command after rectifying the suspension cause. If the first condition occurs, the driver must give descriptor ownership to the DMA and then issue a poll demand command to resume the transfer.

Receive Descriptor

The ***Receive Descriptor Words*** figure shows the structure of the receive descriptor. It can have 32 bytes of descriptor data (8 DWORDs) when advanced time-stamping or checksum is enabled.

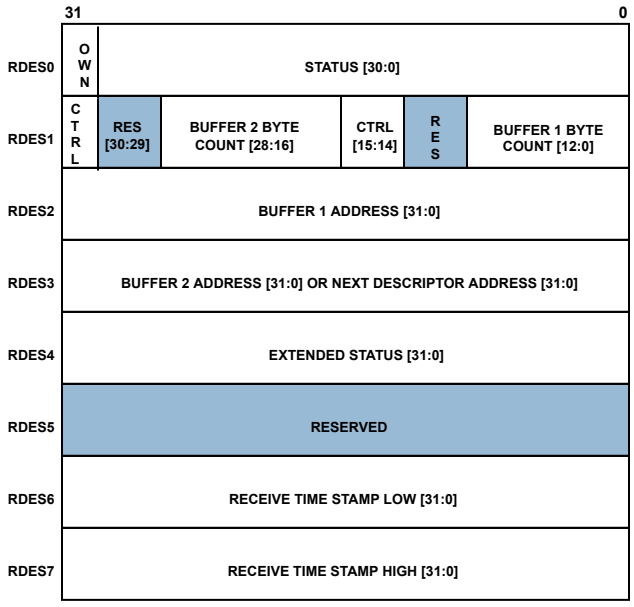

Figure 23-8: Receive Descriptor Words

Table 23-17: Receive Descriptor Fields (RDES0)

Bit	Name	Description
31	OWN	Own. When set, this bit indicates that the DMA of the EMAC subsystem owns the descriptor. When this bit is reset, this bit indicates that the application owns the descriptor. The DMA clears this bit either when it completes the frame reception or when the buffers that are associated with this descriptor are full.
30	AFM	Destination Address Filter Fail. When set, this bit indicates a frame that failed in the DA filter in the EMAC CORE.
29–16	FL	Frame Length. These bits indicate the byte length of the received frame that transferred to application memory (including CRC). This field is valid when last descriptor (RDES0[8]) is set and either the descriptor error (RDES0[14]) or overflow error bits are reset. This field is valid when last descriptor (RDES0[8]) is set. When the last descriptor and error summary bits are not set, this field indicates the accumulated number of bytes that transferred for the current frame.
15	ES	Error Summary. Indicates the logical OR of the following bits. RDES0[1] = CRC Error RDES0[4] = Watchdog Timeout RDES0[6] = Late Collision RDES0[7] = time-stamp Available RDES4[4:3] = IP Header/Payload Error RDES0[11] = Overflow Error

Table 23-17: Receive Descriptor Fields (RDES0) (Continued)

Bit	Name	Description
		RDES0[14] = Descriptor Error. This field is valid only when the last descriptor (RDES0[8]) is set.
14	DE	Descriptor Error. When set, this bit indicates a frame truncation caused by a frame that does not fit within the current descriptor buffers. The DMA does not own the next descriptor. The frame is truncated. This field is valid only when the last descriptor (RDES0[8]) is set.
13	Reserved	
12	LE	Length Error. When set, this bit indicates that the actual length of the frame received and that the length or type field does not match. This bit is valid only when the frame type (RDES0[5]) bit is reset.
11	OE	Overflow Error. When set, this bit indicates that the received frame is damaged due to buffer overflow in MFL.
10	VLAN	VLAN Tag. When set, this bit indicates that the frame pointed to by this descriptor is a VLAN frame tagged by the EMAC CORE.
9	FS	First Descriptor. When set, this bit indicates that this descriptor contains the first buffer of the frame. If the size of the first buffer is 0, the second buffer contains the beginning of the frame. If the size of the second buffer is also 0, the next descriptor contains the beginning of the frame.
8	LS	Last Descriptor. When set, this bit indicates that the buffers pointed to by this descriptor are the last buffers of the frame
7	Timestamp Available	When set, this bit indicates that a snapshot of the timestamp is written in descriptor words 6 (RDES6) and 7 (RDES7). This functionality is valid only when the last descriptor bit (RDES0[8]) is set
6	LC	Late Collision. When set, this bit indicates that a late collision has occurred while receiving the frame in half-duplex mode.
5	FT	Frame Type. When set, this bit indicates that the receive frame is an Ethernet-type frame (the LT field is greater than or equal to 16'h0600). When this bit is reset, it indicates that the received frame is an IEEE802.3 frame. This bit is not valid for runt frames less than 14 bytes.
4	RWT	Receive Watchdog Timeout. When set, this bit indicates that the receive watchdog timer has expired while receiving the current frame. The current frame is truncated after the watchdog timeout.
3	Reserved	
2	DE	Dribble Bit Error. When set, this bit indicates that the received frame has a non-integer multiple of bytes (odd nibbles).
1	CE	CRC Error. When set, this bit indicates that a Cyclic Redundancy Check (CRC) Error occurred on the received frame. This field is valid only when the last descriptor (RDES0[8]) is set.
0	Extended Status Available	When set, this bit indicates that the extended status is available in descriptor word 4 (RDES4). This functionality is valid only when the last descriptor bit (RDES0[8]) is set.

Table 23-18: Receive Descriptor Fields 1 (RDES1)

Bit	Name	Description
31	DIC	Disable Interrupt on Completion. When set, this bit prevents setting the `EMAC_DMA_STAT.RI` bit of the status register for the received frame ending in the buffer indicated by this descriptor. This activity, in turn, disables the assertion of the interrupt to the application due to RI for that frame.
30–29	Reserved	
28–16	RBS2	Receive Buffer 2 Size. These bits indicate the second data buffer size, in bytes. The buffer size must be a multiple of 4 (32-bit bus), even if the value of RDES3 (buffer2 address pointer) does not align to bus width. If the buffer size is not an appropriate multiple of 4, 8, or 16, the resulting behavior is undefined. This field is not valid if RDES1[14] is set.
15	RER	Receive End of Ring. When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a descriptor ring.
14	RCH	Second Address Chained. When set, this bit indicates that the second address in the descriptor is the next descriptor address rather than the second buffer address. When this bit is set, RBS2 (RDES1[28:16]) is a *do-not-care* value. RDES1[15] takes precedence over RDES1[14].
13	Reserved	
12–0	RBS1	Receive Buffer 1 Size. Indicates the size of the first data buffer in bytes. The buffer size must be a multiple of 4 (32-bit bus), even if the value of RDES2 (buffer1 address pointer) is not aligned. When the buffer size is not a multiple of 4, the resulting behavior is undefined. If this field is 0, the DMA ignores this buffer and uses buffer 2 or next descriptor depending on the value of RCH (Bit 14).

Table 23-19: Receive Descriptor Fields 2 (RDES2)

Bit	Name	Description
31–0	Buffer 1 Address Pointer	These bits indicate the physical address of buffer 1. There are no limitations on the buffer address alignment except for the following condition: The DMA uses the configured value for its address generation when the RDES2 value stores the start of frame. The DMA performs a write operation with the RDES2[1:0] bits as 0 during the transfer of the start of frame. However, the frame data shifts per the address pointer of the actual buffer. The DMA ignores RDES2[1:0] when the address pointer is to a buffer where the middle or last part of the frame is stored. (RDES2[1:0] corresponds to a bus width of 32).

Table 23-20: Receive Descriptor Fields 3 (RDES3)

Bit	Name	Description
31–0	Buffer 2 Address Pointer (Next Descriptor Address)	These bits indicate the physical address of buffer 2 when DMA uses a descriptor ring structure. If the second address chained (RDES1[24]) bit is set, this address contains the pointer to the physical memory where the next descriptor is present. If RDES1[24] is set, the buffer (next descriptor) address pointer must be bus width-aligned (RDES3[1:0] = 0, corresponding to a bus width of 32. LSBs are ignored internally.) However, when RDES1[24] is reset, there are no limitations on the RDES3 value, except for the following condition: The DMA uses the configured value for its buffer address generation when the RDES3 value stores the start of frame. The DMA ignores RDES3[1:0] when the address pointer is to a buffer where the middle or last part of the frame is stored. (RDES3[1:0] corresponds to a bus width of 32)

Table 23-21: Receive Descriptor Fields 4 (RDES4)

Bit	Name	Description
31-26	Reserved	
23-21	Reserved	
20-18	VLAN Tag Priority	These bits give the VLAN tag's user value in the received packet. These bits are valid only when the RDES4[16] and RDES4[17] are set. **NOTE:** This bit field is only available with the EMAC extended register set. Otherwise this bit is Reserved.
15	Reserved	
14	Timestamp Dropped	When set, this bit indicates that the timestamp was captured for this frame but got dropped in the MFL RxFIFO because of overflow. **NOTE:** This bit is only available with the EMAC extended register set. Otherwise this bit is Reserved.
13	PTP Version	When set, this bit indicates that the received PTP message has the IEEE 1588 version 2 format. When reset, it has the version 1 format. This description is valid only if the (RDES4[11:8]) is non-zero.
12	PTP Frame Type	When set, this bit indicates that the PTP message transfers directly over Ethernet. When this bit is not set and the message type is non-zero, it indicates that the PTP message transfers over UDP-IPv4 or UDP-IPv6. Bits 6 and 7 have the information on IPv4 or IPv6.
11–8	Message Type	These bits are encoded to give the type of the message received. 0000 = No PTP message received 0001 = SYNC (all clock types) 0010 = Follow_Up (all clock types) 0011 = Delay_Req (all clock types) 0100 = Delay_Resp (all clock types) 0101 = Pdelay_Req (in peer-to-peer transparent clock) or Announce (in ordinary or boundary clock) 0110 = Pdelay_Resp (in peer-to-peer transparent clock) or Management (in ordinary or boundary clock) 0111 = Pdelay_Resp_Follow_Up (in peer-to-peer transparent clock) or Signaling (for ordinary or boundary clock) 1xxx - Reserved
7	IPv6 Packet Received	When set, this bit indicates that the received packet is an IPv6 packet.
6	IPv4 Packet Received	When set, this bit indicates that the received packet is an IPv4 packet.
5	IP Checksum Bypassed	When set, this bit indicates that the checksum offload engine is bypassed.
4	IP Payload Error	When set, this bit indicates that the 16-bit IP payload checksum that the core calculated does not match the corresponding checksum field in the received segment. (For example: the TCP, UDP, or

Table 23-21: Receive Descriptor Fields 4 (RDES4) (Continued)

Bit	Name	Description
		ICMP checksum). The bit is also set when the TCP, UDP, or ICMP segment length does not match the payload length value in the IP header field.
3	IP Header Error	When set, this bit indicates that the 16-bit IPv4 header checksum calculated by the core does not match the received checksum bytes. Or, it indicates that the IP datagram version is not consistent with the Ethernet type value.
2–0	IP Payload Type	These bits indicate the type of payload encapsulated in the IP datagram processed by the receive checksum offload engine (COE). The COE also sets these bits to 2'b00 when it does not process the payload of the IP datagram. It does not process the payload because of an IP header error or fragmented IP. 000 = Unknown or did not process IP payload 001 = UDP 010 = TCP 011 = ICMP 1xx = Reserved

Table 23-22: Extended Receive Descriptor Fields 4 (RDES4)

Bit	Name	Description
31-26	Reserved	
23-21	Reserved	
20-18	VLAN Tag Priority	These bits give the VLAN tag's user value in the received packet. These bits are valid only when the RDES4[16] and RDES4[17] are set.
17	AV Tagged Packet Received	When set, this bit indicates that an AV tagged packet is received. Otherwise, this bit indicates that an untagged AV packet is received. This bit is valid when RDES4[16] is set.
16	AV Packet Received	When set, this bit indicates that an AV packet is received.
15	Reserved	
14	Timestamp Dropped	When set, this bit indicates that the timestamp was captured for this frame but got dropped in the MFL RxFIFO because of overflow.

Table 23-23: Receive Descriptor Fields 6 (RDES6)

Bit	Name	Description
31–0	RTSL	Receive Frame Time-Stamp Low. The DMA updates this field with the least significant 32 bits of the time-stamp captured for the corresponding receive frame. The DMA updates this field only for the last descriptor of the receive frame. The status bit (RDES0[8]) indicates the last descriptor.

Table 23-24: Receive Descriptor Fields 7 (RDES7)

Bit	Name	Description
31–0	RTSH	Receive Frame Time-Stamp High. The DMA updates this field with the most significant 32 bits of the time-stamp captured for the corresponding receive frame. The DMA updates this field only for the last descriptor of the receive frame. The status bit (RDES0[8]) indicates the last descriptor.

EMAC DMA Receive Process

The following sections describe how the receive process for direct memory access works on the EMAC controller.

- Receive Frame Processing
- Receive Descriptor Acquisition
- Receive Process Suspended

The reception process for DMA works as follows:

1. The application sets up receive descriptors (RDES0–RDES3) and sets the `OWN` bit (RDES0 [31]).
2. Once the `EMAC_DMA_OPMODE.SR` bit is set, the DMA enters the run state. While in the run state, the DMA attempts to acquire free descriptors by polling the receive descriptor list. If the fetched descriptor is not free (the application owns the descriptor), the DMA enters the suspend state and jumps to Step 9.
3. The DMA decodes the receive data buffer address from the acquired descriptors.
4. Incoming frames are processed and placed in the data buffers of the acquired descriptor.
5. When the buffer is full or the frame transfer is complete, the receive engine fetches the next descriptor.
6. If the current frame transfer is complete, the DMA proceeds to Step 7. If IEEE 1588 time-stamping is enabled, the DMA writes the time-stamp (if available) to the current descriptor. If the DMA does not own the next fetched descriptor and the frame transfer is not complete, the DMA sets the descriptor error bit in the RDES0. (The bit is set unless flushing is disabled). The DMA closes the current descriptor (clears the `OWN` bit). The DMA marks it as intermediate by clearing the last segment (`LS`) bit in the RDES0 value (marks it as last descriptor if flushing is not disabled). The DMA then proceeds to Step 8. If the DMA owns the next descriptor but the current frame transfer is not complete, the DMA closes the current descriptor as intermediate and reverts to Step 4.
7. If IEEE 1588 time-stamping is enabled, the DMA writes the time-stamp (if available) to RDES2 and RDES3 of the current descriptor. The DMA then takes the status of the receive frame from the MFL and writes the status word to RDES0 of the current descriptor. The `OWN` bit is cleared and the last segment bit is set.
8. The receive engine checks `OWN` bit of the latest descriptor. If the host owns the descriptor (`OWN` bit is 0), the `EMAC_DMA_STAT.RU` bit is set. The DMA receive engine enters the suspended state (Step 9). If the DMA owns the descriptor, the engine returns to Step 4 and awaits the next frame.
9. Before the receive engine enters the suspend state, partial frames are flushed from the receive FIFO (programs control flushing using the `EMAC_DMA_OPMODE.DFF` bit).

10. The receive DMA exits the suspend state when a receive poll demand is given or the start of next frame is available from the receive FIFO of the MFL. The engine proceeds to Step 2 and refetches the next descriptor.

Receive Frame Processing

The EMAC transfers the received frames to the application memory only when:

- the frame passes the address filter subblock, and
- the frame size is greater than or equal to configurable threshold bytes set for the receive FIFO of MFL, or
- the complete frame is written to the FIFO in store-and-forward mode.

If the frame fails the address filtering, the EMAC block drops the frame (unless the `EMAC_MACFRMFILT.RA` bit is set). Frames that are shorter than 64 bytes, because of collision or premature termination, can be purged from the receive FIFO.

After receiving 64 bytes (configurable threshold), the MFL block requests that the DMA block begins transferring the frame data to the receive buffer pointed to by the current descriptor. The DMA sets first descriptor (RDES0 [9]) after the SCB becomes ready to receive the data (if DMA is not fetching transmit data from the application). The descriptors release when the `OWN` (RDES [31]) bit is reset to 0. The bit is reset either as the data buffer fills up or as the last segment of the frame is transferred to the receive buffer. If the frame is contained in a single descriptor, both the last descriptor (RDES [8]) and the first descriptor (RDES [9]) are set.

The DMA fetches the next descriptor, sets the last descriptor (RDES [8]) bit, and releases the RDES0 status bits in the previous frame descriptor. Then the DMA sets the `EMAC_DMA_STAT.RI` bit. The same process repeats unless the DMA encounters a descriptor the application owns. If this encounter occurs, the receive process sets the `EMAC_DMA_STAT.RU` bit and then enters the suspend state. The position in the receive list is retained.

Receive Descriptor Acquisition

The receive engine always attempts to acquire an extra descriptor in anticipation of an incoming frame. Descriptor acquisition is attempted when any of the following conditions is satisfied:

- The`EMAC_DMA_OPMODE.SR` bit is set immediately after being placed in the run state.
- The data buffer of current descriptor is full before the frame ends for the current transfer.
- The controller completes frame reception, but the current receive descriptor is not yet closed.
- The receive process suspends because of an application-owned buffer (RDES0 [31] = 0) and a new frame is received.
- A receive poll demand issues.

Receive Process Suspended

If a new receive frame arrives while the receive process is in suspend state, the DMA refetches the current descriptor in the application memory. If the DMA now owns the descriptor, the receive process reenters the run state and starts frame reception. If the application still owns the descriptor, by default, the DMA discards the current frame at the

top of the receive FIFO and increments the missed frame counter. If more than one frame is stored in the receive FIFO, the process repeats.

Avoid the discarding or flushing of the frame at the top of the receive FIFO by setting the EMAC_DMA_OPMODE.DFF bit. In such conditions, the receive process sets the receive buffer unavailable status and returns to the suspend state.

OWN Bit (Ownership) Semaphore

Usage or ownership of the transmit or receive descriptor between the application and EMAC is mutually exclusive. While the EMAC accesses the descriptor, the application cannot modify it. Conversely, while the host updates the descriptor, the EMAC cannot use the content of the descriptor. This functionality is implemented through the OWN bit in the transmit or receive descriptor, acting as a semaphore to prevent multiple, simultaneous access to the descriptors.

The following example is based on a usage case of 4 WORDs enabled for descriptors. A chain structure configuration is assumed. (The EMAC_DMA_BUSMODE.ATDS bit is not set). However, the explanation of the OWN bit semaphore remains consistent irrespective of any particular mode of operation.

1. Transmit OWN Bit:
    - TDES0 – TDES3 words implement the transmit descriptors. TDES0 [31] is defined as the OWN bit. When TDES0 [31] is set to 0, this bit indicates that the descriptor is available for the application to update. The application sets up the descriptors, including the buffer addresses, by updating TDES0 through TDES3.
    - To release ownership of the descriptor to the EMAC, the application sets the transmit OWN bit, TDES0 [31], to 1. TDES0 [31] = 1 indicates that the descriptor is ready for the EMAC to use. The DMA reads the descriptors, then fetches the data for transmission from the buffer locations pointed to by the transmit descriptors (TDES2 and TDES3). When either the last data buffer is empty or the end-of-frame is reached, DMA clears the TDES0 [31] bit to 0. Now the transmit descriptor releases to the application for updates.
2. Receive OWN Bit:
    - RDES0 – RDES3 words implement the receive descriptors. RDES0 [31] is defined as the OWN bit. When RDES0 [31] is set to 0, this bit indicates that the descriptor is available for the application to update. The application sets up the descriptors, including the buffer locations for writing the received data, by updating RDES0 through RDES3. To give ownership of the descriptor to the EMAC, the host sets the receive OWN bit, RDES0 [31], to 1.
    - RDES0 [31] = 1 indicates that the descriptor is ready for use by the EMAC. The DMA of the EMAC reads the descriptors, then writes the received data to the buffers with locations pointed to by the receive descriptors (RDES2 and RDES3). When either the last data buffer is full or the end-of-frame is reached, DMA clears the RDES0 [31] bit to 0. Now the receive descriptor releases to the application for updates

Application Data Buffer Alignment

The transmit and receive data buffers do not have any restrictions on start address alignment. The start address for the buffers aligns to any of the 4 bytes. However, the DMA always initiates transfers with the address aligned to the bus width with dummy data for the byte lanes not required. This alignment typically happens during the transfer of the beginning or end of an Ethernet frame.

Example for Buffer Read

If the transmit buffer address is 0x0002 and 15 bytes must transfer, the DMA reads 5 full words (5 x 32-bit data) from address 0x0000. However, when transferring data to the EMAC transmit FIFO, the extra bytes (the first 2 bytes) are dropped or ignored. Similarly, the last 3 bytes of the last transfer are also ignored. The DMA always transfers a full 32-bit data to the transmit FIFO, unless it is the end-of-frame.

Example for Buffer Write

If the receive buffer address is 0x0002 and 15 bytes of a received frame must transfer, the DMA writes 5 full words (5 x 32-bit data) to address 0x0000. However, the first 2 bytes of first transfer and the last 3 bytes of the third transfer have dummy data.

Buffer Size Calculations

The DMA engines do not update the size fields in the transmit and receive descriptors alone. The DMA updates only the status fields (RDES0 and TDES0) of the descriptors. The driver must perform the size calculations. The transmit DMA transfers the exact number of bytes (indicated by buffer size field of TDES1) towards the EMAC CORE. If a descriptor is marked as first (`FS` bit of TDES1 is set), then the DMA marks the first transfer from the buffer as the start of frame. If a descriptor is marked as last (`LS` bit of TDES1), the DMA marks the last transfer from that data buffer as the end-of frame to the EMAC.

The receive DMA transfers data to a buffer until the buffer is full or the end-of frame is received from the MFL. If a descriptor is not marked as last (`LS` bit of RDES0), then the buffers of the descriptor are full. The amount of valid data in a buffer is its buffer size field minus the data buffer pointer offset, when the `FS` bit of that descriptor is set. The offset is zero when the data buffer pointer aligns to the data bus width. If a descriptor is marked as last, then the buffer cannot be full (as indicated by the buffer size in RDES1). To compute the amount of valid data in this final buffer, the driver must:

- Read the frame length (`FL` bits of RDES0[29:16]), and
- Subtract the sum of the buffer sizes of the preceding buffers in the frame

The receive DMA always transfers the start of next frame with a new descriptor.

EMAC FIFO Layer (EMAC MFL)

The MAC FIFO layer provides FIFO memory to buffer and regulates the frames between the application system memory and the EMAC CORE. It also allows the transfer of data between the application clock domain and the EMAC clock domains. The MFL layer has transfer controllers for each direction, called the transmit controller

(TxFIFO) and the receive controller (RxFIFO). The datapath for both directions is 32-bit wide and each controller has a dedicated FIFO.

The transmit FIFO size is fixed to 256 bytes. The receive FIFO size is fixed to 128 bytes.

FIFO Layer Transmit Path

The DMA engine controls all transactions for the transmit path with the application. Ethernet frames read from the system memory are pushed into the FIFO by the DMA. The frame is then popped out and transferred to the EMAC CORE when triggered. When the end-of-frame transfers, the status of the transmission is taken from the EMAC CORE and transferred back to the DMA. FIFO-fill level is indicated to the DMA so that it can initiate a data fetch in required bursts from the system memory, through the SCB interface.

When the DMA enables the `EMAC_DMA_OPMODE.OSF` bit, the MFL receives the second frame into the FIFO while transmitting the first frame. When the first frame has transferred, the status is sent to DMA. If the DMA has already completed sending the second packet to the MFL, it waits for the status of the first packet before proceeding to the next frame.

The following are the modes of operation for FIFO transactions.

1. Threshold mode. When the number of bytes in the FIFO crosses the configured threshold level, the data is ready to be popped out and forwarded to the EMAC core. (The data is also ready when the end-of-frame is written before the threshold is crossed). The DMA uses the `TTC` bits of the DMA bus mode register to configure the threshold level.
2. Store-and-Forward mode. In this mode, the MFL pops the frame towards the EMAC core after a complete frame is stored in the FIFO. If the TX FIFO size is smaller than the Ethernet frame for transmission (such as a jumbo frame), then the frame forwards in two cases. The TX FIFO is almost full or the requested FIFO does not have space to accommodate the requested burst-length. Therefore, the FIFO read controller never stalls in store-and-forward mode even if the Ethernet frame length is bigger than the TX FIFO depth.

Transmit FIFO and Half-Duplex Retransmissions

While a frame transfers from the FIFO, a collision event can occur on the EMAC line interface in half-duplex mode. The EMAC then indicates a retry attempt to the MFL. The EMAC gives the status before the end-of-frame transfers from MFL. Then, the MFL enables the retransmission by popping out the frame again from the FIFO.

After more than 96 bytes pop out of FIFO, the FIFO controller frees up that space. The controller makes it available to the DMA to push in more data. Retransmission is not possible after this threshold is crossed or when the EMAC CORE indicates a late-collision event.

Transmit FIFO Flush Operation

The EMAC provides a control to the software to flush the transmit FIFO in the MFL layer using the `EMAC_DMA_OPMODE.FTF` bit. The flush operation is immediate. The MFL clears the transmit FIFO and the corresponding pointers to the initial state. It clears the FIFO and pointers even if it is in the middle of transferring a frame to the EMAC CORE. The data that the MAC transmitter has already accepted is not flushed. The data is

scheduled for transmission and results in underflow. The transmit FIFO does not complete the transfer of the rest of the frame. As in all underflow conditions, a runt frame is transmitted and observed on the line. The status of the frame is marked with both underflow and frame flush events (TDES0 bits 13 and 1).

The MFL layer also stops accepting any data from the application (DMA) during the flush operation. The MFL generates and transfers transmit status words to the application for the number of frames flushed inside the MFL (including partial frames). Frames that completely flush in the MFL have the status bit for frame flush (TDES0 bit 13) set. The MFL completes the flush operation when the application (DMA) accepts all of the status words for the frames that flushed. The MFL then clears the transmit FIFO flush control register bit. The MFL starts accepting new frames from the application (DMA).

FIFO Layer Receive Path

The receive controller operates in the following sequence.

1. When the EMAC core receives a frame, it pushes in data with the frame start and end indicators. The MFL accepts the data and pushes it into the FIFO.
2. The receive controller takes the data out of the FIFO and sends it to the DMA.
    - Threshold mode (default). This mode is configured using `EMAC_DMA_OPMODE.RTC`. When the FIFO receives 64 bytes or a full packet of data, the receive controller pops out the data and indicates its availability to the DMA. Some error frames cannot be dropped, because the error status is received at the end-of-frame. By this time, the start of that frame has already been read out of the FIFO.
    - Rx FIFO Store-and-Forward mode. This mode is configured using `EMAC_DMA_OPMODE.RSF`. A frame is read out only after being written completely into the receive FIFO. In this mode, all error frames are dropped such that only valid frames are read out and forwarded to the application. (Error frames are dropped when the EMAC core is configured for this feature)
3. After the end-of-frame transfers, the status word from EMAC core is also the pushed FIFO. When the status of a partial frame due to overflow is given out, the frame length field in the status word is not valid.

Receive FIFO Multi-Frame Handling

Since the status is available immediately following the data, the MFL stores any number of frames into the FIFO, as long as it is not full.

Receive FIFO Error Handling

If the MFL Rx FIFO is full before it receives the end-of-frame data from the EMAC, the DMA declares an overflow condition. The whole frame (including the status word) drops. The overflow counter in the DMA (overflow counter-register) increments. This activity occurs even if the `EMAC_DMA_OPMODE.FEF` bit is set. If the start address of such a frame has already transferred, the rest of the frame drops. A dummy end-of-frame is written to the FIFO along with the status word. The status indicates a partial frame due to overflow. In such frames, the frame length field is invalid.

The MFL receive control logic can filter error and undersized frames using the EMAC_DMA_OPMODE.FEF and EMAC_DMA_OPMODE.FUF bits. If the start address of the frame has already transferred to the Rx FIFO read controller, that frame is not filtered. The start address of the frame transfers to the read controller after the frame crosses the receive threshold (set by the EMAC_DMA_OPMODE.RTC bits).

If the MFL receive FIFO is configured for store-and-forward mode, it can filter and drop all error frames.

EMAC CORE

The EMAC CORE is the lowest block in the EMAC peripheral and it performs all operations with the external world (PHY chip). It has independent transmit and receive modules. The modules interact with the EMAC FIFO layer at one end and the PHY chip through the reduced media-independent interface interface at the other end. Both modules have several sub blocks which are discussed in subsequent sections.

Transmission initiates when the MFL (FIFO layer) pushes in data with start-of-frame. The CORE then transmits to the reduced media-independent interface. After the end-of-frame transfers out, the CORE gives the status of the transmission back to the MFL. The MFL forwards the transmission to the application through DMA.

A receive operation initiates when the EMAC detects an SFD on the RMII. The CORE strips the preamble and SFD before proceeding to process the frame. The header fields are checked for the filtering and the FCS field used to verify the CRC for the frame. The frame drops in the core when it fails the address filter.

NOTE: The term *CORE* (written in capitals) refers to the internal block of Ethernet peripheral. Do not confuse the term with the *processor core*.

Table 23-25: EMAC CORE Related Registers

Register Name	Description
MAC Configuration*[1]	Establishes receive and transmit operating modes including: • Watchdog, Jabber, and Jumbo frame sizes • Inter Frame Gap • Speed Control – 10/100 Mbps • Full or Half Duplex • Loopback Mode • Checksum Offload • Enabling Tx or Rx Engines
MAC Frame Filter	Contains the filter controls for receiving frames. Some of the controls from this register go to the address check block of the MAC, which performs the first level of address filtering. The second level of filtering is performed on the incoming frame, based on other controls such as pass bad frames and pass control frames.
Hash Table High/Low[1]	A 64-bit hash table is used for group address filtering. For hash filtering, the contents of the destination address in the incoming frame passes through the CRC logic. The upper 6 bits of the CRC register index the contents of the hash table.

Table 23-25: EMAC CORE Related Registers (Continued)

Register Name	Description
SMI Address[1]	Controls the management cycles to the external PHY through the Station Management interface. The register also includes a field to program the frequency of MDC.
SMI Data[1]	Stores write data for the PHY register at the address specified in SMI Address register. This register also stores read data from the PHY register at the address specified by SMI address register.
Flow Control[1]	Controls the generation and reception of the control (pause command) frames by the flow control module of the EMAC. The fields of the control frame are selected as specified in the 802.3x specification. The EMAC uses the pause time value from this register in the pause time field of the control frame. The host must make sure that the activate bit is cleared before writing to the register.
VLAN Tag[1]	Contains the IEEE 802.1Q VLAN tag to identify the VLAN frames. The MAC compares the 13th and 14th bytes of the receiving frame (length or type) with 16.h8100. The following 2 bytes are compared with the VLAN tag. If a match occurs, it sets the received VLAN bit in the receive frame status. The legal length of the frame increases from 1518 bytes to 1522 bytes.
Debug	Provides the status of all main modules of the transmit and receive datapaths and the FIFOs. An all-zero status indicates that the MAC core is in idle state (and FIFOs are empty) and no activity exists in the datapaths.
Interrupt Status	The contents of this register identify the events in the EMAC-CORE that can generate MMC and PTP-related interrupts.
Interrupt Mask	Enables the program to mask the interrupt signal because of the corresponding PTP event in the interrupt status register.
MAC Address0 High/Low[1]	Holds the upper or lower 16 bits of the MAC address of the station. The first DA byte that is received on the RMII interface corresponds to the LS byte (bits [7:0]) of the MAC address low register. For example, if 0x112233445566 is received (0x11 is the first byte) on the RMII as the destination address, then the macaddress0 register [47:0] is compared with 0x665544332211.
Operation Mode[1]	

*1 There must not be any further writes to these registers until the first write updates. Otherwise, the second write operation is not updated properly. For correct operation, the delay between two writes to the same register location must be at least 8 cycles of 50MHz RMII REFCLK.

NOTE: Refer to the “Register Description” section for the detailed bit-level explanation of the registers.

EMAC CORE Transmission Engine

The following modules constitute the transmission function (transmission engine components) of the EMAC:

- Transmit Bus Interface Module (TBU)
- Transmit Frame Controller Module (TFC)
- Transmit Checksum Offload Engine (TCOE)

- Transmit Protocol Engine Module (TPE)
- Transmit Scheduler Module (STX)
- Transmit CRC Generator Module (CTX)
- Transmit Flow Control Module (FTX)

Transmit Bus Interface Module (TBU)

This module interfaces the transmit path of the EMAC CORE with the MAC Layer FIFO interface. This module outputs the transmit status to the application at the end of normal transmission or collision.

Transmit Frame Controller Module (TFC)

The transmit frame controller regulates frames as well as converts the 32-bit input data into an 8-bit stream.

When the number of bytes received from the application falls below 60 (DA+SA+LT+DATA), the state machine automatically appends zeros to the transmitting frame. The state machine makes the data length exactly 46 bytes to meet the requirement for minimum data field of IEEE 802.3. The EMAC module can also be programmed to not append any padding.

The frame controller receives the computed CRC and appends it as the FCS field to the data transmitting out. When the EMAC is programmed to not append the CRC value to the end of Ethernet frames, the TFC module ignores the computed CRC. However, when the EMAC is programmed to append pads for frames (DA+SA+LT +DATA) less than 60 bytes, then the CRC is always appended at the end of padded frame.

Transmit Checksum Offload Engine (TCOE)

Communication protocols such as TCP and UDP implement checksum fields, which help determine the integrity of data transmitted over a network. The most widespread use of Ethernet is to encapsulate TCP and UDP over IP datagrams. Therefore, the EMAC has a checksum offload engine (COE) to support checksum calculation and insertion in the transmit path, and error detection in the receive path.

NOTE: The checksum for TCP, UDP, or ICMP is calculated over a complete frame, and then inserted into its corresponding header field. Because of this requirement, this function is enabled only when the transmit FIFO configuration is for store-and-forward mode. (The `EMAC_DMA_OPMODE.TSF` bit is set). If the MAC configuration is for threshold (cut-through) mode, the transmit COE is bypassed.

NOTE: Programs must make sure that the transmit FIFO is deep enough to store a complete frame before that frame transfers to the EMAC CORE transmitter. The program must enable the checksum insertion only in the frames that are less than the following number of bytes in size (even in the store-and-forward mode): FIFO depth – PBL – 3 FIFO locations, where PBL is the programmed burst-length in the DMA bus mode register.

IP Header Checksum

In IPv4 datagrams, the 16-bit header checksum field indicates the integrity of the header fields (bytes 11 and 12 of the IPv4 datagram). The COE detects an IPv4 datagram when the Ethernet type field of the frame has the value 0x0800 and the version field of the IP datagram has the value 0x4. The checksum field of the input frame is ignored during calculation and replaced with the calculated value.

The IP header error status bit in transmit descriptor word TDES0 indicates the result of this IP header checksum calculation. The status bit is set whenever the values of the Ethernet type field and the IP header version field are not consistent. Or, the status bit is set when the Ethernet frame does not have enough data, as indicated by the IP header length field. In other words, this bit is set when an IP header error is asserted under the following circumstances.

For IPv4 datagrams:

- The received Ethernet type is 0x0800, but the version field of the IP header is not equal to 0x4.
- The IPv4 header length field indicates a value less than 0x5 (20 bytes).
- The total frame length is less than the value given in the IPv4 header length field.

For IPv6 datagrams:

- The Ethernet type is 0x86dd but the IP header version field is not equal to 0x6.
- The frame ends before the IPv6 header (40 bytes) or extension header (as given in the corresponding header length field in an extension header) is received.

If the COE detects an IP header error, it still inserts an IPv4 header checksum if the Ethernet type field indicates an IPv4 payload.

NOTE: IPv6 headers do not have a checksum field. Therefore, the COE does not modify the IPv6 header fields.

TCP/UDP/ICMP Checksum

The TCP/UDP/ICMP checksum engine processes the IPv4 or IPv6 header (including extension headers) and determines whether the encapsulated payload is TCP, UDP, or ICMP.

NOTE: See IETF specifications RFC 791, RFC 793, RFC 768, RFC 792, RFC 2460, and RFC 4443 for IPv4, TCP, UDP, ICMP, IPv6, and ICMPv6 packet header specifications, respectively.

NOTE: For non-TCP/UDP/ICMP/ICMPv6 payloads, this checksum engine is bypassed and nothing further is modified in the frame.

NOTE: For ICMP-over-IPv4 packets, the checksum field in the ICMP packet must always be 0x0000 in both modes, because pseudo-headers are not defined for such packets. If it does not equal 0x0000, an incorrect checksum can be inserted into the packet.

NOTE: This engine does not process fragmented IP frames (IPv4 or IPv6), IP frames with security features (such as an encapsulated security payload), and IPv6 frames with routing headers. The checksum engine bypasses the checksum insertion for such frames even if the checksum insertion is enabled.

The checksum is calculated for the TCP, UDP, or ICMP payload and inserted into its corresponding field in the header. This engine can work in the following two ways.

- The TCP, UDP, or ICMPv6 pseudo-header is not included in the checksum calculation and is assumed to be present in the checksum field of the input frame. This engine includes the checksum field in the checksum calculation, and then replaces the checksum field with the final calculated checksum.
- The engine ignores the checksum field, includes the TCP, UDP, or ICMPv6 pseudo-header data into the checksum calculation, and overwrites the checksum field with the final calculated value.

The status bit for the payload checksum error in the transmit descriptor word TDES0 indicates the result of this operation. The checksum engine sets the status bit for the payload checksum error when:

- The checksum engine detects that the frame has been forwarded to the MAC transmitter engine in the store-and-forward mode, *and*
- The end-of-frame (EOF) has not been written to the FIFO, *or*
- The packet ends before the number of bytes indicated by the payload length field in the IP header is received.

When the packet is longer than the indicated payload length, the COE ignores them as stuff bytes, and no error is reported. When the engine detects the first type of error, it does not modify the TCP, UDP, or ICMP header. For the second error type, it still inserts the calculated checksum into the corresponding header field.

Transmit checksum offloading is enabled by setting the CIC bits [23:22] of TDES0 word in the transmit descriptor.

Transmit Protocol Engine Module (TPE)

The transmit protocol engine consists of a state-machine that controls the protocol-level operation of Ethernet frame transmission. The module performs the following functions to meet the IEEE 802.3 specifications.

- Generates preamble and SFD
- Generates jam pattern in half-duplex mode
- Contains timestamp snapshot logic for IEEE 1588 support
- Jabber timeout
- Flow control for half-duplex mode (back pressure)
- Generates transmit frame status

When a new frame transmission is requested, the protocol engine sends out the preamble and SFD, followed by the data received. The preamble is defined as 7 bytes of 10101010 pattern. The SFD is defined as 1 byte of 10101011 pattern.

The collision window is defined as 1 slot time (512-bit times for 10/100 Mbps). The jam pattern generation is applicable only to half-duplex mode, not to full-duplex mode. A collision can occur any time from the beginning of the frame to the end of the CRC field. When a collision happens, the state machine sends a 32-bit jam pattern of 0x55555555 on the RMII. The pattern informs all other stations that a collision has occurred. If the collision happens during the preamble transmission phase, it completes the transmission of preamble and SFD and then sends the jam pattern. If the collision occurs after the collision window and before the end of the FCS field, it sends a 32-bit jam pattern. It also sets the late collision bit in the transmit frame status.

The module maintains a jabber timer to cut off the transmission of Ethernet frames when the TFC module transfers more than 2,048 (default) bytes. The timeout changes to 10,240 bytes when the jumbo frame is enabled.

The transmit state machine uses the deferral mechanism for the flow control (back pressure) in half-duplex mode. When the application asks to stop receiving frames, the module sends a JAM pattern of 32 bytes. It sends the pattern whenever it senses a reception of a frame. Transmit flow control must be enabled. This activity results in a collision and the remote station backs off.

The application can request a flow control signal by setting the `EMAC_FLOWCTL.FCBBPA` bit. If the application requests a frame transmission, then it is scheduled and transmitted even when the back pressure is activated. If the back pressure is activated for a long time, then the remote stations abort their transmissions due to excessive collisions. (For example, a long time is when more than 16 consecutive collision events occur).

If PTP timestamping is enabled for the transmit frame, this block takes a snapshot of the PTP System time when the SFD is put onto the transmit bus.

Transmit Scheduler Module (STX)

The transmit scheduler is responsible for scheduling the frame transmission on the RMII. The two major functions of this module are:

- Maintain the inter-frame gap between two transmitted frames.
- Follow the truncated binary exponential back-off algorithm for half-duplex mode.

The scheduler maintains an idle period of the configured inter-frame gap (`EMAC_MACCFG.IFG` bits) between any two transmitted frames. The scheduler starts its IFG counter when the carrier signal of the reduced media-independent interface goes inactive. In half-duplex mode and when IFG is configured for 96-bit times, the scheduler follows the rule of deference specified in Section 4.2.3.2.1 of the IEEE 802.3 specification. The module resets its IFG counter when a carrier is detected during the first two-thirds (64-bit times for all IFG values) of the IFG interval. If the carrier is detected during the final one third of the IFG interval, the scheduler continues the IFG count and enables the transmitter after the IFG interval.

Transmit CRC Generator Module (CTX)

The transmit CRC generator module generates the CRC for the FCS field of the Ethernet frame (DA + SA + LT + DATA + PAD).

This module calculates the 32-bit CRC for the FCS field of the Ethernet frame. The following polynomial defines the encoding:

G (x) = x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1.

Transmit Flow Control Module (FTX)

The transmit flow control module generates pause frames and transmits them to the frame controller as necessary, in full-duplex mode. The application can request the flow control module to send a pause frame by setting the `EMAC_FLOWCTL.FCBBPA` bit.

If the application has requested flow control, the flow control module generates and transmits a single pause frame. The value of the pause time in the generated frame contains the programmed pause time value configured using the `EMAC_FLOWCTL.PT` bit. The module can extend the pause or end the pause prior to the time specified in the previously transmitted pause frame. To change the pause, the application must request another pause frame transmission after programming the `EMAC_FLOWCTL.PT` bit with an appropriate value.

If the flow control signal goes inactive prior to the sampling time, the flow control module transmits a pause frame with zero pause time. This event indicates to the remote end that the receive buffer is ready to receive new data frames.

EMAC CORE Reception Engine

The following are the functional blocks (reception engine components) in the receive path of the EMAC core.

- Receive Protocol Engine Module (RPE)
- Receive CRC Module (CRX)
- Receive Frame Controller Module (RFC)
- Receive Flow Control Module (FRX)
- Receive Checksum Offload Engine (RCOE)
- Receive Bus Interface Unit Module (RBU)
- Address Filtering Module (AFM)

Receive Protocol Engine Module (RPE)

The receive protocol engine is a state-machine that strips the incoming preamble and SFD. Once the receive data valid (`EMAC_CRS`) signal of the RMII becomes active, the protocol engine begins hunting for the SFD field from the receive modifier logic. Until then, the state machine drops the receiving preambles. Once the SFD is detected, it begins sending the data of the Ethernet frame to the frame controller. It begins with the first byte following the SFD (destination address).

NOTE: According to the IEEE 802.3 Ethernet specifications, the EMAC receiver does not need to look or check for the preamble pattern. It has to wait only for the SFD pattern to identify the start of a frame. Then the EMAC receiver accepts a frame even when no preamble is received before the SFD pattern.

If PTP timestamping is enabled, the RPE takes a snapshot of the PTP system time when detecting any SFD of the frame on the reduced media-independent interface. Unless the MAC filters out and drops the frame, this timestamp passes on to the application

The protocol engine also decodes the length or type field of the receiving Ethernet frame. The state machine sends the data of the frame up to the count specified in the length or type field

- If the length or type field is less than 0x600, and
- If the MAC is programmed for the auto CRC or PAD stripping option

It then starts dropping bytes (including the FCS field).

If the length or type field is greater than or equal to 0x600, the protocol engine sends all received Ethernet frame data to the frame controller. The transfer does not depend on the value of the programmed auto-CRC strip option.

The EMAC is programmed with the watchdog timer enabled (default setting). In this configuration, frames above 2,048 (10,240 if jumbo frame is enabled) bytes (DA + SA + LT + DATA + PAD + FCS) are cut off at the protocol engine. Set the `EMAC_MACCFG.WD` bit to disable this feature. However, even when the watchdog timer is disabled, frames greater than 16 KB are cut off and a watchdog timeout status is issued.

The EMAC supports loopback of transmitted frames onto its receiver. By default, the EMAC loopback function is disabled. Set the `EMAC_MACCFG.LM` bit to enable the function.

At the end of every received frame, the protocol engine generates received frame status and sends it to the frame controller. Control, missed frame, and filter fail status are added to the receive status in the frame controller.

Receive CRC Module (CRX)

The receive CRC module checks for any CRC errors in the receiving frame.

This module calculates the 32-bit CRC for the received frame that includes the destination address field through the FCS field (DA+SA+LT+DATA+PAD+FCS). The following generating polynomial defines the encoding.

G (x) = x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1

Irrespective of the auto pad or CRC strip, the CRC module receives the entire frame to compute the CRC check for received frame.

Receive Frame Controller Module (RFC)

The main functions of the frame controller are:

- Converting the 8-bit stream data to 32-bit data
- Frame filtering
- Attaching the calculated IP checksum
- Updating the receive status

If the EMAC_MACFRMFILT.RA bit is set, the RFC module initiates the data transfer as soon as possible. At the end of the data transfer, the frame controller sends out the received frame status that includes the address filtering pass or fail status.

If the EMAC_MACFRMFILT.RA bit is reset, the frame controller performs frame filtering based on the destination or source address. (The application still must perform another level of filtering if it decides not to receive any bad frames like runt, CRC error frames, for example). After receiving the destination or source address bytes, the frame controller checks the filter-fail signal from the AFM module for an address match. On detecting a filter-fail from AFB, the frame is dropped and not transferred to the application.

Receive Flow Control Module (FRX)

The receive flow controller detects the receiving pause frame and pauses the frame transmission for the delay specified within the received pause frame. The flow controller is enabled only in full-duplex mode. The EMAC uses the EMAC_FLOWCTL.RFE bit to enable or disable the function for pause frame detection.

Once the receive flow control is enabled, the flow controller begins monitoring the received frame destination address for any match with the multicast address of the control frame (0x0180C2000001). If a match is detected, it indicates to the frame controller, that the destination address of the received frame matches the reserved control frame destination address. The RFC module then decides whether to transfer the received control frame to the application, based on the EMAC_MACFRMFILT.PCF bit setting.

The receive flow controller also decodes the type, opcode, and pause timer field of the receiving control frame. The flow controller requests the MAC transmitter pause the transmission of any data frame.

- If the byte count of the frame status indicates 64 bytes, and
- If there is no CRC error

The transmission is paused for the decoded pause time value, multiplied by the slot time (64-byte times). Meanwhile, if another pause frame is detected with a zero pause time value, the module resets the pause time and gives another pause request to the transmitter. The module does not generate a pause request to the transmitter:

- If the received control frame does not match the type field (0x8808), opcode (0x00001), or byte length (64 bytes), or
- If there is a CRC error

For a pause frame with a multicast destination address, the frame controller filters the frame based on the address match from the flow controller. For a pause frame with a unicast destination address, the filtering in the FRX module depends on:

- If the destination address matched the contents of the MAC address register 0 (EMAC_ADDR0_HI or EMAC_ADDR0_LO), and
- If the EMAC_FLOWCTL.UP bit is set

(The module detects a pause frame even with a unicast destination address). The EMAC uses the EMAC_MACFRMFILT.PCF bits to control the filtering for control frames in addition to the address filter module.

Receive Checksum Offload Engine (RCOE)

When checksum offloading is enabled, both IPv4 and IPv6 frames in the received Ethernet frames are detected and processed for data integrity. Programs can enable this module by setting the `EMAC_MACCFG.IPC` bit. The EMAC receiver identifies IPv4 or IPv6 frames by checking for value 0x0800 or 0x86DD, respectively, in the received Ethernet type field of frames. This identification applies to VLAN-tagged frames as well. *Extended descriptor mode (8 x32-bit words) must be enabled to get the IPC checksum engine status in RDES4.* To check status, poll bit 0 of RDES0 word of receive descriptor. Then, if this bit is set, parse bits [7:0] of RDES4 word.

The receive checksum offload engine calculates IPv4 header checksums and checks if they match the received IPv4 header checksums. The IP header error bit is set for any mismatch between the indicated payload type (Ethernet type field) and the IP header version. The IP header error bit is also set when the received frame does not have enough bytes, as indicated by the length field of the IPv4 header. (The bit is set when fewer than 20 bytes are available in an IPv4 or IPv6 header).

This engine also identifies a TCP, UDP, or ICMP payload in the received IP datagrams (IPv4 or IPv6). The engine calculates the checksum of such payloads properly, as defined in the TCP, UDP, or ICMP specifications. This engine includes the TCP/UDP/ICMPv6 pseudo-header bytes for checksum calculation and checks whether the received checksum field matches the calculated value. The result of this operation appears as a payload checksum error bit in the receive status word. This status bit is also set if the length of the TCP, UDP, or ICMP payload does not tally to the expected payload length given in the IP header.

NOTE: The COE engine bypasses the payload of fragmented IP datagrams, IP datagrams with security features, IPv6 routing headers, and payloads other than TCP, UDP, or ICMP. This information is given in the receive status(whether the checksum engine is bypassed or not).

The ***Checksum Error Status*** table shows bit combination in receive descriptors (frame status with full checksum offload engine enabled and advanced timestamps not enabled).

Table 23-26: Checksum Error Status

IEEE802.3 Frame: bit 5 of RDES0	Header Checksum Error: bit 3 of RDES4	Payload Checksum Error: bit 4 of RDES4	Frame Status
0	0	0	The frame is an IEEE 802.3 frame (length field value is less than 0x0600).
1	0	0	IPv4/IPv6 type frame in which no checksum error is detected.
1	0	1	IPv4/IPv6 type frame in which a payload checksum error (as described for PCE) is detected
1	1	0	IPv4/IPv6 type frame in which IP header checksum error (as described for IPC HCE) is detected.
1	1	1	IPv4/IPv6 type frame in which both PCE and IPC HCE is detected.

Table 23-26: Checksum Error Status (Continued)

IEEE802.3 Frame: bit 5 of RDES0	Header Checksum Error: bit 3 of RDES4	Payload Checksum Error: bit 4 of RDES4	Frame Status
0	0	1	IPv4/IPv6 type frame in which there is no IP HCE and the payload check is bypassed due to unsupported payload.
0	1	1	Type frame which is neither IPv4 or IPv6 (COE bypasses the checksum check completely)
0	1	0	Reserved

Receive Bus Interface Unit Module (RBU)

The receive bus interface unit (RBU) constructs the 32-bit data received from the frame controller into a 32-bit FIFO-based protocol.

Address Filtering Module (AFM)

The address filtering (AFM) module performs the destination checking function on all received frames and reports the address filtering status to the frame controller. The address checking is based on different parameters (frame filter register, EMAC_MACFRMFILT) chosen by the application. These parameters are inputs to the AFM module as control signals. The AFM module reports the status of the address filtering based on the combination of these inputs. The AFM module also reports whether the receiving frame is a multicast frame or a broadcast frame, as well as the address filter status. The AFM module uses the physical (MAC) address of the station and the multicast hash table for address checking.

- ***Hash or Perfect Address Filter.*** The destination address filter can be configured to pass a frame when its destination address matches either the hash filter or the perfect filter. Set the EMAC_MACFRMFILT.HPF bit, the corresponding EMAC_MACFRMFILT.HUC, or EMAC_MACFRMFILT.HMC bits. This configuration applies to both unicast and multicast frames. If the EMAC_MACFRMFILT.HPF bit is reset, only one of the filters (hash or perfect) is applied to the received frame.

  NOTE: Hash filtering is not perfect filtering because a 48-bit MAC address is reduced to a 6-bit hash value. So, there can be instances where more than one address has the same hash value.

- ***Unicast Destination Address Filter.***
  - The AFM supports one MAC address for unicast perfect filtering. If perfect filtering is selected, the AFM compares all 48 bits of the received unicast address with the programmed MAC address for any match. (The EMAC_MACFRMFILT.HUC bit is reset for perfect filtering).
  - In hash filtering mode, the AFM performs imperfect filtering for unicast addresses using a 64-bit hash table. (The EMAC_MACFRMFILT.HUC bit is set in hash filtering mode). For hash filtering, the AFM uses the upper 6-bit CRC of the received destination address to index the content of the hash table. A value of 000000 selects bit 0 of the selected register, and a value of 111111 selects bit 63 of the hash table register.

If the corresponding bit (indicated by the 6-bit CRC) is set to 1, the unicast frame has passed the hash filter. Otherwise, the frame has failed the hash filter.

- ***Multicast Destination Address Filter.***
  - Program the EMAC to pass all multicast frames by setting the `EMAC_MACFRMFILT.PM` bit. If the `EMAC_MACFRMFILT.PM` bit is reset, the AFM filters multicast addresses based on the `EMAC_MACFRMFILT.HMC` bit. In perfect filtering mode, the multicast address is compared with the programmed MAC destination address register. The EMAC also supports group address filtering.
  - In hash filtering mode, the AFM performs imperfect filtering using a 64-bit hash table. For hash filtering, the AFM uses the upper 6-bit CRC of the received multicast address to index the content of the hash table. A value of 000000 selects bit 0 of the selected register and a value of 111111 selects bit 63 of the hash table register. If the corresponding bit is set to 1, then the multicast frame has passed the hash filter. Otherwise, the frame has failed the hash filter.
- ***Broadcast Address Filter.*** The AFM does not filter any broadcast frames in the default mode. However, if the EMAC is programmed to reject all broadcast frames by setting the `EMAC_MACFRMFILT.DBF` bit, the AFM asserts the filter fail signal, whenever a broadcast frame is received.
- ***Inverse Filtering Operation.***There is an option to invert the filter-match result at the final output. The EMAC uses the `EMAC_MACFRMFILT.DAIF` bit to control this operation. The function of this bit applies to both unicast and multicast DA frames. The result of the unicast or multicast destination address filter is inverted in this mode.

Destination Address Filtering

The ***Destination Address Filtering*** table provides filtering possibilities for the destination address using the EMAC AFM module. The MAC receive frame filter register (`EMAC_MACFRMFILT`) contains these bits.

Table 23-27: Destination Address Filtering

Frame Type	Bit Setting (0 = Cleared, 1 = Set, X = Do-not-care)							DA Filter Operation
	PR	HPF	HUC	HMC	DAIF	PM	DBF	
Broadcast	1	X	X	X	X	X	X	Pass
	0	X	X	X	X	X	0	Pass
	0	X	X	X	X	X	1	Fail
Unicast	1	X	X	X	X	X	X	Pass all frames
	0	X	0	X	0	X	X	Pass on perfect or group filter match
	0	X	0	X	1	X	X	Fail on perfect or group filter match
	0	0	1	X	0	X	X	Pass on hash filter match
	0	0	1	X	1	X	X	Fail on hash filter match

Table 23-27: Destination Address Filtering (Continued)

Frame Type	Bit Setting (0 = Cleared, 1 = Set, X = Do-not-care)							DA Filter Operation
	PR	HPF	HUC	HMC	DAIF	PM	DBF	
	0	1	1	X	0	X	X	Pass on hash or perfect or group filter match
	0	1	1	X	1	X	X	Fail on hash or perfect or group filter match
Multicast	1	X	X	X	X	X	X	Pass all frames
	X	X	X	X	X	1	X	Pass all frames
	0	X	X	0	0	0	X	Pass on perfect or group filter match and drop PAUSE control frames if PCF = 0x
	0	0	X	1	0	0	X	Pass on hash filter match and drop PAUSE control frames if PCF = 0x
	0	1	X	1	0	0	X	Pass on hash or perfect or group filter match and drop PAUSE control frames if PCF = 0x
	0	X	X	0	1	0	X	Fail on perfect or group filter match and drop PAUSE control frames if PCF = 0x
	0	0	X	1	1	0	X	Fail on hash filter match and drop PAUSE control frames if PCF = 0x
	0	1	X	1	1	0	X	Fail on hash or perfect or group filter match and drop PAUSE control frames if PCF = 0x

EMAC Station Management Interface (SMI)

The IEEE 802.3 MII station management interface, also known as the MDIO management interface, allows the processor to monitor and control one or more external Ethernet physical-layer transceivers. (Physical-layer transceivers are commonly called PHYs). The management interface physically consists of a 2-wire serial connection composed of the MDC (management data clock) output signal and the MDIO (management data input/output) bidirectional data signal. The IEEE 802.3 MII station management interface also applies to RMII.

The application can select one of the 32 PHYs and one of the 32 registers within any PHY and send control data or receive status information. Only one register in one PHY can be addressed at any given time. The EMAC CORE initiates all the transfers. The PHY chip only acts as a slave device.

Standard PHY control and status registers typically provide

- Device capability status bits (for example: auto-negotiation, duplex modes, 10/100 speeds, and protocols)
- Device status bits (for example: auto-negotiation complete, link status, remote fault)
- Device control bits (for example: reset, speed selection, loopback, and auto-negotiation start)

Upon power-up, an MDIO read access (at default rates) of device capabilities in PHY status registers can determine the supported PHY features.

The MII management logical interface specifies:

- A set of 16-bit device control or status registers within the PHYs, including both required registers with standardized bit definitions as well as optional vendor-specified registers.
- A 5-bit device addressing scheme which allows the MAC to select one of up to 32 externally connected PHY devices.
- A 5-bit register addressing scheme for selecting the target register within the addressed device.
- A transfer frame protocol for 16-bit read and write accesses to PHY registers through the MDC and MDIO signals under control of the MAC.

Table 23-28: Station Management Interface pins

Station Management Interface Pins	Pin Description
MDIO – Management Data I/O	A periodic clock that runs at a maximum period of 400 ns. Always driven by the EMAC to PHY.
MDC – Management Data Clock	Data signal driven by EMAC or PHY, depending on write or read access based on EMAC; synchronous to MDC.

MDC Clock Frequency

The EMAC uses the `EMAC_SMI_ADDR.CR` bit field to determine the frequency of MDC as shown in the ***MDC Clock Frequency Selection*** table. The clock range selection determines the frequency of the clock relative to the SYSCLK frequency. The table shows the suggested range of SYSCLK frequency applicable for each value of the `EMAC_SMI_ADDR.CR` field. The programmability based on SYSCLK frequency range ensures that the MDC clock frequency range is within the IEEE specifications of 1.0 MHz to 2.4 MHz. However, the EMAC MDC can also support higher frequencies for PHY devices that support the frequencies.

Table 23-29: MDC Clock Frequency Selection

`EMAC_SMI_ADDR.CR` Selection	**Programmed SYSCLK Frequency Range**	**Frequency of MDC**	**Min and Max MDC Freq (Per Specifications)**
0000	60–100 MHz	SYSCLK/42	MIN = 1.43 MHz and MAX = 2.39 MHz
0010	20–35 MHz	SYSCLK/16	MIN = 1.25 MHz and MAX = 2.19 MHz

Table 23-29: MDC Clock Frequency Selection (Continued)

EMAC_SMI_ADDR.CR Selection	Programmed SYSCLK Frequency Range	Frequency of MDC	Min and Max MDC Freq (Per Specifications)
0011	35–60 MHz	SYSCLK/26	MIN = 1.35 MHz and MAX = 2.31 MHz

The ***MDIO Frame Parameters*** table provides MDIO data transfer parameters. The write and read sequences provided in the tables, ***MDIO Write Data Sequence*** and ***MDIO Read Data Sequence***, are based on these parameters.

Table 23-30: MDIO Frame Parameters

Parameter	Description
IDLE	The MDIO line is three-state (noted as Z in sequence); there is no clock on MDC.
PREAMBLE	32 continuous bits, each of value 1
START	Start-of-frame is 01
OPCODE	10 for read and 01 for write
PHY ADDR	5-bit address select for one of 32 PHYs (noted as AAAAA in sequence)
REG ADDR	Register address in the selected PHY (noted as RRRRR in sequence)
TA	Turnaround is Z0 for read and 10 for write (Z = high impedance)
DATA	Any 16-bit value. Driven by MAC or PHY based on direction (noted as DDD...DDD).

Table 23-31: MDIO Write Data Sequence

IDLE	PREAMBLE	START	OPCODE	PHY ADDR	REG ADDR	TA	DATA	IDLE
Z	111...111	01	01	AAAAA	RRRRR	10	DDD... DDD	Z

Table 23-32: MDIO Read Data Sequence

IDLE	PREAMBLE	START	OPCODE	PHY ADDR	REG ADDR	TA	DATA	IDLE
Z	111...111	01	10	AAAAA	RRRRR	Z0	DDD... DDD	Z

SMI Write Operation

When programs set the EMAC_SMI_ADDR.SMIW (write) and EMAC_SMI_ADDR.SMIB (busy) bits, the station management interface (SMI) initiates a write operation into the PHY registers. The write operation uses the management frame format (the PHY address, the register address in PHY, and the write data) specified in the IEEE specifications. (Section 22.2.4.5 of IEEE standard). The application must not change the EMAC_SMI_ADDR register contents or the EMAC_SMI_DATA register while the transaction is ongoing.

Write operations to the EMAC_SMI_ADDR register or the EMAC_SMI_DATA register during the transfer period are ignored (while the EMAC_SMI_ADDR.SMIB bit is high). The transaction completes without error. After the write

operation has completed, the SMI indicates the same by resetting the EMAC_SMI_ADDR.SMIB bit. The EMAC drives the MDIO line for the complete duration of the frame. The ***SMI Write Operation through MDIO/MDC Pins*** figure shows this operation.

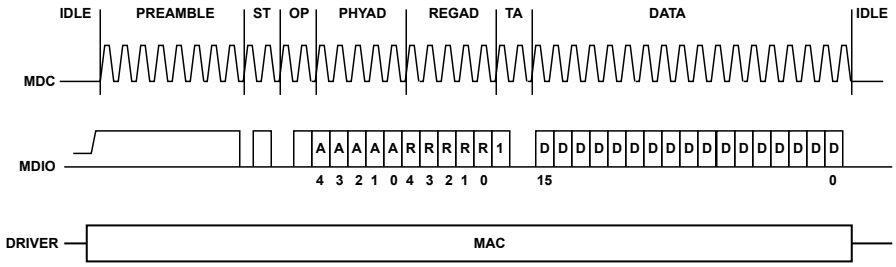

Figure 23-9: SMI Write Operation through MDIO/MDC Pins

SMI Read Operation

When programs set the EMAC_SMI_ADDR.SMIB bit with the EMAC_SMI_ADDR.SMIW bit cleared (=0), the station management interface (SMI) initiates a read operation in the PHY registers. It transfers the PHY address and the register address in the PHY to the SMI. The application must not change the EMAC_SMI_ADDR register contents or the EMAC_SMI_DATA register while the transaction is ongoing.

Write operations to the EMAC_SMI_ADDR register or the EMAC_SMI_DATA register during the transfer period are ignored (while the EMAC_SMI_ADDR.SMIB bit is high). The transaction completes without error. After the read operation has completed, the SMI resets the EMAC_SMI_ADDR.SMIB bit and updates the EMAC_SMI_DATA register with the data read from the PHY. The EMAC drives the MDIO line for the complete duration of the frame except during the data fields when the PHY drives the MDIO line. The ***SMI Read Operation through MDIO/MDC Pins*** figure shows this operation.

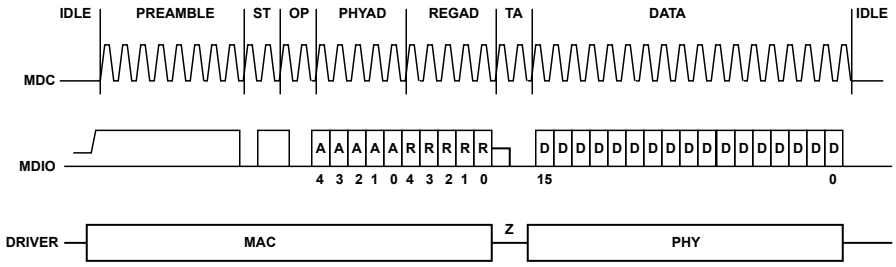

Figure 23-10: SMI Read Operation through MDIO/MDC Pins

EMAC Management Counters (MMC)

The EMAC provides a comprehensive set of 32-bit MAC management counters. It uses these counters for gathering statistics on the received and transmitted frames. The MMC subblock also includes

- A control register (EMAC_MMC_CTL) for managing the behavior of the counters
- Two 32-bit registers containing interrupts generated (EMAC_MMC_RXINT and EMAC_MMC_TXINT)
- Two 32-bit registers containing masks for the interrupt register (EMAC_MMC_RXIMSK and EMAC_MMC_TXIMSK)

The MMC receive counters are updated for frames passed by the address filtering subblock in the EMAC CORE. Statistics of frames dropped by the AFM module are not updated unless they are runt frames of less than 6 bytes. (Destination address bytes are not received fully). The module is also capable of gathering statistics on encapsulated IPv4, IPv6, and TCP, UDP, or ICMP payloads in received Ethernet frames.

The MMC register naming conventions are as follows:

- Tx as a prefix or suffix indicates counters associated with transmission.
- Rx as a prefix or suffix indicates counters associated with reception.
- _G as a suffix indicates registers that count good frames only.
- _GB as a suffix indicates registers that count frames regardless of whether they are good or bad.

Transmitted frames are considered *good* when transmitted successfully. In other words, a transmitted frame is good if the frame transmission does not abort due to any of the following errors:

- Jabber timeout
- No carrier or loss of carrier
- Late collision
- Frame underflow
- Excessive deferral
- Excessive collision

Received frames are good when none of the following errors exists:

- CRC error
- Runt frame (shorter than 64 bytes)
- Alignment error
- Length error (non-type frames only)
- Out-of-range (non-type frames only, longer than maximum size)

The maximum frame size depends on the frame type, as follows:

- Untagged frame maxsize = 1518
- VLAN frame maxsize = 1522
- Jumbo frame maxsize = 9018
- Jumbo VLAN frame maxsize = 9022

The `EMAC_MMC_CTL` register also contains bits that control preset, freeze and roll-over of counters. The EMAC uses the `EMAC_MMC_CTL.RDRST` bit to enable an auto-reset feature whenever the counters are read. The EMAC uses the `EMAC_MMC_CTL.RST` bit to reset all the counters.

The MMC can trigger an interrupt when the corresponding bits are enabled in the transmit, receive, and IPC mask registers, and when the particular counter reaches half or full. The status is also updated in the corresponding interrupt register.

MMC Receive Interrupt Register

The EMAC_MMC_RXINT register maintains the interrupts that are generated when the receive statistic counters reach half their maximum values (0x80000000), and when they cross their maximum values (0xFFFFFFFF). When EMAC_MMC_CTL.NOROLL is set, then interrupts are set, but the counter remains at all ones. The EMAC_MMC_RXINT register is a 32-bit wide register. An interrupt bit is cleared when the respective MMC counter that caused the interrupt is read. The least significant byte lane (bits 7–0) of the respective counter must be read to clear the interrupt bit.

MMC Transmit Interrupt Register

The EMAC_MMC_TXINT register maintains the interrupts generated when the transmit statistic counters reach half their maximum values (0x80000000), and when they cross their maximum values (0xFFFFFFFF). When EMAC_MMC_CTL.NOROLL is set, then interrupts are set, but the counter remains at all ones. The EMAC_MMC_TXINT register is a 32-bit wide register. An interrupt bit is cleared when the respective MMC counter that caused the interrupt is read. The least significant byte lane (bits 7–0) of the respective counter must be read to clear the interrupt bit.

MMC Receive Checksum Offload Interrupt Register

The EMAC_MMC_RXINT.CRCERR register maintains the interrupts generated when receive IPC statistic counters reach half their maximum values (0x80000000), and when they cross their maximum values (0xFFFFFFFF). When EMAC_MMC_CTL.NOROLL is set, then interrupts are set, but the counter remains at all ones. The EMAC_MMC_RXINT.CRCERR register is 32 bits wide. When the MMC IPC counter that caused the interrupt is read, its corresponding interrupt bit is cleared. The least-significant byte lane (bits 7–0) of the counter must be read to clear the interrupt bit.

EMAC Precision Time Protocol (PTP) Engine

The following sections describe the precision time protocol (PTP) engine.

IEEE1588 and the PTP Engine

The Ethernet MAC peripheral includes a PTP engine to assist applications requiring time synchronization. The PTP module is tightly integrated with the EMAC CORE to aid hardware time stamping defined in the IEEE1588 2002/2008 standards. Applications can use accurate hardware timestamps through TCP/IP stacks (if using Network layer communication) to exchange time information across devices connected over network. Applications can also use accurate hardware timestamps through Ethernet device drivers (if using MAC layer communication) to exchange time information.

PTP Engine

For calculation of drift in time between two Ethernet devices, the device records its system time whenever a timing message is sent or received (IEEE 1588 protocol). Due to the indeterministic delay of a software system for a node, the software is unable to capture an accurate time when the message is sent or received. However, the hardware can monitor the signal on the communication media and get an accurate message of arrival and departure time.

The PTP (precision time protocol) module is closely integrated with the EMAC module. It provides hardware assistance to implement both the IEEE 1588–2002 and IEEE 1588–2008 standards on Ethernet (IEEE 802.3). It takes one input clock signal as its PTP clock and maintains the timing information (called *system time*) at the nanosecond level.

The PTP module includes hardware for clock and system time adjustment. The pulse-per-second (PPS) signals physically represent the system time. PPS can be programmed to a fixed frequency or provide flexibility to the signal in terms of pulse width, interval, start, and stop time of the signals. The PTP module can be programmed to trigger an alarm interrupt when system time reaches specified time.

The PTP module can be programmed to detect different types of received frames, capture the system time, and timestamp those frames with the captured system time. Programs can configure any frame so that the PTP module capture the system time when it is transmitted. The PTP module can also capture the system time when an event is detected on the auxiliary snapshot trigger input pins (`EMAC_PTPAUXIN[n]`).

IEEE1588 Standard

Many systems require two independent devices to operate in a time synchronized fashion. If each system relied solely on its oscillator, differences between the characteristics and operating conditions of each oscillator would limit the ability of the clocks to operate synchronously. To serve applications requiring synchronized clocks, the system uses a periodic correction mechanism.

A simple way to synchronize multiple systems is to choose one system (with the best clock) as a master. The system master broadcasts the clock and timing information to other systems (slaves) and then the slaves adjust their clocks and timing according to that of master. However, this method has limitations. The master cannot broadcast the time at infinitesimal intervals. Path delay (propagation delay) exists between a master and a slave and the delay varies between each slave and master.

IEEE 1588 is also known as precision time protocol or PTP. The standard specifies a protocol used to synchronize the time and clock of multiple devices, dispersed but interconnected by any communication. For example, Ethernet (IEEE 802.3). According to the protocol, timing messages are exchanged between two devices. Then, one of the devices calculates its drift from other device and corrects its system time.(Both devices must have the same representation of their system time). The protocol resolves path delay between devices. It also helps synchronize the clocks of multiple devices and all of the limitations mentioned are resolved.

IEEE 1588 was published in 2002 where four types of timing messages were defined: Sync, Follow_Up, Delay_Req, and Delay_Resp. Here the protocol synchronizes two or more devices where one is a master and others are slaves. The master device sends Sync, Follow_Up, and Delay_Resp messages to the slave device in the system. The slave sends the Delay_Req messages to the master device. A following section provides more information on IEEE 1588–2002.

In 2008 a newer version of IEEE 1588 was introduced which provides further mechanisms to measure the peer-to-peer delay. Three more timing messages (PdelayReq, PdelayResp, and PdelayRespFollowup) were added to implement peer-to-peer synchronization. A following section provides more information on IEEE 1588–2008.

IEEE 1588–2002

The IEEE 1588–2002 standard defines the precision time protocol (PTP). The protocol allows precise synchronization of clocks in measurement and control systems that use network communication, local computing, and distributed objects. The protocol applies to systems that communicate by local area networks that support multicast messaging, including (but not limited to) Ethernet. This protocol also allows heterogeneous systems that include clocks of varying inherent precision, resolution, and stability to synchronize. The protocol supports system-wide synchronization accuracy in the submicrosecond range with minimal network and local clock computing resources.

The PTP is transported over UDP/IP. The system or network is classified into master and slave nodes for distributing the timing or clock information. The ***IEEE 1588–2002 PTP Process*** figure shows the process that PTP uses for synchronizing a slave node to a master node by exchanging PTP messages.

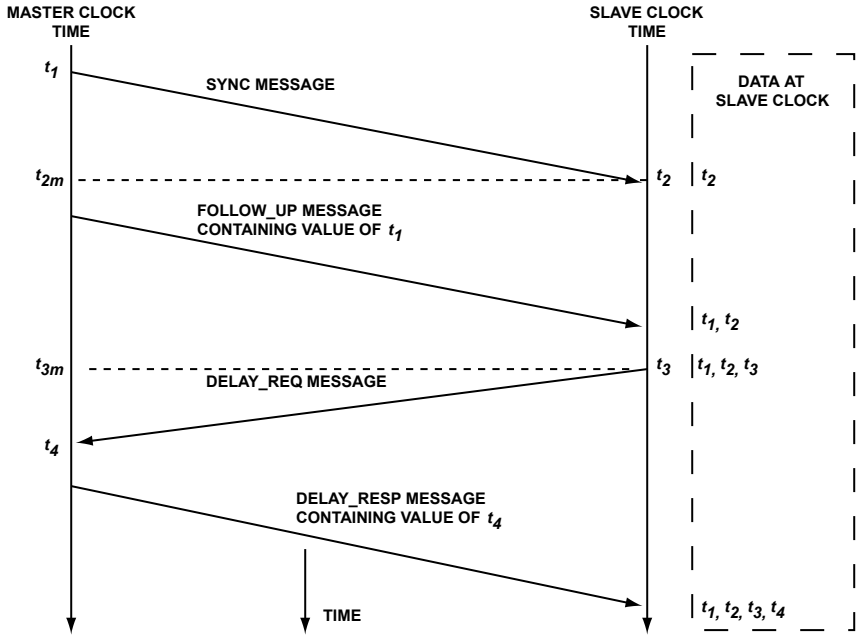

Figure 23-11: IEEE 1588–2002 PTP Process

As shown in the figure, the PTP uses the following process:

1. The master broadcasts the PTP sync messages to all its nodes. The sync message contains the reference time information of the master. The time at which this message leaves the system of the master is t_1. The master must capture this time for Ethernet ports, at RMII.
2. The slave receives the Sync message and also captures the exact time, t_2, using its timing reference.
3. The master sends a Follow_up message to the slave, which contains t1 information for later use.

4. The slave sends a Delay_Req message to the master, noting the exact time, t_3, at which this frame leaves the RMII.
5. The master receives the message, capturing the exact time, t_4, at which it enters its system.
6. The master sends the t_4 information to the slave in the Delay_Resp message.
7. The slave uses the four values of t_1, t_2, t_3, and t_4 to synchronize its local timing reference to the timing reference of the master.

Most of the PTP implementation happens in the software above the UDP layer. However, the hardware support must capture the exact time when specific PTP packets enter or leave the Ethernet port at the RMII. Hardware must capture this timing information and return it to the software for the proper implementation of PTP with high accuracy.

IEEE 1588–2008 Advanced Timestamps

In addition to the basic timestamp features mentioned in IEEE 1588–2002 timestamps, the EMAC supports the following advanced timestamp features defined in the IEEE 1588–2008 standard.

- Support for the IEEE 1588–2008 (Version 2) timestamp format.
- Provides an option to take snapshot of all frames or only PTP type frames.
- Provides an option to take snapshot of only event messages.
- Provides an option to select the node to be a master or slave.
- Identifies the PTP message type, version, and PTP payload in frames sent directly over Ethernet and sends the status.
- Provides an option to run nanoseconds time in digital or binary format.

Peer-to-Peer (P2P) PTP Message Support

The IEEE 1588–2008 version supports Peer-to-Peer PTP (Pdelay) message in addition to SYNC, Delay Request, Follow-up, and Delay Response messages. Refer to the ***Propagation Delay Calculation between Nodes Supporting Peer-to-Peer Path Correction*** figure. The figure shows the method to calculate the propagation delay between nodes supporting peer-to-peer path correction.

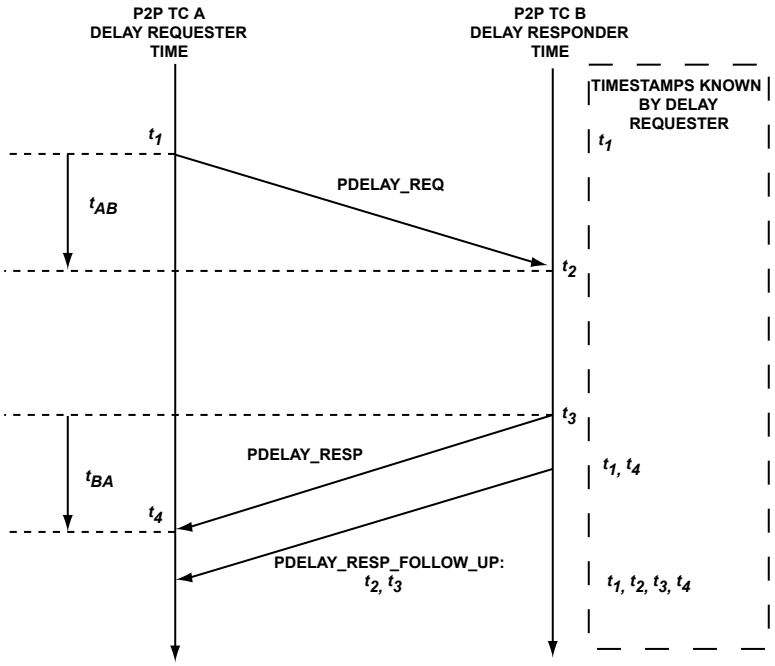

Figure 23-12: Propagation Delay Calculation between Nodes Supporting Peer-to-Peer Path Correction

As shown in the figure, the propagation delay is calculated in the following way:

1. Port-1 issues a Pdelay_Req message and generates a timestamp, t1, for the Pdelay_Req message.
2. Port-2 receives the Pdelay_Req message and generates a timestamp, t2, for this message.
3. Port-2 returns a Pdelay_Resp message and generates a timestamp, t3, for this message. To minimize errors due to frequency offset between the two ports, Port-2 returns the Pdelay_Resp message as quickly as possible after the receipt of the Pdelay_Req message. The Port-2 returns any one of the following:
   - The difference between the timestamps t2 and t3 in the Pdelay_Resp message.
   - The difference between the timestamps t2 and t3 in the Pdelay_Resp_Follow_Up message.
   - The timestamps t2 and t3 in the Pdelay_Resp and Pdelay_Resp_Follow_Up messages respectively.
4. Port-1 generates a timestamp, t4, on receiving the Pdelay_Resp message.

Port-1 uses all four timestamps to compute the mean link delay.

Block Diagram

The ***PTP Block Diagram*** figure shows the functional block diagram of PTP module.

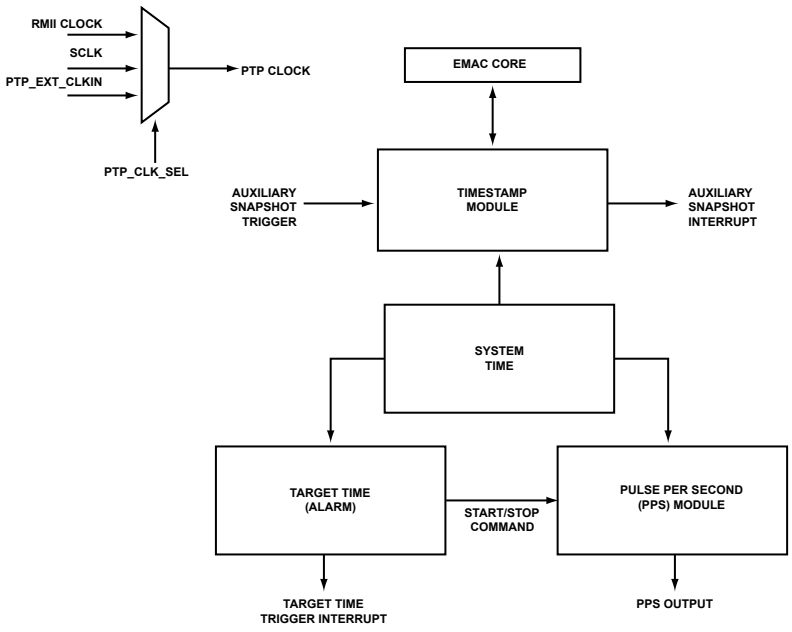

Figure 23-13: PTP Block Diagram

A system time module is present which keeps the time of PTP module. It consists of hardware which can be programmed for time initialization, time correction, and clock correction.

The time-stamp module captures the time (provided by the system time module) at various conditions. For example, when the EMAC sends or receives a frame or during the rising edge of the auxiliary snapshot trigger (`EMAC_PTPAUXIN[n]`) pins. When system time is captured after detection of a frame, the timestamp module automatically includes the time information in the frame descriptor. Time stamping on the detection of a frame can be programmed on a per frame basis.

The PTP clock drives the PTP module. This clock can be selected from three different clock sources.

The pulse per second (PPS) module generates a pulse or train of pulse on the PPS output pin, (`EMAC_PTPPPS[n]`) and it is the physical representation of system time. PPS can be fixed (where only frequency varies) or flexible (where width, interval, start time, and stop time can be programmed).

The target time module acts as an alarm for the PTP module. Whenever system time reaches a value equal to programmed target time, the target time trigger interrupt is generated. By appropriate programming, The target time trigger can also start or stop flexible PPS output at specific time.

PTP Module Clock

The PTP module clock features include Clock Source Selection and Clock Frequency Range.

Clock Source Selection

The PTP module can take one of these clock sources as its input clock — SYSCLK, RMII reference or PTP external clock.

As shown in the ***PTP Clock Source Selection*** table, the PADS_PCFG0 register selects the PTP clock source.

Table 23-33: PTP Clock Source Selection

PADS_PCFG0	PTP Clock Source	Clock Description
00	EMAC_RMII	RMII reference clock
10	PTP external clock	Clock available on EMAC_PTPCLKIN[n] pin
X1	SYSCLK	Processor system clock

Clock Frequency Range

The resolution, or granularity, of the reference time source determines the accuracy of the synchronization. Therefore, a higher PTP clock frequency gives better system performance. The timing constraints achievable for logic operating on the selected PTP clock source limit the maximum PTP clock frequency.

The minimum PTP clock frequency depends on the time required between two consecutive frames. The IEEE specification determines the RMII clock frequency. The minimum PTP clock frequency required for proper operation depends on the operating mode and operating speed of the MAC. See the ***Minimum PTP Clock Frequency*** table.

A minimum delay required between two consecutive timestamp captures is 8 clock cycles of RMII and 3 clock cycles of PTP clocks. If the delay between two timestamp captures is less than this delay, the EMAC does not take a timestamp snapshot for the second frame.

Table 23-34: Minimum PTP Clock Frequency

Mode	Minimum Gap Between Two Frames	Minimum PTP Frequency
100-Mbps full-duplex operation	336 RMII clocks (256 clocks for a 64-byte frame + 48 clocks of min IFG + 32 clocks of preamble)	(3 × PTP period) + (8 × RMII period) ≤ (336 × RMII period) Maximum PTP period = (336 – 8) × 20 ns ÷ 3 = 2,186 ns Minimum PTP frequency = 0.46 MHz

The minimum PTP clock frequency also depends on the maximum value of the EMAC_TM_SUBSEC register, so that even at the highest EMAC_TM_SUBSEC.SSINC value the EMAC_TM_SEC register value can be incremented every second. Since the EMAC_TM_SUBSEC.SSINC register contains an 8-bit field the minimum PTP clock frequency allowed is approximately 4 MHz.

Timestamp Module

The timestamp module captures time in seconds and nanoseconds maintained as system time. The timestamp module also captures time when specific events occur. Events include detection of a frame transmitted or received over the EMAC and a rising edge on the EMAC_PTPAUXIN[n] pin. The time-stamp module does not need to timestamp all of the transmitted or received frames over the EMAC. The PTP module can be programmed to detect specific kinds of frames for timestamping.

Frame Detection and Timestamping

The PTP module automatically monitors all received and transmitted IEEE 1588 event messages on the Ethernet. If the module detects an event message, it takes a snapshot of the system time. The PTP module stores the value to the 64-bit fields in transmit or receive descriptor.

The timestamping occurs at the EMAC RMII interface when the module sees the start of frame of an event message packet.

Transmit Path Timestamping

The EMAC captures a timestamp when a frame transmits on the RMII. Timestamp capture is controllable on a per-frame basis. In other words, each transmit frame can be marked to indicate whether a timestamp is captured for that frame or not.

Applications can extend the descriptor word length from 4 words to 8 words by setting the `EMAC_DMA_BUSMODE.ATDS` bit. To enable the timestamp function, set the TTSE (transmit timestamp enable) bit in transmit descriptor word TDES0. When the PTP module captures a timestamp of a transmitted frame, it notifies the application by setting the TTSS (transmit timestamp status) in TDES0.

The EMAC returns the timestamp to the software inside the corresponding transmit descriptor, automatically connecting the timestamp to the specific frame. The 64-bit timestamp information is written to the TDES6 and TDES7 fields. The TDES6 field holds the 32-bit LSBs of the timestamp (system time nanoseconds), except as described in transmit timestamp field and TDES7 field holds the 32-bit MSBs (system time seconds). After the PTP module timestamps the frame, the application can get the timestamp along with the transmit status from the EMAC.

NOTE: The PTP module timestamps all the transmitting frames having TTSE set in its TDES0. It does not distinguish according to the type of transmitting frame.

Receive Path Timestamping

The PTP module automatically monitors all received and transmitted IEEE 1588 event messages on the Ethernet. If an event message is detected, the module takes a snapshot of the system time. The module stores its value to the 64-bit fields in transmit or receive descriptor. The timestamping is done at the EMAC RMII interface when the module sees the start of frame of an event message packet.

PTP module captures the timestamp of received frames on the RMII. Timestamp capture is controllable on a per-frame and per-type basis. In other words, each received frame is timestamped according to the frame type.

Applications can extend the descriptor word length from 4 words to 8 words by setting `EMAC_DMA_BUSMODE.ATDS` to store timestamp and received message status. The PTP notifies the application of receive timestamp availability when it sets bit 7 (timestamp available) in receive descriptor word RDES0.

When bit 0 (extended status available) is set in RDES0, it indicates that the extended status of the PTP frame is provided in the RDES4 word. Extended status includes PTP version, PTP frame type, and message type. The EMAC returns the timestamp to the software inside the corresponding receive descriptor. The 64-bit timestamp

information is written back to the RDES6 and RDES7 fields in memory. The RDES6 holds the 32-bit LSBs of the timestamp (system time nanoseconds), except as mentioned in receive timestamp field and RDES7 field holds 32-bit MSBs (system time seconds).

The timestamp is written only to that receive descriptor for which the last descriptor status field has been set to 1. When the timestamp is not available (for example, because of an RxFIFO overflow), an all-ones pattern is written to the descriptors (RDES6 and RDES7). The write operation indicates that the timestamp is not correct. RDES0 [7] indicates whether the timestamp is updated in RDES6/7.

The PTP module processes received frames to identify valid PTP frames. Use the EMAC_TM_CTL register to control the snapshot of the time sent to the application.

The PTP module can be programmed to detect all received frames or only some types of PTP frames, according to bit settings in the EMAC_TM_CTL register. Refer to the ***PTP Frame Type Selections*** table.

Table 23-35: PTP Frame Type Selections

TSENALL (bit 8)	SNAPTYPSEL (bits [17:16])	TSMSTRENA (bit 15)	TSEVNTENA (bit 14)	Frames
1	X	X	X	All
0	00	X	0	Sync, Follow_Up, Delay_Req, Delay_Resp
0	00	0	1	Sync
0	00	1	1	Delay_Req
0	01	X	0	Sync, Follow_Up, Delay_Req, Delay_Resp, Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up
0	01	0	1	Sync, Pdelay_Req, Pdelay_Resp
0	01	1	1	Delay_Req, Pdelay_Req, Pdelay_Resp
0	10	X	X	Sync, Delay_Req
0	11	X	X	Pdelay_Req, Pdelay_Resp

PTP Processing and Control

When the EMAC receives a frame, frame detection and timestamping are based on some of the PTP fields in the frame. The ***PTP Message Format (IEEE 1588–2008)*** table shows the common message header for the PTP messages. This format is derived from the IEEE standard 1588–2008. When the EMAC sends a PTP frame, the frame follows this format.

When a frame is received, the PTP module compares these fields with standard values and finds out the type of PTP frame and other information. For example, PTP version, IP version, and others. The module then updates the

related fields in RDES4. When a frame is transmitted, programs must ensure that all the fields are appropriate. The PTP module on the other end of a communication must correctly detect and decode the frame.

Table 23-36: PTP Message Format (IEEE 1588–2008)

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
transportSpecific				messageType				1	0
Reserved				versionPTP				1	1
messageLength								2	2
domainNumber								1	4
Reserved								1	5
flagField								2	6
correctionField								8	8
Reserved								4	16
sourcePortIdentity								10	20
sequenceId								2	30
controlField (used in version 1. For version 2, messageType field is used for detecting different message types.)								1	32
logMessageInterva								1	33

There are some fields in the Ethernet payload that can be used to detect the PTP packet type and control the snapshot taken. These fields are different for the following PTP frames:

- PTP Frames Over IPv4
- PTP Frames Over IPv6
- PTP Frames Over Ethernet

For these PTP frames, EMAC does not consider the PTP version 1 messages as valid when the frame consists of Peer delay multicast address as destination address (DA).

PTP Frame Over IPv4

The ***IPv4-UDP PTP Frame Fields Required for Control and Status*** table provides information about the fields that are matched to control snapshot for the PTP packets. The packets are sent over UDP over IPv4 for IEEE 1588 version 1 and 2. The octet positions for the tagged frames are offset by 4. The positions are based on IEEE 1588–2008 standards and the message format. The format is defined in the ***PTP Message Format (IEEE 1588–2008)*** table in the PTP Processing and Control section.

Table 23-37: IPv4-UDP PTP Frame Fields Required for Control and Status

Field Matched	Octet Position	Matched Value	Description
MAC Frame type	12, 13	0x0800	IPv4 datagram
IP Version and Header Length	14	0x45	IP version is IPv4
Layer 4 Protocol	23	0x11	UDP
IP Multicast Address (IEEE 1588 Version 1)	30, 31, 32, 33	0xE0,0x00, 0x01,0x81 (or 0x82 or 0x83 or 0x84)	Multicast IPv4 addresses allowed. 224.0.1.129 224.0.1.130 224.0.1.131 224.0.1.132
IP Multicast Address (IEEE 1588 Version 2)	30, 31, 32, 33	0xE0, 0x00, 0x01, 0x81 (Hex) 0xE0, 0x00, 0x00, 0x6B (Hex)	PTP-Primary multicast address: 224.0.1.129 PTP-Peer delay multicast address: 224.0.0.107
UDP Destination Port	36, 37	0x013F 0x0140	0x013F - PTP Event Messages. These are SYNC, Delay_Req (IEEE 1588 version 1 and 2) or Pdelay_Req, Pdelay_Resp (IEEE 1588 version 2 only). 0x0140 - PTP general messages
PTP Control Field (IEEE version 1)	74	0x00/0x01/0x02/ 0x03/0x04	0x00 - SYNC 0x01 - Delay_Req 0x02 - Follow_Up 0x03 - Delay_Resp 0x04 - Management
PTP Message Type Field (IEEE version 2)	42 (nibble)	0x0/0x1/0x2/0x3/0x8/0x9/0xA/0xB/ 0xC/0xD	0x0 - SYNC 0x1 - Delay_Req 0x2 - Pdelay_Req 0x3 - Pdelay_Resp 0x8 - Follow_Up 0x9 - Delay_Resp 0xA - Pdelay_Resp_Follow_Up 0xB - Announce 0xC - Signaling 0xD - Management
PTP Version	43 (nibble)	0x1 or 0x2	0x1 - Supports PTP version 1 0x2 - Supports PTP version 2

PTP Frame Over IPv6

The ***IPv6-UDP PTP Frame Fields Required for Control And Status*** table provides information about the fields that are matched to control the snapshots for the PTP packets. The packets are sent over UDP over IPv6 for IEEE 1588 version 1 and 2. The octet positions for the tagged frames are offset by 4. The positions are based on IEEE 1588–2008 standards and the message format defined in PTP Message Format (IEEE 1588–2008).

Table 23-38: IPv6-UDP PTP Frame Fields Required for Control and Status

Field Matched	Octet Position	Matched Value	Description
MAC Frame type	12, 13	0x86DD	IP datagram
IP Version	14 (bits [7:4])	0x06	IP version is IPv6
Layer 4 Protocol	20 (IPv6 extension header not defined for PTP packets)	0x11	UDP
PTP Multicast Address	38–53	FF0x:0:0:0:0:0:0:0:181 (Hex) FF02:0:0:0:0:0:0:0:6B (Hex)	PTP - primary multicast address: FF0x: 0:0:0:0:0:0:0:181 (Hex) PTP - Peer delay multicast address: FF02:0:0:0:0:0:0:0:6B (Hex)
UDP Destination Port	56, 57 (IPv6 extension header not defined for PTP packets)	0x013F, 0x0140	0x013F - PTP event messages 0x0140 - PTP general messages
PTP Control Field (IEEE 1588 version 1)	93 (IPv6 extension header not defined for PTP packets)	0x00/0x01/0x02/ 0x03/0x04	0x00 - SYNC 0x01 - Delay_Req 0x02 - Follow_Up 0x03 - Delay_Resp 0x04 - Management (version1)
PTP Message Type Field (IEEE version 2)	74 (nibble) (IPv6 extension header not defined for PTP packets)	0x0/0x1/0x2/0x3/0x8/0x9/0xA/0xB/ 0xC/0xD	0x0 - SYNC 0x1 - Delay_Req 0x2 - Pdelay_Req 0x3 - Pdelay_Resp 0x8 - Follow_Up 0x9 - Delay_Resp 0xA - Pdelay_Resp_Follow_Up 0xB - Announce 0xC - Signaling 0xD - Management
PTP Version	75 (nibble)	0x1 or 0x2	0x1 - Supports PTP version 1 0x2 - Supports PTP version 2

PTP Frame Over Ethernet

Refer to the ***Ethernet PTP Frame Fields Required for Control and Status*** table. The table provides information about the fields that are matched to control the snapshots for the PTP packets sent over Ethernet for IEEE 1588 version 1 and 2. The octet positions for the tagged frames are offset by 4. The positions are based on IEEE 1588–2008 standards and the message format defined in the table.

Table 23-39: Ethernet PTP Frame Fields Required for Control and Status

Field Matched	Octet Position	Matched value	Description
MAC Destination Multicast Address (The address match of destination address (DA) programmed in MAC address 0 is used when the EMAC_TM_CTL.TSENMACADDR bit is set)	0–5	01-1B-19-00-00-00 01-80-C2-00-00-0E	All PTP messages can use any of the following multicast addresses: 01-1B-19-00-00-00 01-80-C2-00-00-0E
MAC Frame Type	12, 13	0x88F7	PTP Ethernet frame
PTP control field (IEEE Version 1)	45	0x00/0x01/0x02/ 0x03/0x04	0x00 - SYNC 0x01 - Delay_Req 0x02 - Follow_Up 0x03 - Delay_Resp 0x04 - Management
PTP Message Type Field (IEEE version 2)	14 (nibble)	0x0/0x1/0x2/0x3/0x8/0x9/0xA/0xB/ 0xC/0xD	0x0 - SYNC 0x1 - Delay_Req 0x2 - Pdelay_Req 0x3 - Pdelay_Resp 0x8 - Follow_Up 0x9 - Delay_Resp 0xA - Pdelay_Resp_Follow_Up 0xB - Announce 0xC - Signaling 0xD - Management
PTP Version	15(nibble)	0x1 or 0x2	0x1 - Supports PTP version 1 0x2 - Supports PTP version 2

Auxiliary Timestamp Snapshot

The auxiliary snapshot feature stores snapshots of the system time whenever a rising edge is detected on the EMAC_PTPAUXIN[n] pins.

The PTP stores 64 bits of captured timestamp in a 4-deep FIFO. When a snapshot is stored, the PTP indicates this event to the EMAC with the auxiliary snapshot interrupt. The EMAC_TM_STMPSTAT.ATSTS bit is set. The value of

the snapshot is read through the EMAC_TM_AUXSTMP_SEC and EMAC_TM_AUXSTMP_NSEC registers. If the FIFO becomes full and an external trigger to take the snapshot is asserted, then the snapshot trigger-missed status is set in the EMAC_TM_STMPSTAT.ATSSTM bit. The latest snapshot is not written to the FIFO when it is full.

When a host reads the 64-bit timestamp from the FIFO through the EMAC_TM_AUXSTMP_SEC and EMAC_TM_AUXSTMP_NSEC registers, the space becomes available to store the next snapshot.

NOTE: A space in the FIFO is created whenever the EMAC_TM_AUXSTMP_SEC register is read. Therefore, read the EMAC_TM_AUXSTMP_NSEC register before reading the EMAC_TM_AUXSTMP_SEC register.

The program can clear the FIFO by setting the EMAC_TM_CTL.ATSFC bit. When multiple snapshots are present in the FIFO, the EMAC_TM_STMPSTAT.ATSNS bits indicate the count.

NOTE: The minimum gap between two events on the EMAC_PTPAUXIN[n] pin must be 4 cycles of PTP_CLK + 3 cycles of SYSCLK). Otherwise, the logic misses the rising-edge of the trigger.

System Time

To get a snapshot of the time, the EMAC requires a reference time in 64-bit format as defined in the IEEE 1588 specification. The PTP module maintains 80-bit time, known as system time. The PTP clock updates system time.

The 80-bit timing reference is split into the following three registers:

- EMAC_TM_NSEC – 32-bit nanoseconds register which provides time in nanoseconds
- EMAC_TM_SEC – 32-bit seconds register which provides time in seconds
- EMAC_TM_HISEC – 16-bit high seconds register which provides time beyond the seconds register. The IEEE 1588 standard does not include this register. Its use is application-specific.

The 64-bit system time (seconds and nanoseconds) is the source for taking timestamps for Ethernet frames being transmitted or received at the RMII.

Since the PTP clock frequency does not correspond to a 1ns period, the EMAC_TM_NSEC register is incremented with a value equal to the PTP clock period for every PTP clock cycle. The function uses the EMAC_TM_SUBSEC register. The EMAC_TM_NSEC value is incremented with the value programmed in EMAC_TM_SUBSEC register every PTP clock cycle.

Whenever the EMAC_TM_SEC register overflows from 0xFFFFFFFF to 0x00000000, the seconds overflow interrupt is triggered. The EMAC uses the EMAC_TM_STMPSTAT.TSSOVF bit to indicate the event. After a seconds overflow, the EMAC_TM_HISEC register increments by one.

The system time module supports the following two types of rollover modes for the EMAC_TM_NSEC register.

- Digital rollover mode. The maximum value in the nanoseconds field is 0x3B9AC9FF, that is, 10^9 nanoseconds. After it reaches this value, the EMAC_TM_SEC register increments and the EMAC_TM_NSEC register restarts counting from zero. Accuracy in digital rollover mode it is 1 ns per bit.

- Binary rollover mode. The nanoseconds field rolls over and increments the seconds field after the value reaches 0x7FFFFFFF. Accuracy in binary rollover mode is ~0.466 ns per bit.

System Time Adjustment

The following sections describe the process for system time adjustment.

System Time Initialization

System time can be initialized with 64-bit time when the PTP module is enabled. The initial value is written to the EMAC_TM_SECUPDT and EMAC_TM_NSECUPDT system time update registers. The system time counter is written with the value in the registers when the EMAC_TM_CTL.TSINIT bit is set.

Coarse Correction Method

If slave system time has an offset based on the system time of the master, then the coarse correction method can correct it. The time offset value is written to the EMAC_TM_SECUPDT and EMAC_TM_NSECUPDT registers. The offset value is then added to or subtracted from the system time when the EMAC_TM_CTL.TSUPDT bit is set. Use the EMAC_TM_NSECUPDT.ADDSUB bit to choose addition or subtraction. System time correction occurs in one clock cycle using the coarse correction method.

NOTE: During subtraction, the EMAC_TM_SECUPDT register value must be less than the value of the EMAC_TM_SEC register. Check the value prior to subtracting using coarse correction.

Fine Correction Method

If a slave PTP clock frequency has a drift based on the master PTP clock (as defined in IEEE 1588), it can be corrected using the fine correction method. Using this method, system time is corrected over a period (unlike coarse correction where it happens in one clock cycle). This correction helps maintain linear time and does not introduce drastic changes (or a large jitter) in the reference time between PTP sync message intervals.

Using this method, an accumulator sums the contents of the EMAC_TM_ADDEND register. The ***System Time Update, Fine Correction Method*** figure shows the method. The arithmetic-carry that the accumulator generates acts as a pulse to increment the system time counter. The accumulator and the addend are 32-bit registers. Here, the accumulator acts as a high-precision frequency divider.

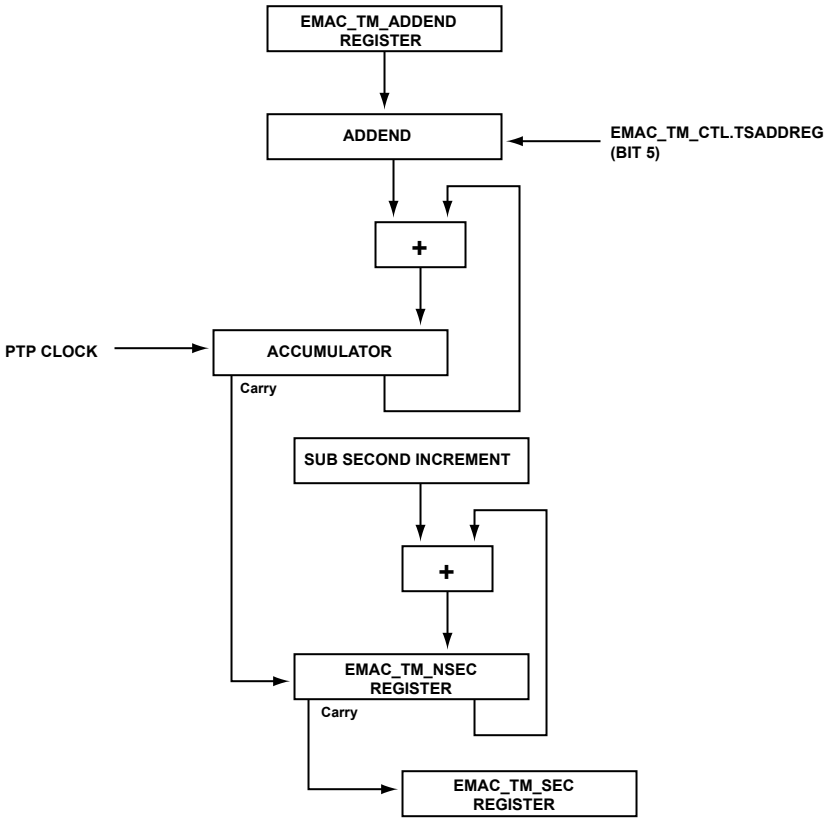

Figure 23-14: System Time Update, Fine Correction Method

Calculating Addend Value

This section describes the fine correction process for system time.

In this example, the master clock runs at 50 MHz and the slave clock has drifted to 66 MHz. The goal is to adjust the slave system time to 50 MHz, so that the slave PTP module synchronizes with the master. Using the figure in Fine Correction Method, the nanoseconds increment signal runs at 50 MHz. The nanoseconds increment is the carry from accumulator register. The addend value increments the carry from accumulator at the rate of the slave clock (66 MHz).

The accumulator overflows and generates a carry every N addend value, so N × Addend = 2^{32}.

The accumulator increments at 66 MHz. This addition brings the carry to 50 MHz N = 66/50 = 1.32.

Hence, the addend = 2^{32}/1.32 = 0xC1F07C1F.

Therefore, if addend is programmed with 0xC1F07C1F, the slave system time runs at 50 MHz which synchronizes with the master.

In the Fine Correction Method figure, the subsecond increment is the value programmed in the `EMAC_TM_SUBSEC` register which increments the `EMAC_TM_NSEC` register according to the frequency of the nanoseconds increment signal.

In the example, the sub second increment is 20 (for digital rollover) or 43 (for binary rollover). This addition increments the EMAC_TM_NSEC register by 20 ns (1/50 MHz).

The software must calculate the drift in frequency and update the EMAC_TM_ADDEND register accordingly.

NOTE: The PTP reference clock is the clock at which the system time is updated. When the EMAC_TM_CTL.TSCFUPDT bit is set to 0, this clock equals the PTP clock. Using fine correction, the PTP reference clock is generated on the nanoseconds increment signal at which the system time updates.

Target Time Trigger (Alarm)

The PTP module provides an alarm function by triggering an alarm at a preset time. It sets the EMAC_TM_STMPSTAT.TSTARGT bit when the system time matches the value of the EMAC_TM_TGTM and EMAC_TM_NTGTM registers. This trigger can generate an interrupt and command the flexible PPS module to start or stop PPS output, depending on value programmed in EMAC_TM_PPSCTL.TRGTMODSEL bits.

The trigger is enabled by setting EMAC_TM_CTL.TSTRIG bit. Once an alarm has occurred, if the PTP needs another alarm, the software must:

- Clear the status bit
- Reprogram the EMAC_TM_TGTM and EMAC_TM_NTGTM registers to a future value, and
- Set the EMAC_TM_CTL.TSTRIG bit

If the time programmed in the target time registers has elapsed, then a target time programming error is indicated by setting the EMAC_TM_STMPSTAT.TSTRGTERR bit.

The alarm time is represented in absolute units, not relative units. For example, if the software must generate an alarm after 10 seconds, it must read the current system time value. Then, the software must add the number corresponding to 10 seconds, and write the result back to the target time registers.

Pulse-Per-Second (PPS)

Pulse-per-second (PPS) is a physical representation of system time. It consists of a single pulse or train of pulses. The PTP uses PPS for extra synchronization or to monitor the synchronization performance between clocks. With proper configuration, the PTP module can generate PPS signals that are output on the EMAC_PTPPPS[n] pins. The PTP supports two kinds of PPS output, fixed and flexible.

Fixed Pulse-Per-Second Output

The EMAC supports fixed pulse-per-second (PPS) output that indicates 1-second intervals (default). Change the frequency of the PPS output by configuring the EMAC_TM_PPSCTL.PPSCTL bits. The default value for these bits is 0000, which configures a 1Hz signal with a pulse width equal to the period of the PTP clock.

The ***PPS Output Frequencies*** table shows various PPS output frequencies.

Table 23-40: PPS Output Frequencies

PPSCTL Bit Setting	Binary Rollover	Digital Rollover
0001	2 Hz	1 Hz
0010	4 Hz	2 Hz
0011	8 Hz	4 Hz
...	...	...
1111	32.768 kHz	16.384 kHz

In binary rollover mode, the PPS output has a duty cycle of 50% with these frequencies.

In digital rollover mode, the PPS output frequency is an average number. The actual clock is a different frequency that is synchronized every second. PPS output pulses have different periods and duty cycles and this behavior is because of the non-linear toggling of the bits in digital rollover mode. For example:

- When `EMAC_TM_PPSCTL.PPSCTL` = 0001, the PPS (1 Hz) has a low period of 537 ms and a high period of 463 ms.
- When `EMAC_TM_PPSCTL.PPSCTL` = 0010, the PPS (2 Hz) is a sequence of:
  - One clock of 50-percent duty cycle and 537-ms period
  - Second clock of 463-ms period (268 ms low and 195 ms high)
- When `EMAC_TM_PPSCTL.PPSCTL` = 0011, the PPS (4 Hz) is a sequence of:
  - Three clocks of 50-percent duty cycle and 268-ms period
  - Fourth clock of 195-ms period (134 ms low and 61 ms high)

Flexible Pulse-Per-Second Output

The EMAC also provides the flexibility to program the start or stop time, width, and interval of the pulse generated on the PPS output. Enable this feature, called flexible PPS, by setting the `EMAC_TM_PPSCTL.PPSEN` bit.

The flexible PPS output options are:

- Supports programming the start point of the single pulse and start and stop points of the pulse train in terms of system time. The target time registers program the start and stop time.
- Supports programming the stop time in advance. Programs can configure the stop time before the actual start time has elapsed.
- Supports programming the width, between the rising edge and corresponding falling edge of the PPS signal output, in terms of number of units of subsecond increment. This value is configured in the `EMAC_TM_SUBSEC` register.
- Supports programming the interval, between the rising edges of PPS signal, in terms of number of units of subsecond increment. This value is configured in the `EMAC_TM_SUBSEC` register.

- Provides the option to cancel the programmed PPS start or stop request.
- Indicates error if the start or stop time programmed has already elapsed.

PPS Start or Stop Time

Start time can initially be programmed in the target time registers. If necessary, the start or stop time can be programmed again but only after the earlier programmed value is synchronized to the PTP clock domain. The EMAC_TM_NTGTM.TSTRBUSY bit indicates the status of synchronization. Programs can configure the start or stop time in advance, even before the earlier stop or start time has elapsed.

Program the start or stop time with advanced system time to ensure proper PPS signal output. If the application programs a start or stop time that has already elapsed, then the EMAC sets the EMAC_TM_STMPSTAT.TSTRGTERR bit, indicating the error. If enabled, the EMAC also sets the target time trigger (alarm) interrupt event. The application can cancel the start or stop request only if the corresponding start or stop time has not elapsed. If the time has elapsed, the cancel command has no effect.

PPS Width and Interval

The PPS width and interval are programmed in terms of granularity of system time, that is, the number of the units of subsecond increment value. For example, with the PTP reference clock of 50 MHz, a PPS pulse width of 40 ns, and an interval of 100 ns, program the width and interval to 2 and 5, respectively.

Use a faster PTP reference clock to achieve smaller granularity. Before commanding to trigger a pulse or pulse train on the PPS output, programs must configure or update the interval and width of the PPS signal output.

PPS Command

When the PPS module has a flexible PPS output configuration, the PTP can use the EMAC_TM_PPSCTL.PPSCTL bits to command the PPS module to use any of the flexible PPS features.

Programming these bits with a non-zero value instructs the PPS module to initiate an event. Once the command transfers or synchronizes to the PTP clock domain, these bits clear automatically. Software must ensure that these bits are programmed only when they are all-zero.

The ***Flexible PPS Output Commands*** table explains the different commands and their function.

Table 23-41: Flexible PPS Output Commands

PPSCTL (Bits 3–0)	Command	Description
0000	No Command	
0001	Start Single Pulse	Generates single pulse rising at start point defined in target time registers and of duration defined in EMAC_TM_PPSWIDTH register.
0010	Start Pulse train	Generates train of pulses rising at the start time configured in the Target Time registers, of duration configured in the EMAC_TM_PPSWIDTH register. The train of pulses repeats at the interval configured in the EMAC_TM_PPSINTVL register. By default, the PPS

Table 23-41: Flexible PPS Output Commands (Continued)

PPSCTL (Bits 3–0)	Command	Description
		pulse train is free-running unless stopped by stop pulse train at time or stop pulse train immediately commands.
0011	Cancel Start	Cancels the start single pulse and start pulse train commands when the system time has not crossed the programmed start time.
0100	Stop Pulse train at time	Stops the train of pulses initiated by the command for start pulse train after the time programmed in the target time registers elapses.
0101	Stop Pulse train immediately	Immediately stops the train of pulses initiated by the command for start pulse train.
0110	Cancel Stop Pulse train	Cancels the stop pulse train at time command when the programmed stop time has not elapsed. The PPS pulse train becomes free-running on the successful execution of this command.
0111-1111	Reserved	

PTP Interrupts

Set the EMAC_IMSK.TS bit to enable interrupts from the PTP module. The EMAC uses the EMAC_ISTAT.TS bit to indicate the status of the interrupt. The PTP supports the following three types of interrupts.

Auxiliary Snapshot Trigger

When an external event occurs on the EMAC_PTPAUXIN[n] pins and a timestamp snapshot occurs, an auxiliary snapshot interrupt is triggered. The EMAC uses the EMAC_TM_STMPSTAT.ATSTS bit to indicate the interrupt.

Target Time Reached

This interrupt is triggered when the system time becomes equal to the value written in the EMAC_TM_TGTMand EMAC_TM_NTGTM registers. Enable or disable the interrupt using the EMAC_TM_CTL.TSTRIG and EMAC_TM_PPSCTL.TRGTMODSEL bits. This interrupt can be used as an alarm and is indicated on the EMAC_TM_STMPSTAT.TSTARGT bit.

System Time Seconds Register Overflow

This interrupt is triggered when the EMAC_TM_SEC register overflows from 0xFFFF FFFF to 0x0000 0000. This interrupt is indicated on the EMAC_TM_STMPSTAT.TSSOVF bit. As soon as EMAC_TM_SEC register overflows, the EMAC_TM_HISEC register increments by one.

EMAC Event Control

The EMAC has a dedicated interrupt signal registered with the system event controller (SEC) of the processor. Various interrupt sources within EMAC peripheral are shared through this interrupt line. Refer to the "System Event Controller" chapter for details on how interrupts work in this product and how to configure them.

EMAC Interrupt Signals

Interrupts from the EMAC are triggered from the EMAC DMA layer or the EMAC CORE layer. Interrupts are triggered from EMAC DMA when a particular status bit is set in the EMAC_DMA_STAT register. An interrupt line is asserted only when the corresponding bits are enabled in the DMA interrupt enable register. Similarly, interrupts are triggered from the EMAC CORE when a particular MMC status bit or PTP status bit is set in the interrupt status register.

An interrupt line is asserted only when the corresponding bits are enabled in the MMC mask registers for MMC counters or the interrupt mask register for PTP. DMA status register also reflects the MMC interrupt status. The following lists show the two groups of interrupts in the DMA status register.

NIS – Normal Interrupt source summary:

- Transmit Interrupt
- Transmit Buffer Unavailable
- Receive Interrupt
- Early Receive Interrupt

AIS – Abnormal Interrupt source summary:

- Transmit Process Stopped
- Transmit Jabber Timeout
- Receive FIFO Overflow
- Transmit Underflow
- Receive Buffer Unavailable
- Receive Process Stopped
- Receive Watchdog Timeout
- Early Transmit Interrupt
- Fatal Bus Error

The EMAC generates an interrupt only once for simultaneous, multiple events. The driver must read the EMAC_DMA_STAT register for the cause of the interrupt. It can generate a new interrupt once the driver has cleared the appropriate bit in DMA status register.

For example, the controller generates a receive interrupt (EMAC_DMA_STAT.RI bit) and the driver begins reading the EMAC_DMA_STAT register. Next, a receive buffer unavailable interrupt (EMAC_DMA_STAT.RU bit) occurs. The driver clears the EMAC_DMA_STAT.RI bit but the internal interrupt signal is not deasserted, because of the active or pending EMAC_DMA_STAT.RU interrupt. The driver must scan all of the descriptors, from the last recorded position to the first one owned by the DMA, to know which descriptor has asserted the interrupt.

Interrupts are cleared by writing a 1 to the corresponding bit position in the EMAC_DMA_STAT register. When all the enabled interrupts within a group are cleared, the corresponding summary bit is cleared.

An interrupt delay timer provides (receive interrupt watchdog timer register) flexible control of the receive interrupt.

When the interrupt timer is programmed with a non-zero value, it is activated as soon as the RxDMA transfers a received frame to system memory. The transfer occurs without asserting the receive interrupt. This interrupt is not enabled in the corresponding receive descriptor (RDES1[31] in the receive DMA descriptors).

When this timer runs out (per the programmed value), the EMAC_DMA_STAT.RI bit is set. The interrupt is asserted when the corresponding EMAC_DMA_STAT.RI bit is enabled in the interrupt enable register. The timer is disabled before it runs out, when a frame is transferred to memory and when the EMAC_DMA_STAT.RI bit is set because it is enabled for that descriptor.

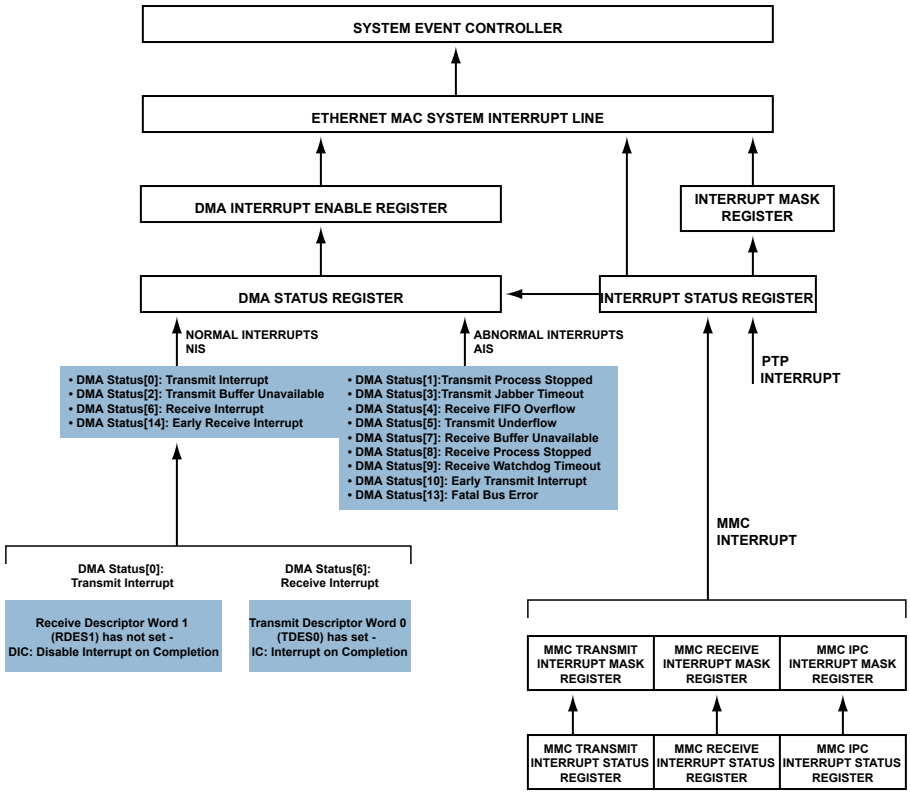

Figure 23-15: EMAC Interrupt Flow Diagram

PHYINT Interrupt Signal

A PHY device can notify the EMAC when it detects changes to the link status, such as auto-negotiation or a duplex-mode change. The external PHY chip typically includes an interrupt generation pin to aid this status change notification to the MAC. This signal is typically called PHYINT. A failing or rising edge on this signal can detect a PHY interrupt at the EMAC.

In the processor, any of the GPIO pins can be a used as a PHYINT signal. Use the following procedure to configure a GPIO as a PHYINT signal.

1. Program the GPIO to detect a falling or rising edge sensitive interrupt.
2. Program the PHY to generate the interrupt on a signal status change.
3. If PHYINT is asserted, read the PHY status register through the station management interface.

NOTE: The PHYINT is not part of EMAC module. However, any GPIO pin can be configured to interrupt the processor when a rising edge generated by PHY is detected.

Refer to the PORT chapter for more info on configuring GPIO pins for input.

EMAC Programming Model

This section provides the programming model of Ethernet MAC peripheral for developers.

EMAC Programming Steps

The following sections provide some general programming information

DMA Initialization

Use the following procedure to initialize DMA.

1. Perform a software reset by setting the `EMAC_DMA_BUSMODE.SWR` bit. This action resets all of the EMAC internal registers and logic.
2. Wait for the completion of the reset process by polling the `EMAC_DMA_BUSMODE.SWR` bit which is only cleared (automatically) after the reset operation completes.
3. Poll the `EMAC_DMA_BMSTAT.BUSRD` and `EMAC_DMA_BMSTAT.BUSWR` bits to confirm that all previously initiated (before software-reset) or ongoing SCB transactions are complete.
4. Program the required fields in the `EMAC_DMA_BMMODE` register:
    a. Address aligned bursts
    b. Fixed burst or undefined burst
    c. Burst length values and burst mode values
    d. Descriptor length (only valid when using ring mode)
5. Program the SCB interface options in the `EMAC_DMA_BMMODE` register. If fixed burst-length is enabled, then select the maximum burst-length possible on the SCB bus (bits `EMAC_DMA_BMMODE.BLEN4`, `EMAC_DMA_BMMODE.BLEN8`, `EMAC_DMA_BMMODE.BLEN16`).
6. Create a proper descriptor chain for transmit and receive. In addition, ensure that the DMA owns the receive descriptors (the `OWN` bit of the descriptor is set). For OSF mode, create at least two descriptors.

7. Ensure that the software creates three or more different transmit or receive descriptors in the chain before reusing any of the descriptors.
8. Initialize the EMAC_DMA_RXDSC_CUR and EMAC_DMA_TXDSC_CUR registers with the base address of the receive and transmit descriptors respectively.
9. Program the required fields in the EMAC_DMA_OPMODE register to initialize the mode of operation as follows:
    a. Receive and transmit store and forward
    b. Receive and transmit threshold control
    c. Error Frame and undersized good frame forwarding enable
    d. OSF mode
10. Clear the interrupt requests by writing to those bits of the EMAC_DMA_STAT register (interrupt bits only) that are set. For example, by writing 1 into bit 16, the normal interrupt summary clears this bit.
11. Enable the interrupts by programming the EMAC_DMA_IEN register.
12. Start the receive and transmit DMA by setting the EMAC_DMA_OPMODE.SR and EMAC_DMA_OPMODE.ST bits.

EMAC CORE Initialization

Use the following procedure to initialize the EMAC core.

1. Program the EMAC management address register (EMAC_SMI_ADDR) for controlling the management cycles for external PHY. For example, physical layer address (EMAC_SMI_ADDR.PA). In addition, set the EMAC_SMI_ADDR.SMIB bit for writing into PHY and reading from PHY.
2. Read the 16-bit data of management address register (EMAC_SMI_DATA) from the PHY for link up, speed of operation, and mode of operation. Specify the appropriate address value in the EMAC_SMI_ADDR.PA bit field.
3. Program the MAC address in the EMAC_ADDR0_HI and EMAC_ADDR0_LO registers.
4. If using hash filtering, program the hash table high and low registers register (EMAC_HASHTBL_HI, EMAC_HASHTBL_LO).
5. Program the fields to set the appropriate filters for the incoming frames in the MAC frame filter register (EMAC_MACFRMFILT):
    a. Receive all.
    b. Promiscuous mode.
    c. Hash or perfect filter.
    d. Unicast, multicast, broadcast, and control frames filter settings.
6. Program the fields for proper flow control in flow control register (EMAC_FLOWCTL):
    a. Pause time and other pause frame control bits.

b. Receive and transmit flow control bits.

c. Flow control busy or backpressure activate.

7. Program the EMAC interrupt mask register bits (EMAC_IMSK), as needed.
8. Program the appropriate fields in MAC configuration register (EMAC_MACCFG). For example, inter-frame gap while transmission and jabber disable. Based on the auto-negotiation desired, set the duplex mode (EMAC_MACCFG.DM bit) or speed select (EMAC_MACCFG.FES bit).
9. Set the transmit enable (EMAC_MACCFG.TE) and receive enable (EMAC_MACCFG.RE) bits.

Performing Normal Transmit and Receive Operations

For normal transmit and receive interrupts, the program must first read the interrupt status.

1. Poll the descriptors, reading the status of the descriptor owned by the application (either transmit or receive).
2. Set the appropriate values for the descriptors, ensuring that transmit and receive descriptors are DMA descriptors to resume the transmission and reception of data.

   ADDITIONAL INFORMATION: If the descriptors are not DMA (or no descriptor is available), the DMA goes into SUSPEND state.
3. Write a 0 into the Tx/Rx poll demand registers (EMAC_DMA_TXPOLL and EMAC_DMA_RXPOLL).

   Transmit or receive operations are resumed by freeing the descriptors and issuing a poll demand.
4. Read (for the debug process), the values of the current host transmitter or receiver descriptor address pointer (EMAC_DMA_TXDSC_CUR, EMAC_DMA_RXDSC_CUR) registers.
5. Read (for the debug process), the values of the current host transmit buffer address pointer and receive buffer address pointer (EMAC_DMA_TXBUF_CUR, EMAC_DMA_TXBUF_CUR) registers.

Stopping and Starting Transfers

Use the following procedure to stop and start EMAC transfers.

1. Disable the transmit DMA (if applicable), by clearing the EMAC_DMA_OPMODE.ST bit.
2. Wait for any previous frame transmissions to complete. Check for completion by reading the appropriate bits of the debug register (EMAC_DBG).
3. Disable the MAC transmitter and MAC receiver by clearing the EMAC_MACCFG.TE and EMAC_MACCFG.RE bits.
4. Disable the receive DMA (if applicable), after ensuring that the data in the receive FIFO transfers to the system memory by reading the EMAC_DBG register.
5. Make sure that both the transmit and receive FIFOs are empty.

6. To restart the operation, first start the DMA, and then enable the MAC transmitter and receiver.

Interrupts and Interrupt Service Routines

The following procedure describes specific steps for enabling interrupts and using their ISRs.

This procedure is typically performed with EMAC and DMA initialization and operations.

1. Receive interrupts are enabled for descriptors by default. Enable transmit interrupts for individual descriptors by setting the IC bit (bit 30) in the TDES0 word of the transmit descriptor.
2. Enable the required bits in the DMA interrupt enable register (EMAC_DMA_IEN).

   ADDITIONAL INFORMATION: Setting the EMAC_DMA_IEN.NIS or EMAC_DMA_IEN.AIS bits can turn on the occurrence of all normal or abnormal interrupt conditions. Individual conditions can also be enabled on using individual bits in the EMAC_DMA_IEN register.
3. Enable MMC overflow interrupts by setting appropriate bits in the EMAC_MMC_RXIMSK and EMAC_MMC_TXIMSK registers.
4. Enable PTP interrupts by setting the EMAC_IMSK.TS bit.
5. Once an EMAC interrupt is asserted and the SEC branches execution to the EMAC ISR, perform the following software program sequence.
   a. Read DMA status from the EMAC_DMA_STAT register.
   b. Clear the interrupt source by writing 1 (W1C) to the bits that are set in the EMAC_DMA_STAT register.
   c. Check for normal/abnormal/mmc/ptp interrupts by parsing the status bits read earlier, and call the appropriate service function.

      ADDITIONAL INFORMATION: Normal interrupt assertions include the transmit and receive interrupt. Abnormal interrupt assertions include the receive underflow.
6. The MMC handler functions use the following sequence.
   a. Read the EMAC_ISTAT register and parse for the EMAC_ISTAT.MMCTX and EMAC_ISTAT.MMCRX bits to determine if the interrupt is a transmit counter or receive counter-interrupt.
   b. Read the EMAC_MMC_RXINT or EMAC_MMC_TXINT registers to determine which of the counters have triggered the interrupt.
   c. Read the respective MMC counter that caused the interrupt to clear it.
7. PTP handler functions use the following sequence:
   a. Read the EMAC_ISTAT.TS bit to determine if a PTP interrupt occurred.
   b. Read EMAC_TM_STMPSTAT register to determine the interrupt source by parsing the EMAC_TM_STMPSTAT.ATSTS, EMAC_TM_STMPSTAT.TSTARGT, and EMAC_TM_STMPSTAT.TSSOVF bits.

c. Clear the interrupt source by reading the EMAC_TM_STMPSTAT register.

Enabling Checksum for Transmit and Receive

Use the following steps to enable transmit and receive checksums.

Enabling receive and transmit checksums is typically performed with EMAC and DMA initialization and operations. Transmit and receive checksum features are independent of each other.

1. To enable transmit checksum insertion:
   a. Enable store-and-forward mode in the FIFO by setting the EMAC_DMA_OPMODE.TSF bit.
   b. Ensure that the transmit frame can be contained within the 256 byte Tx FIFO conforming to the size rule: FIFO Depth – PBL – 3 FIFO locations, where PBL is burst length.
   c. Program the following necessary parameters for transmit checksum, by programming (CIC) checksum insertion control in TDES0: IP header checksum, IP header checksum and payload checksum, IP header checksum, payload checksum, and pseudo header checksum.

   A higher layer such as the IP stack sends out the packet to the EMAC which inserts the checksum as configured.
2. To enable receive checksum verification:
   a. Enable receive checksum offload engine by setting the EMAC_MACCFG.IPC bit.
   b. Enable 8 word descriptor (32 bytes), by setting the EMAC_DMA_BUSMODE.ATDS bit.
   c. Provide a total of 8 x 32-bit word space for the receive descriptor.
   d. Wait for the receive interrupt and check for extended status availability by parsing bit 0 in the RDES0 word.
   e. If extended status available, read RDES4 and pass to a higher layer such as the IP stack.

      The higher software layer can check for IPv4/IPv6/payload type and checksum payload or header errors.

Programming the System Time Module

Use the following procedure to configure the PTP module

1. Enable the PTP module by setting the EMAC_TM_CTL.TSENA bit 0.
2. System Time Initialization
   a. The time (seconds and nanoseconds) at which system time is initialized. Write the time into the EMAC_TM_SECUPDT and EMAC_TM_NSECUPDT registers.
   b. Set EMAC_TM_CTL.TSINIT bit. System time is initialized and this bit is cleared automatically.

c. Configure binary or digital rollover of the EMAC_TM_NSEC register using the EMAC_TM_CTL.TSCTRLSSR bit.

3. System Time Coarse Correction
   a. Write the offset time (seconds and nanoseconds) to add to or subtract from the system time using the EMAC_TM_SECUPDT and EMAC_TM_NSECUPDT registers.
   b. Choose between add or subtract offset time using the EMAC_TM_NSECUPDT.ADDSUB bit.
   c. Set the EMAC_TM_CTL.TSUPDT bit to correct system time with offset time. This bit is cleared automatically.
4. System Time Fine Correction
   a. Calculate the addend value based on the input PTP clock frequency and the frequency requirement. See Fine Correction Method.
   b. Write the calculated addend value in EMAC_TM_ADDEND register and set the EMAC_TM_CTL.TSADDREG bit to update the addend value. This bit is cleared automatically.
   c. Configure the EMAC_TM_SUBSEC register based on new PTP frequency.
5. Target Time Trigger (Alarm)
   a. Set the EMAC_IMSK.TS bit to enable PTP interrupts.
   b. Program the EMAC_TM_PPSCTL.TRGTMODSEL bit with 00 or 10 (for PPS start or stop time programming).
   c. Program the time when the interrupt must trigger using the EMAC_TM_TGTM and EMAC_TM_NTGTM registers. The programmed time must be greater than the current system time.

      ADDITIONAL INFORMATION: If the programmed time is not greater than the target time, a programming error occurs. The EMAC uses the EMAC_TM_STMPSTAT.TSTRGTERR bit to indicate the error.
   d. Set the EMAC_TM_CTL.TSTRIG bit to enable the target time trigger interrupt.

      After the system time reaches the programmed target time (in step 2), the target time trigger interrupt occurs. The EMAC_TM_STMPSTAT.TSTARGT and EMAC_ISTAT.TS bits indicate the error. The EMAC_TM_CTL.TSTRIG bit is cleared automatically.

Programming the PTP for Frame Detection and Timestamping

Use the following procedure to configure the PTP module.

1. For timestamping a transmitting frame, set the TTSE bit in the TDES0 register of the corresponding frame.
2. Extend the descriptor word length from 4 words to 8 words by setting the EMAC_DMA_BUSMODE.ATDS bit.
3. Configure bits 18–10 in the EMAC_TM_CTL register so that the PTP module detects and timestamps only specific types of received frames. Refer to the EMAC_TM_CTL register description for more information.

4. Enable the PTP module by setting the EMAC_TM_CTL.TSENA bit.
5. Initialize the system time.
6. Verify the RDES4 register for the status of the received frame and the RDES6 and RDES7 registers for timestamp nanoseconds and seconds value.

Programming for Auxiliary Timestamps

1. Set the EMAC_IMSK.TS bit to enable PTP interrupts.
2. Set the EMAC_TM_CTL.TSENA bit to enable the PTP module.
3. Initialize system time.

   ADDITIONAL INFORMATION: Whenever a rising edge on the auxiliary timestamp trigger pin is detected, system time seconds and nanoseconds are captured and stored into 4-deep auxiliary timestamp FIFO. An auxiliary timestamp trigger interrupt occurs. The EMAC uses the EMAC_TM_STMPSTAT.ATSTS and the EMAC_IMSK.TS bits to indicate the interrupt.
4. Read the contents of the FIFO one-by-one through EMAC_TM_AUXSTMP_SEC and EMAC_TM_AUXSTMP_NSEC registers. One level of the FIFO is cleared when the EMAC_TM_AUXSTMP_SEC register is read. Therefore, read the EMAC_TM_AUXSTMP_NSEC register before the EMAC_TM_AUXSTMP_SEC register.
5. Set the EMAC_TM_CTL.ATSFC bit to clear the FIFO.

Programming Fixed Pulse-Per-Second Output

Use the following procedure to program the fixed PPS output.

1. Enable the PTP module by setting the EMAC_TM_CTL.TSENA bit.
2. Output the PPS waveform by configuring the EMAC_TM_PPSCTL.PPSCTL bits and binary or digital rollover using the EMAC_TM_CTL.TSCTRLSSR bit. See Fixed Pulse-Per-Second Output.

Programming Flexible Pulse-Per-Second Output

Use the following procedure to program flexible PPS output.

1. Enable the PTP module by setting the EMAC_TM_CTL.TSENA bit.
2. Set the EMAC_TM_PPSCTL.PPSEN bit to enable flexible PPS output.
3. Program the EMAC_TM_PPSCTL.TRGTMODSEL bits with 11 or 10 (for target time trigger interrupt).
4. Program the start time value when the PPS output starts using the EMAC_TM_TGTM and EMAC_TM_NTGTM registers. Ensure that the EMAC_TM_NTGTM.TSTRBUSY bit is reset before programming the target time registers again.

5. Program the period of the PPS signal output using the EMAC_TM_PPSINTVL register for pulse train output. Program the width of the PPS signal output in the EMAC_TM_PPSWIDTH register for single pulse or pulse train output.
6. Ensure that the EMAC_TM_PPSCTL.PPSCTL bits are cleared. Then, program the bits to 0001 to start single pulse, or to 0010 to start pulse train at programmed start time (Step 4).

   ADDITIONAL INFORMATION: The PPS pulse train is free-running unless stopped by a STOP pulse train at time command (EMAC_TM_PPSCTL.PPSCTL = 0100) or STOP pulse train immediately command EMAC_TM_PPSCTL.PPSCTL = 0101).
7. The start of pulse generation can be canceled by giving the cancel start command (EMAC_TM_PPSCTL.PPSCTL = 0011) before the programmed start time (Step 4) elapses.
8. Program the stop time value when the PPS output must stop using the EMAC_TM_TGTM and EMAC_TM_NTGTM registers. Ensure that the EMAC_TM_NTGTM.TSTRBUSY bit is reset before programming the target time registers again.
9. Ensure that the EMAC_TM_PPSCTL.PPSCTL bits are cleared. Then, program the bits to 0100. This programming stops the train of pulses on PPS signal output after the programmed stop time (Step 8) elapses.

   ADDITIONAL INFORMATION: The pulse train can be stopped immediately by giving the STOP pulse train immediately command (EMAC_TM_PPSCTL.PPSCTL = 0101). Program the EMAC_TM_PPSCTL.PPSCTL bits to 0110 before the programmed stop time (Step 8) elapses to cancel the stop pulse train command (given in Step 9).

EMAC Programming Concepts

The following sections provide basic information and guidelines to help with programming the EMAC module.

IEEE 802.3 Ethernet Packet Structure

The ***IEEE 802.3 Frame Structure*** table provides typical frame format of an Ethernet packet. Refer to the IEEE standards for detailed information on Ethernet packets and their format.

Table 23-42: IEEE 802.3 Frame Structure

Parameter	Description	Position in Ethernet Packet	Total Bytes
PREAMBLE	This parameter is a 56-bit (7-byte) pattern of alternating 1 and 0 bits (#10101010), which allows devices on the network to detect a new incoming frame for synchronization.	1	7
SFD	The SFD (#10101011) is a 1-byte pattern designed to break the preamble pattern, and signal the start of the actual frame.	2	1
DA	48-bit destination address. This parameter can be a unicast, multicast, or broadcast address.	3	6

Table 23-42: IEEE 802.3 Frame Structure (Continued)

Parameter	Description	Position in Ethernet Packet	Total Bytes
SA	48-bit long source address, typically a unicast, multicast, or broadcast address.	4	6
LT	Typically this field is the length, in terms of the number of bytes, and can be anywhere between 0–1500. When the value is greater than or equal to 0x0600, this field also indicates the type of special payload carried by the frame. Examples include 0x8808 for flow control and 0x0800 for IPv4.	5	2
DATA	Actual application data payload, usually between 0–1500.	6	0–1500
PAD	This field compensates for data frames that are shorter than 64 bytes long, not including the preamble.	7	0–46
FCS	The frame check sequence is a 32-bit cyclic redundancy check that detects corrupted data within the entire frame. This parameter is generated from a CRC-32 polynomial code (CRC-32-IEEE): G (x) = x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1.	8	4

Frame Size Statistics for Application Software

Table 23-43: Ethernet Frame Size Statistics

Frame size statistics	VLAN-specific change Comments	
Information bytes/Header	4 byte 802.1Q header inserted after source address and before Type/LAN in 802.3 packets = 22 bytes.	6 x 2 + 2 + 4 = 18 bytes (DA+SA+LT+FCS)
Minimum Frame Size (typical)	If DATA is NULL, 42-byte padding makes 64 bytes (42 +22)	64 bytes. If DATA is NULL, 46-byte padding makes 64 bytes (46 +18)
Maximum Frame Size (typical)	1522 bytes	1518 bytes (1500 bytes DATA and 18-bytes header)
Jumbo Frame Size	9022 bytes	Typical industry standard. Ethernet jumbo frame size treated as 9018 bytes.

Software Visualization of Programmable Packet Size

The ***Visualization of Programmable Packet Size*** table provides the byte sizes of packets with various configurations.

Table 23-44: Visualization of Programmable Packet Size

Size in Bytes	Comments
16384	Receive watchdog and transmit jabber disabled, jumbo frames enabled.
10240	Receive watchdog and transmit jabber disabled, jumbo frames disabled.
2048	Receive watchdog and transmit jabber enabled.
1518	Typical max size of Ethernet frame. Receive watchdog and transmit jabber enabled.

Table 23-44: Visualization of Programmable Packet Size (Continued)

Size in Bytes	Comments
64	Typical minimum size of Ethernet frame.
< 64	Runt frames requiring Zero-PAD.

Ethernet Packet Structure in C

The following is an example for Ethernet packet structure in the C language.

```
typedef struct ETHER_PACKET
{
    char  dst_addr[6];          //destination address
    char  src_addr[6];          //source address
    char  length[2];            //length of actual data
    char  data[DATA_SIZE];      //application data
    char  fdlimit[DELIMIT_SIZE];//32-bit delimit (if manual appending)
    char  fcs[4];               //crc frame checksum, used by RX buffer.
} ETHER_PACKET;
```

DMA Descriptor Implementation in C

The following code is a simple implementation of descriptors in ring and chain model in C language. Typically 4 WORDs (32-bit) are used for descriptors. Using checksum offload or the PTP engine requires 8 WORDs. Only high-level common functions across transmit and receive descriptors are considered here.

```
/* DMA Ring Descriptor */
typedef struct EMAC_DMADESC_RING
{
    unsigned int                    Status;     //TDES0 OR RDES0
    unsigned int                    ControlDesc; //TDES1 OR RDES1
    unsigned int                    StartAddr1;  //TDES2 OR RDES2
    unsigned int                    StartAddr2;  //TDES3 OR RDES3
    #ifdef CHECKSUM_OFFLOAD
    struct EMAC_EXT_STAT      ExtendedStat;
    #endif
} EMAC_DMADESC_RING;
/* DMA Chain Descriptor */
typedef struct EMAC_DMADESC_CHAIN
{
    unsigned int                    Status;     //TDES0 OR RDES0
    unsigned int                    ControlDesc; //TDES1 OR RDES1
    unsigned int                    StartAddr;     //TDES2 OR RDES2
    struct EMAC_DMADESC_CHAIN     *pNextDesc;  //TDES3 OR RDES3
    #ifdef CHECKSUM_OFFLOAD
    struct EMAC_EXT_STAT     ExtendedStat;
    #endif
} EMAC_DMADESC_CHAIN;
/* Extended Status Descriptor with PTP not enabled*/
```

```
typedef struct EMAC_EXT_STAT
{
    #ifdef RX_DESC
    unsigned int                    CheckSumStat;//RDES4
    #ifdef TX_DESC
    unsigned int                    Reserved;     //TDES4
    #endif
    unsigned int                    Reserved;    //RDES5 OR TDES5
    unsigned int                    Reserved;      //RDES6 OR TDES6
    unsigned int                    Reserved;     //RDES7 OR TDES7
} EMAC_EXT_STAT;
```

PTP Header Structure in C

The following code is an example of the PTM message format.

```
/* PTP Message Format (Refer to PTP Frame Over IPv4)*/
    typedef struct EMAC_PTP_HEADER
    {
    unsigned char    messageType:4,    //PTP Version 2 message type
    transportSpecific:4;
    unsigned char    versionPTP;       //PTP Version (1 or 2)
    unsigned short    messageLength;
    unsigned char    domainNumber;
    unsigned char    RESERVED1;
    unsigned short    flagField;
    unsigned char    correctionField[8];
    unsigned char    RESERVED2[4];
    unsigned char    sourcePortIdentity[10];
    unsigned short    sequenceid;
    unsigned char    controlField;//PTP Version 1 message type
    unsigned char    logMessageInterval;
    }EMAC_PTP_HEADER;
```

ADSP-CM40x EMAC Register Descriptions

Ethernet MAC (EMAC) contains the following registers.

Table 23-45: ADSP-CM40x EMAC Register List

Name	Description
EMAC_ADDR0_HI	MAC Address 0 High Register
EMAC_ADDR0_LO	MAC Address 0 Low Register
EMAC_DBG	Debug Register
EMAC_DMA_BMMODE	DMA SCB Bus Mode Register
EMAC_DMA_BMSTAT	DMA SCB Status Register

Table 23-45: ADSP-CM40x EMAC Register List (Continued)

Name	Description
EMAC_DMA_BUSMODE	DMA Bus Mode Register
EMAC_DMA_IEN	DMA Interrupt Enable Register
EMAC_DMA_MISS_FRM	DMA Missed Frame Register
EMAC_DMA_OPMODE	DMA Operation Mode Register
EMAC_DMA_RXBUF_CUR	DMA Rx Buffer Current Register
EMAC_DMA_RXDSC_ADDR	DMA Rx Descriptor List Address Register
EMAC_DMA_RXDSC_CUR	DMA Rx Descriptor Current Register
EMAC_DMA_RXIWDOG	DMA Rx Interrupt Watch Dog Register
EMAC_DMA_RXPOLL	DMA Rx Poll Demand register
EMAC_DMA_STAT	DMA Status Register
EMAC_DMA_TXBUF_CUR	DMA Tx Buffer Current Register
EMAC_DMA_TXDSC_ADDR	DMA Tx Descriptor List Address Register
EMAC_DMA_TXDSC_CUR	DMA Tx Descriptor Current Register
EMAC_DMA_TXPOLL	DMA Tx Poll Demand Register
EMAC_FLOWCTL	FLow Control Register
EMAC_HASHTBL_HI	Hash Table High Register
EMAC_HASHTBL_LO	Hash Table Low Register
EMAC_IMSK	Interrupt Mask Register
EMAC_IPC_RXIMSK	MMC IPC Rx Interrupt Mask Register
EMAC_IPC_RXINT	MMC IPC Rx Interrupt Register
EMAC_ISTAT	Interrupt Status Register
EMAC_MACCFG	MAC Configuration Register
EMAC_MACFRMFILT	MAC Rx Frame Filter Register
EMAC_MMC_CTL	MMC Control Register
EMAC_MMC_RXIMSK	MMC Rx Interrupt Mask Register
EMAC_MMC_RXINT	MMC Rx Interrupt Register
EMAC_MMC_TXIMSK	MMC TX Interrupt Mask Register
EMAC_MMC_TXINT	MMC Tx Interrupt Register
EMAC_RX1024TOMAX_GB	Rx 1024- to Max-Byte Frames (Good/Bad) Register
EMAC_RX128TO255_GB	Rx 128- to 255-Byte Frames (Good/Bad) Register
EMAC_RX256TO511_GB	Rx 256- to 511-Byte Frames (Good/Bad) Register

Table 23-45: ADSP-CM40x EMAC Register List (Continued)

Name	Description
EMAC_RX512TO1023_GB	Rx 512- to 1023-Byte Frames (Good/Bad) Register
EMAC_RX64_GB	Rx 64-Byte Frames (Good/Bad) Register
EMAC_RX65TO127_GB	Rx 65- to 127-Byte Frames (Good/Bad) Register
EMAC_RXALIGN_ERR	Rx alignment Error Register
EMAC_RXBCASTFRM_G	Rx Broadcast Frames (Good) Register
EMAC_RXCRC_ERR	Rx CRC Error Register
EMAC_RXFIFO_OVF	Rx FIFO Overflow Register
EMAC_RXFRMCNT_GB	Rx Frame Count (Good/Bad) Register
EMAC_RXICMP_ERR_FRM	Rx ICMP Error Frames Register
EMAC_RXICMP_ERR_OCT	Rx ICMP Error Octets Register
EMAC_RXICMP_GD_FRM	Rx ICMP Good Frames Register
EMAC_RXICMP_GD_OCT	Rx ICMP Good Octets Register
EMAC_RXIPV4_FRAG_FRM	Rx IPv4 Datagrams Fragmented Frames Register
EMAC_RXIPV4_FRAG_OCT	Rx IPv4 Datagrams Fragmented Octets Register
EMAC_RXIPV4_GD_FRM	Rx IPv4 Datagrams (Good) Register
EMAC_RXIPV4_GD_OCT	Rx IPv4 Datagrams Good Octets Register
EMAC_RXIPV4_HDR_ERR_FRM	Rx IPv4 Datagrams Header Errors Register
EMAC_RXIPV4_HDR_ERR_OCT	Rx IPv4 Datagrams Header Errors Register
EMAC_RXIPV4_NOPAY_FRM	Rx IPv4 Datagrams No Payload Frame Register
EMAC_RXIPV4_NOPAY_OCT	Rx IPv4 Datagrams No Payload Octets Register
EMAC_RXIPV4_UDSBL_FRM	Rx IPv4 UDP Disabled Frames Register
EMAC_RXIPV4_UDSBL_OCT	Rx IPv4 UDP Disabled Octets Register
EMAC_RXIPV6_GD_FRM	Rx IPv6 Datagrams Good Frames Register
EMAC_RXIPV6_GD_OCT	Rx IPv6 Good Octets Register
EMAC_RXIPV6_HDR_ERR_FRM	Rx IPv6 Datagrams Header Error Frames Register
EMAC_RXIPV6_HDR_ERR_OCT	Rx IPv6 Header Errors Register
EMAC_RXIPV6_NOPAY_FRM	Rx IPv6 Datagrams No Payload Frames Register
EMAC_RXIPV6_NOPAY_OCT	Rx IPv6 No Payload Octets Register
EMAC_RXJAB_ERR	Rx Jab Error Register
EMAC_RXLEN_ERR	Rx Length Error Register
EMAC_RXMCASTFRM_G	Rx Multicast Frames (Good) Register

Table 23-45: ADSP-CM40x EMAC Register List (Continued)

Name	Description
EMAC_RXOCTCNT_G	Rx Octet Count (Good) Register
EMAC_RXOCTCNT_GB	Rx Octet Count (Good/Bad) Register
EMAC_RXOORTYPE	Rx Out Of Range Type Register
EMAC_RXOSIZE_G	Rx Oversize (Good) Register
EMAC_RXPAUSEFRM	Rx Pause Frames Register
EMAC_RXRUNT_ERR	Rx Runt Error Register
EMAC_RXTCP_ERR_FRM	Rx TCP Error Frames Register
EMAC_RXTCP_ERR_OCT	Rx TCP Error Octets Register
EMAC_RXTCP_GD_FRM	Rx TCP Good Frames Register
EMAC_RXTCP_GD_OCT	Rx TCP Good Octets Register
EMAC_RXUCASTFRM_G	Rx Unicast Frames (Good) Register
EMAC_RXUDP_ERR_FRM	Rx UDP Error Frames Register
EMAC_RXUDP_ERR_OCT	Rx UDP Error Octets Register
EMAC_RXUDP_GD_FRM	Rx UDP Good Frames Register
EMAC_RXUDP_GD_OCT	Rx UDP Good Octets Register
EMAC_RXUSIZE_G	Rx Undersize (Good) Register
EMAC_RXVLANFRM_GB	Rx VLAN Frames (Good/Bad) Register
EMAC_RXWDOG_ERR	Rx Watch Dog Error Register
EMAC_SMI_ADDR	SMI Address Register
EMAC_SMI_DATA	SMI Data Register
EMAC_TM_ADDEND	Time Stamp Addend Register
EMAC_TM_AUXSTMP_NSEC	Time Stamp Auxiliary TS Nano Seconds Register
EMAC_TM_AUXSTMP_SEC	Time Stamp Auxiliary TM Seconds Register
EMAC_TM_CTL	Time Stamp Control Register
EMAC_TM_HISEC	Time Stamp High Second Register
EMAC_TM_NSEC	Time Stamp Nanoseconds Register
EMAC_TM_NSECUPDT	Time Stamp Nanoseconds Update Register
EMAC_TM_NTGTM	Time Stamp Target Time Nanoseconds Register
EMAC_TM_PPSCTL	PPS Control Register
EMAC_TM_PPSINTVL	Time Stamp PPS Interval Register
EMAC_TM_PPSWIDTH	PPS Width Register

Table 23-45: ADSP-CM40x EMAC Register List (Continued)

Name	Description
EMAC_TM_SEC	Time Stamp Low Seconds Register
EMAC_TM_SECUPDT	Time Stamp Seconds Update Register
EMAC_TM_STMPSTAT	Time Stamp Status Register
EMAC_TM_SUBSEC	Time Stamp Sub Second Increment Register
EMAC_TM_TGTM	Time Stamp Target Time Seconds Register
EMAC_TX1024TOMAX_GB	Tx 1024- to Max-Byte Frames (Good/Bad) Register
EMAC_TX128TO255_GB	Tx 128- to 255-Byte Frames (Good/Bad) Register
EMAC_TX256TO511_GB	Tx 256- to 511-Byte Frames (Good/Bad) Register
EMAC_TX512TO1023_GB	Tx 512- to 1023-Byte Frames (Good/Bad) Register
EMAC_TX64_GB	Tx 64-Byte Frames (Good/Bad) Register
EMAC_TX65TO127_GB	Tx 65- to 127-Byte Frames (Good/Bad) Register
EMAC_TXBCASTFRM_G	Tx Broadcast Frames (Good) Register
EMAC_TXBCASTFRM_GB	Tx Broadcast Frames (Good/Bad) Register
EMAC_TXCARR_ERR	Tx Carrier Error Register
EMAC_TXDEFERRED	Tx Deferred Register
EMAC_TXEXCESSCOL	Tx Excess Collision Register
EMAC_TXEXCESSDEF	Tx Excess Deferral Register
EMAC_TXFRMCNT_G	Tx Frame Count (Good) Register
EMAC_TXFRMCNT_GB	Tx Frame Count (Good/Bad) Register
EMAC_TXLATECOL	Tx Late Collision Register
EMAC_TXMCASTFRM_G	Tx Multicast Frames (Good) Register
EMAC_TXMCASTFRM_GB	Tx Multicast Frames (Good/Bad) Register
EMAC_TXMULTCOL_G	Tx Multiple Collision (Good) Register
EMAC_TXOCTCNT_G	Tx Octet Count (Good) Register
EMAC_TXOCTCNT_GB	Tx OCT Count (Good/Bad) Register
EMAC_TXPAUSEFRM	Tx Pause Frame Register
EMAC_TXSNGCOL_G	Tx Single Collision (Good) Register
EMAC_TXUCASTFRM_GB	Tx Unicast Frames (Good/Bad) Register
EMAC_TXUNDR_ERR	Tx Underflow Error Register
EMAC_TXVLANFRM_G	Tx VLAN Frames (Good) Register
EMAC_VLANTAG	VLAN Tag Register

MAC Address 0 High Register

The EMAC_ADDR0_HI register holds the address 0 high bits.

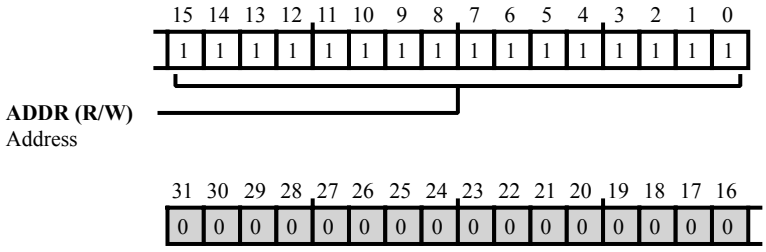

Figure 23-16: EMAC_ADDR0_HI Register Diagram

Table 23-46: EMAC_ADDR0_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	ADDR	Address. The EMAC_ADDR0_HI.ADDR bits contain the upper 16 bits (47:32) of the 6-byte first MAC address. This address is used by the MAC for filtering for received frames and for inserting the MAC address in the Transmit Flow Control (PAUSE) Frames.

MAC Address 0 Low Register

The EMAC_ADDR0_LO register holds the address 0 low bits.

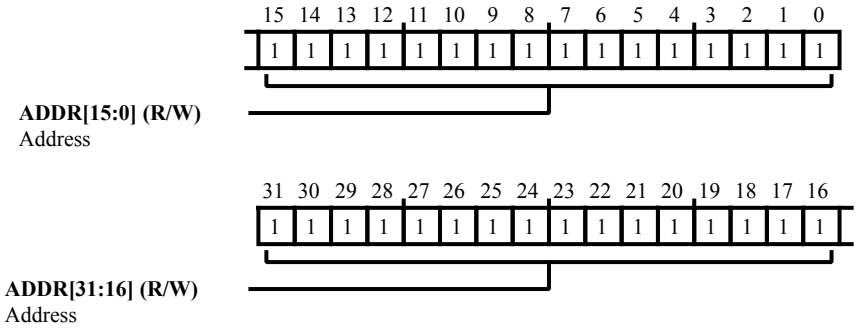

Figure 23-17: EMAC_ADDR0_LO Register Diagram

Table 23-47: EMAC_ADDR0_LO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDR	Address. The EMAC_ADDR0_LO.ADDR bits contain the lower 32 bits of the 6-byte first MAC address. This address is used by the MAC for filtering for received frames and for inserting the MAC address in the Transmit Flow Control (PAUSE) Frames.

Debug Register

The EMAC_DBG register contains EMAC debug status information.

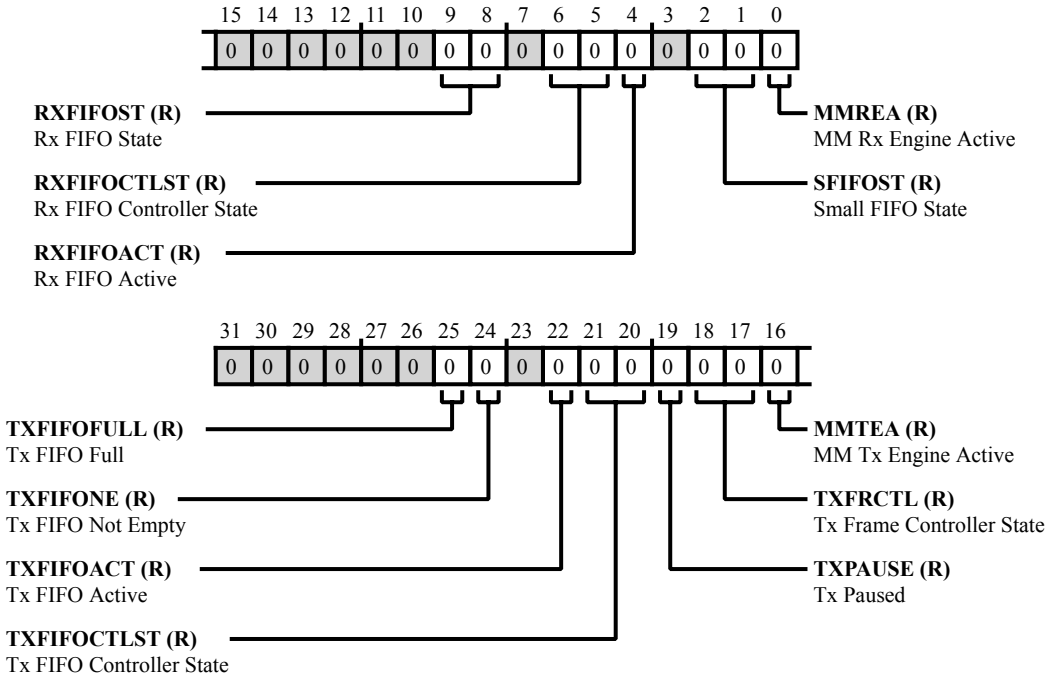

Figure 23-18: EMAC_DBG Register Diagram

Table 23-48: EMAC_DBG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/NW)	TXFIFOFULL	Tx FIFO Full. The EMAC_DBG.TXFIFOFULL bit, when high, indicates that the MFL TxStatus FIFO is full, and the MFL cannot accept any more frames for transmission.
24 (R/NW)	TXFIFONE	Tx FIFO Not Empty. The EMAC_DBG.TXFIFONE bit, when high, indicates that the MFL TxFIFO is not empty and has some data left for transmission.
22 (R/NW)	TXFIFOACT	Tx FIFO Active. The EMAC_DBG.TXFIFOACT bit, when high, indicates that the MFL TxFIFO write controller is active and transferring data to the TxFIFO.
21:20 (R/NW)	TXFIFOCTLST	Tx FIFO Controller State. The EMAC_DBG.TXFIFOCTLST bits indicate the state of the TxFIFO read controller as: 00=IDLE state, 01=READ state (transferring data to MAC transmitter), 10=Waiting for TxStatus from MAC transmitter, and 11=Writing the received TxStatus or flushing the TxFIFO

Table 23-48: EMAC_DBG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/NW)	TXPAUSE	Tx Paused. The EMAC_DBG.TXPAUSE bit, when high, indicates that the MAC transmitter is in PAUSE condition (in full-duplex only) and does not schedule any frame for transmission.	
18:17 (R/NW)	TXFRCTL	Tx Frame Controller State. The EMAC_DBG.TXFRCTL bits indicate the state of the MAC transmit frame controller module.	
		0	Idle Frame controller is in idle state.
		1	Wait Frame controller is waiting for status of previous frame or IFG/backoff period end.
		2	Pause Frame controller is generating and transmitting a PAUSE control frame (in full duplex mode).
		3	Transmit Frame controller is transferring input frame for transmission.
16 (R/NW)	MMTEA	MM Tx Engine Active. The EMAC_DBG.MMTEA bit, when high, indicates that the MAC core transmit protocol engine is actively transmitting data and is not in IDLE state.	
9:8 (R/NW)	RXFIFOST	Rx FIFO State. The EMAC_DBG.RXFIFOST bits give the status of the RxFIFO fill level and indicate the relationship to the flow-control activation threshold.	
		0	Rx FIFO Empty
		1	Rx FIFO Below De-activate FCT
		2	Rx FIFO Above De-activate FCT
		3	Rx FIFO Full
6:5 (R/NW)	RXFIFOCTLST	Rx FIFO Controller State. The EMAC_DBG.RXFIFOCTLST bits give the state of the RxFIFO read controller.	
		0	Idle Read controller is in idle state.
		1	Read Data Read controller is reading frame data.
		2	Read Status Read controller is reading frame status or time-stamp.
		3	Flush Read controller is flushing the frame data and status.
4 (R/NW)	RXFIFOACT	Rx FIFO Active. The EMAC_DBG.RXFIFOACT bit, when high, indicates that the MFL RxFIFO write controller is active and is transferring a received frame to the FIFO.	

Table 23-48: EMAC_DBG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2:1 (R/NW)	SFIFOST	Small FIFO State. The `EMAC_DBG.SFIFOST` bit, when high, indicates the active state of the small FIFO read and write controllers respectively of the MAC receive frame controller module.
0 (R/NW)	MMREA	MM Rx Engine Active. The `EMAC_DBG.MMREA` bit, when high, indicates that the MAC core receive protocol engine is actively receiving data and is not in IDLE state.

DMA SCB Bus Mode Register

The EMAC_DMA_BMMODE register selects EMAC DMA system cross bar bus mode features.

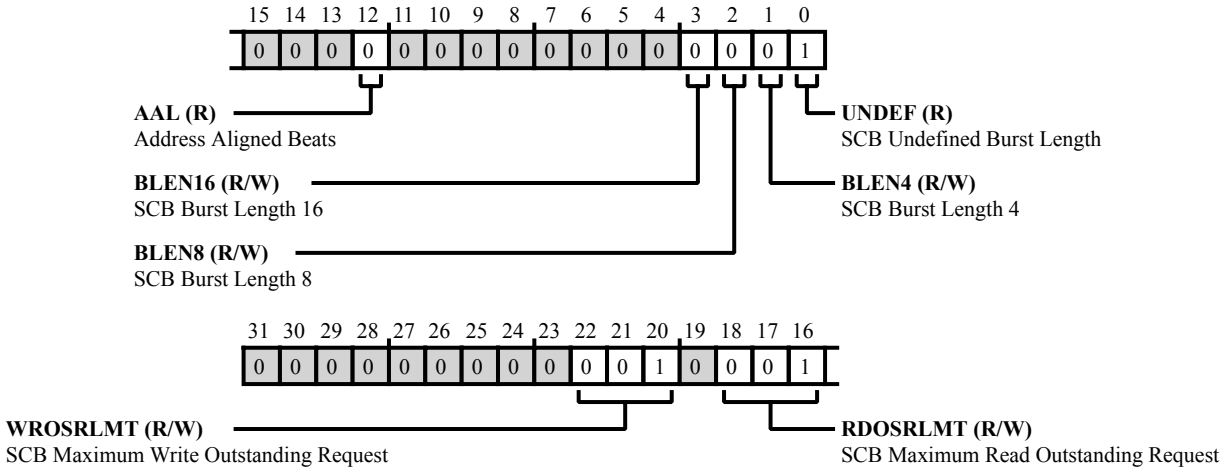

Figure 23-19: EMAC_DMA_BMMODE Register Diagram

Table 23-49: EMAC_DMA_BMMODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
22:20 (R/W)	WROSRLMT	SCB Maximum Write Outstanding Request. The EMAC_DMA_BMMODE.WROSRLMT bit field's value limits the maximum outstanding request on the SCB write interface. Maximum outstanding requests = WR_OSR_LMT+1. EMAC-SCB supports up to 4 outstanding write requests.
18:16 (R/W)	RDOSRLMT	SCB Maximum Read Outstanding Request. The EMAC_DMA_BMMODE.RDOSRLMT bit field's value limits the maximum outstanding request on the SCB read interface. Maximum outstanding requests = RD_OSR_LMT+1. EMAC-SCB supports up to 4 outstanding read requests.
12 (R/NW)	AAL	Address Aligned Beats. The EMAC_DMA_BMMODE.AAL bit (read-only) reflects the state of the EMAC_DMA_BUSMODE.AAL bit. When this bit is set to 1, EMAC-SCB performs address-aligned burst transfers on both read and write channels.
3 (R/W)	BLEN16	SCB Burst Length 16. The EMAC_DMA_BMMODE.BLEN16 bit, when set (or when EMAC_DMA_BMMODE.UNDEF is set to 1), directs the EMAC-SCB to select a burst length of 16 on the SCB master interface.
2 (R/W)	BLEN8	SCB Burst Length 8. The EMAC_DMA_BMMODE.BLEN8 bit, when set (or when EMAC_DMA_BMMODE.UNDEF is set to 1), directs the EMAC-SCB to select a burst length of 8 on the SCB master interface.

Table 23-49: EMAC_DMA_BMMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	BLEN4	SCB Burst Length 4. The EMAC_DMA_BMMODE.BLEN4 bit, when set (or when EMAC_DMA_BMMODE.UNDEF is set to 1), directs the EMAC-SCB to select a burst length of 4 on the SCB master interface.
0 (R/NW)	UNDEF	SCB Undefined Burst Length. The EMAC_DMA_BMMODE.UNDEF bit (read-only) indicates the complement (invert) value of EMAC_DMA_BUSMODE.FB bit . When this bit is set to 1, the EMAC-SCB is allowed to perform any burst length equal to or below the maximum allowed burst length as programmed in bits[3:1]. When this bit is set to 0, the EMAC-SCB is allowed to perform only fixed burst lengths as indicated by 16/8/4, or a burst length of 1.

DMA SCB Status Register

The EMAC_DMA_BMSTAT register indicates EMAC DMA system cross bar status.

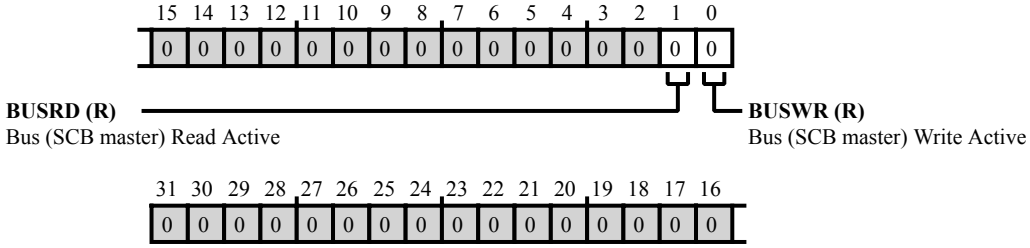

Figure 23-20: EMAC_DMA_BMSTAT Register Diagram

Table 23-50: EMAC_DMA_BMSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/NW)	BUSRD	Bus (SCB master) Read Active. The EMAC_DMA_BMSTAT.BUSRD bit, when high, indicates that SCB Master's read channel is active and transferring data.
0 (R/NW)	BUSWR	Bus (SCB master) Write Active. The EMAC_DMA_BMSTAT.BUSWR bit, when high, indicates that SCB Master's write channel is active and transferring data.

DMA Bus Mode Register

The EMAC_DMA_BUSMODE register selects the DMA bus operating modes for EMAC DMA.

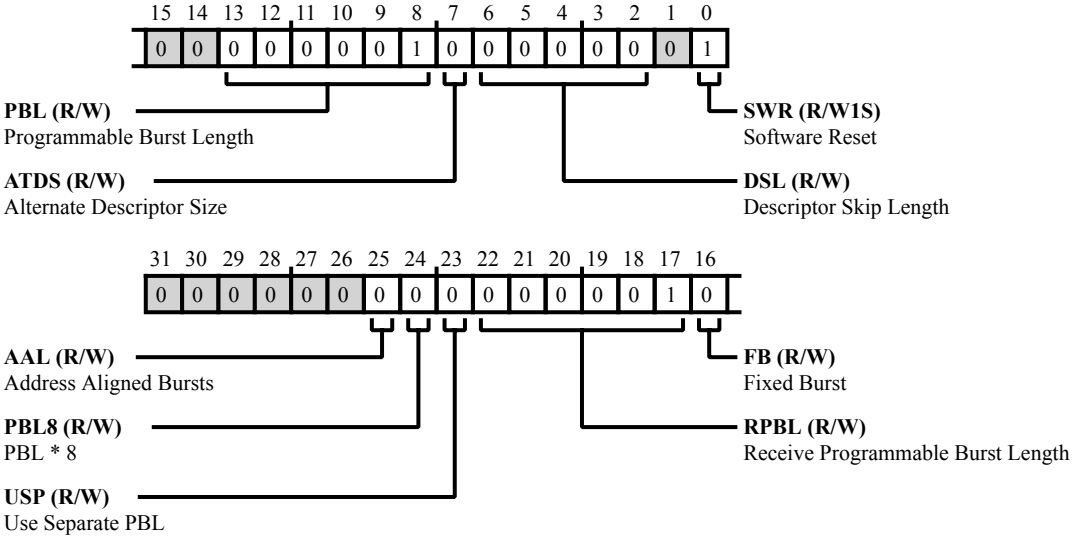

Figure 23-21: EMAC_DMA_BUSMODE Register Diagram

Table 23-51: EMAC_DMA_BUSMODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/W)	AAL	Address Aligned Bursts. The EMAC_DMA_BUSMODE.AAL bit, when set high and the FB bit equals 1, directs the SCB interface to generate all bursts aligned to the start address LS bits. If the FB bit is equal to 0, the first burst (accessing the data buffers start address) is not aligned, but subsequent bursts are aligned to the address.
24 (R/W)	PBL8	PBL * 8. The EMAC_DMA_BUSMODE.PBL8 bit, when set high, multiplies the PBL value programmed (bits [22:17] and bits [13:8]) eight times. Therefore, the DMA transfers the data in 8, 16, and 32 beats depending on the PBL value.
23 (R/W)	USP	Use Separate PBL. The EMAC_DMA_BUSMODE.USP bit, when set high, configures the Rx DMA to use the value configured in bits [22:17] as PBL while the PBL value in bits [13:8] is applicable to Tx DMA operations only.
22:17 (R/W)	RPBL	Receive Programmable Burst Length. The EMAC_DMA_BUSMODE.RPBL bits indicate the maximum number of beats to be transferred in one Rx DMA transaction. This is the maximum value that is used in a single block Read/Write. The Rx DMA always attempts to burst as specified in RPBL every time it starts a Burst transfer on the host bus. RPBL can be programmed with

Table 23-51: EMAC_DMA_BUSMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		permissible values of 1, 2, 4, 8, 16, and 32. Any other value results in undefined behavior. These bits are valid and applicable only when USP is set high.
16 (R/W)	FB	Fixed Burst. The EMAC_DMA_BUSMODE.FB bit controls whether the SCB Master interface performs fixed burst transfers or not. See the EMAC_DMA_BMMODE.UNDEF bit description for more information.
13:8 (R/W)	PBL	Programmable Burst Length. The EMAC_DMA_BUSMODE.PBL bits indicate the maximum number of beats to be transferred in one DMA transaction. This is the maximum value that is used in a single block Read/Write. The DMA always attempts to burst as specified in PBL each time it starts a Burst transfer on the host bus. Any other value results in undefined behavior. When USP is set high, this PBL value is applicable for Tx DMA transactions only. PBL-max limit = (FIFO size / 2) / 4. PBL-max limit (transmit) = 256 bytes / 2 /4 = 32. PBL-max limit (receive) = 128 bytes / 2 /4 = 16. Note that this PBL is at the DMA end. If PBL= 32 and if BLEN16 is enabled, the DMA automatically splits 32 bursts in to 2 x 16 bursts. If EMAC_DMA_BUSMODE.PBL =8, and if EMAC_DMA_BMMODE.BLEN16 is enabled, the max burst is limited to EMAC_DMA_BMMODE.BLEN8. If EMAC_DMA_BUSMODE.PBL8 bit is set, the programmed PBL value is multiplied by 8 times internally. However, the result cannot be more than the above maximum limits specified above.
7 (R/W)	ATDS	Alternate Descriptor Size. The EMAC_DMA_BUSMODE.ATDS bit, when set, increases the size of the alternate descriptor to 32 bytes (8 DWORDS). This is required when the Advanced Time Stamp feature or Full IPC Offload Engine is enabled in the receiver. When reset, the descriptor size reverts back to 4 DWORDs (16 bytes). The enhanced descriptor is not required if the Advanced Time Stamp and IPC Full Checksum Offload features are not enabled. In such case, you can use the 16 bytes descriptor to save 4 bytes of memory.
6:2 (R/W)	DSL	Descriptor Skip Length. The EMAC_DMA_BUSMODE.DSL bit specifies the number of 32-bit words to skip between two unchained descriptors. The address skipping starts from the end of current descriptor to the start of next descriptor. When DSL value is equal to zero, then the descriptor table is taken as contiguous by the DMA, in Ring mode.
0 (R/W1S)	SWR	Software Reset. The EMAC_DMA_BUSMODE.SWR bit, when set, directs the MAC DMA Controller to reset all MAC Subsystem internal registers and logic. It is cleared automatically after the reset operation has completed in all of the core clock domains. Read a 0 value in

Table 23-51: EMAC_DMA_BUSMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		this bit before re-programming any register of the core. Note: The reset operation is completed only when all the resets in all the active clock domains are de-asserted. Therefore, it is essential that all the PHY inputs clocks (applicable for the selected PHY interface) are present for software reset completion.

DMA Interrupt Enable Register

The EMAC_DMA_IEN register enables (unmasks) EMAC DMA interrupts.

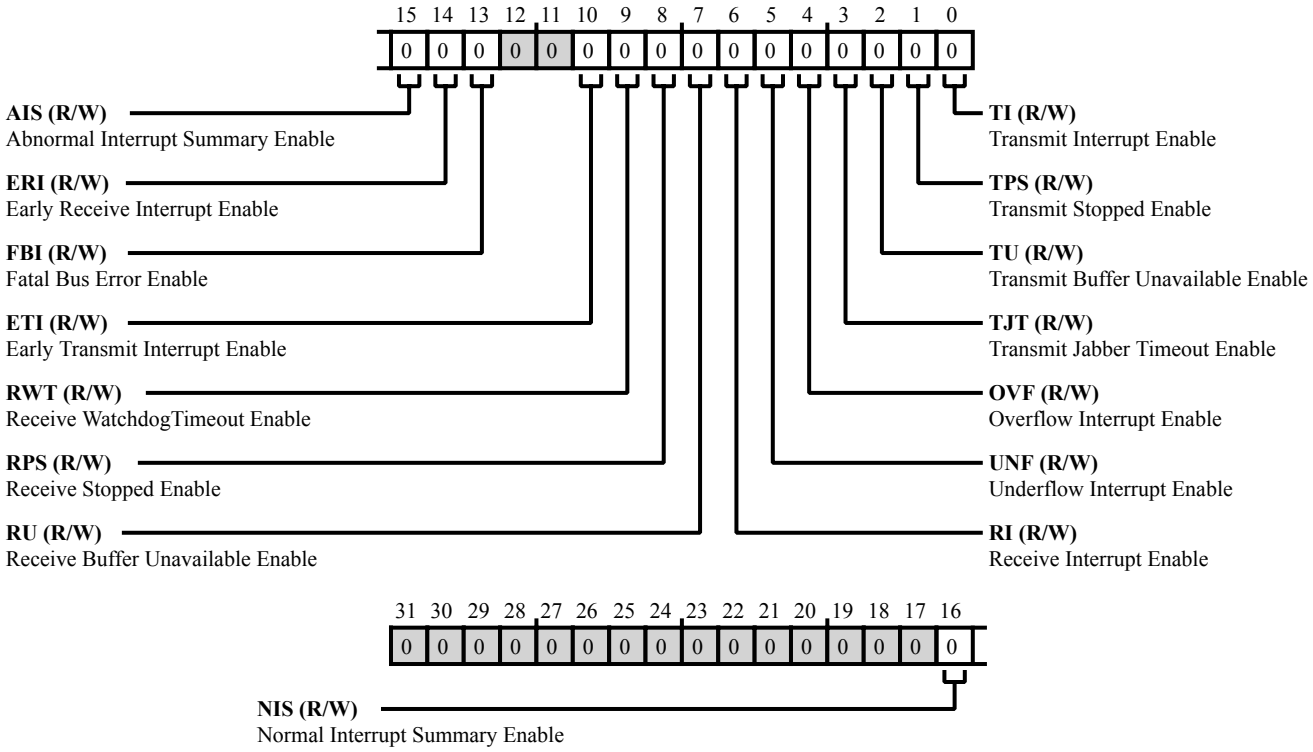

Figure 23-22: EMAC_DMA_IEN Register Diagram

Table 23-52: EMAC_DMA_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	NIS	Normal Interrupt Summary Enable. The EMAC_DMA_IEN.NIS bit, when set, enables a normal interrupt. When this bit is reset, a normal interrupt is disabled. This bit enables the following bits: EMAC_DMA_STAT.TI, EMAC_DMA_STAT.TU, EMAC_DMA_STAT.RI, and EMAC_DMA_STAT.ERI.
15 (R/W)	AIS	Abnormal Interrupt Summary Enable. The EMAC_DMA_IEN.AIS bit, when set, enables an abnormal interrupt. When this bit is reset, an Abnormal Interrupt is disabled. This bit enables the following bits: EMAC_DMA_STAT.TPS, EMAC_DMA_STAT.TJT, EMAC_DMA_STAT.OVF, EMAC_DMA_STAT.RU, EMAC_DMA_STAT.RPS, EMAC_DMA_STAT.RWT, EMAC_DMA_STAT.ETI, and EMAC_DMA_STAT.FBI.
14 (R/W)	ERI	Early Receive Interrupt Enable.

Table 23-52: EMAC_DMA_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		The EMAC_DMA_IEN.ERI bit, when set (and with EMAC_DMA_IEN.NIS =1), enables the Early Receive Interrupt. When this bit is reset, Early Receive Interrupt is disabled.
13 (R/W)	FBI	Fatal Bus Error Enable. The EMAC_DMA_IEN.FBI bit, when set (and with EMAC_DMA_IEN.AIS =1), enables the Fatal Bus Error Interrupt. When this bit is reset, Fatal Bus Error Enable Interrupt is disabled.
10 (R/W)	ETI	Early Transmit Interrupt Enable. The EMAC_DMA_IEN.ETI bit, when this bit is set (and with EMAC_DMA_IEN.AIS =1), enables the Early Transmit Interrupt. When this bit is reset, Early Transmit Interrupt is disabled.
9 (R/W)	RWT	Receive WatchdogTimeout Enable. The EMAC_DMA_IEN.RWT bit, when set (and with EMAC_DMA_IEN.AIS =1), enables the Receive Watchdog Timeout Interrupt. When this bit is reset, Receive Watchdog Timeout Interrupt is disabled.
8 (R/W)	RPS	Receive Stopped Enable. The EMAC_DMA_IEN.RPS bit, when set (and with EMAC_DMA_IEN.AIS =1), enables the Receive Stopped Interrupt is enabled. When this bit is reset, Receive Stopped Interrupt is disabled.
7 (R/W)	RU	Receive Buffer Unavailable Enable. The EMAC_DMA_IEN.RU bit, when set (and with EMAC_DMA_IEN.AIS =1), enables the Receive Buffer Unavailable Interrupt. When this bit is reset, the Receive Buffer Unavailable Interrupt is disabled.
6 (R/W)	RI	Receive Interrupt Enable. The EMAC_DMA_IEN.RI bit, when set (and with EMAC_DMA_IEN.NIS =1), enables the Receive Interrupt. When this bit is reset, Receive Interrupt is disabled.
5 (R/W)	UNF	Underflow Interrupt Enable. The EMAC_DMA_IEN.UNF bit, when set (and with EMAC_DMA_IEN.AIS =1), enables the Transmit Underflow Interrupt. When this bit is reset, Underflow Interrupt is disabled.
4 (R/W)	OVF	Overflow Interrupt Enable. The EMAC_DMA_IEN.OVF bit, when set (and with EMAC_DMA_IEN.AIS =1), enables the Receive Overflow Interrupt. When this bit is reset, Overflow Interrupt is disabled.
3 (R/W)	TJT	Transmit Jabber Timeout Enable. The EMAC_DMA_IEN.TJT bit, when set (and with EMAC_DMA_IEN.AIS =1), enables the Transmit Jabber Timeout Interrupt. When this bit is reset, Transmit Jabber Timeout Interrupt is disabled.
2	TU	Transmit Buffer Unavailable Enable.

Table 23-52: EMAC_DMA_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(R/W)		The EMAC_DMA_IEN.TU bit, when set (and with EMAC_DMA_IEN.NIS =1), enables the Transmit Buffer Unavailable Interrupt. When this bit is reset, Transmit Buffer Unavailable Interrupt is disabled.
1 (R/W)	TPS	Transmit Stopped Enable. The EMAC_DMA_IEN.TPS bit, when set (and with EMAC_DMA_IEN.AIS =1), enables the Transmission Stopped Interrupt. When this bit is reset, Transmission Stopped Interrupt is disabled.
0 (R/W)	TI	Transmit Interrupt Enable. The EMAC_DMA_IEN.TI bit, when set (and with EMAC_DMA_IEN.NIS =1), enables the Transmit Interrupt. When this bit is reset, Transmit Interrupt is disabled.

DMA Missed Frame Register

The EMAC_DMA_MISS_FRM register contains counters for EMAC DMA missed frames and buffer overflows.

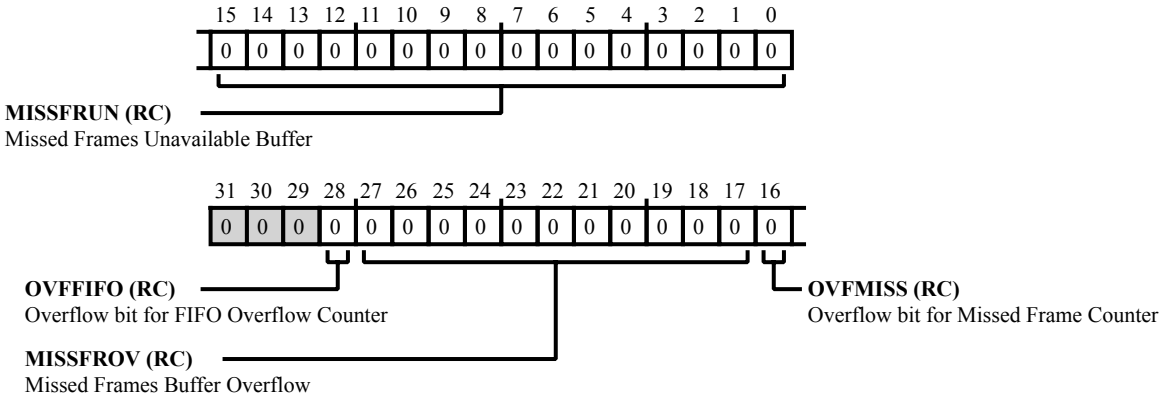

Figure 23-23: EMAC_DMA_MISS_FRM Register Diagram

Table 23-53: EMAC_DMA_MISS_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
28 (RC/NW)	OVFFIFO	Overflow bit for FIFO Overflow Counter. The EMAC_DMA_MISS_FRM.OVFFIFO bit holds the overflow bit for FIFO Overflow Counter.
27:17 (RC/NW)	MISSFROV	Missed Frames Buffer Overflow. The EMAC_DMA_MISS_FRM.MISSFROV bits indicate the number of frames missed by the application due to buffer overflow.
16 (RC/NW)	OVFMISS	Overflow bit for Missed Frame Counter. The EMAC_DMA_MISS_FRM.OVFMISS bit holds the overflow bit for the Missed Frame Counter.
15:0 (RC/NW)	MISSFRUN	Missed Frames Unavailable Buffer. The EMAC_DMA_MISS_FRM.MISSFRUN bits indicate the number of frames missed by the controller because of the Application Receive Buffer being unavailable.

DMA Operation Mode Register

The EMAC_DMA_OPMODE register selects receive and transmit DMA operating modes.

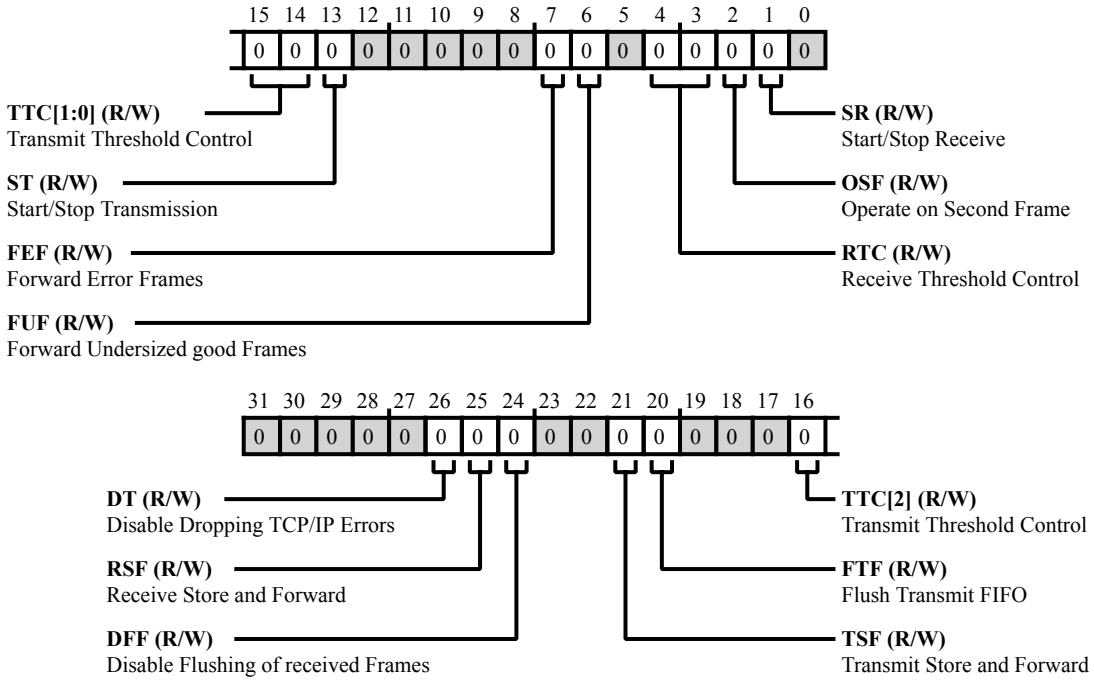

Figure 23-24: EMAC_DMA_OPMODE Register Diagram

Table 23-54: EMAC_DMA_OPMODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
26 (R/W)	DT	Disable Dropping TCP/IP Errors. The EMAC_DMA_OPMODE.DT bit, when set, directs the core not to drop frames that only have errors detected by the Receive Checksum Offload engine. Such frames do not have any errors (including FCS error) in the Ethernet frame received by the MAC but have errors in the encapsulated payload only. When this bit is reset, all error frames are dropped if the EMAC_DMA_OPMODE.FEF bit is reset.
25 (R/W)	RSF	Receive Store and Forward. The EMAC_DMA_OPMODE.RSF bit, when set, directs the MFL only to read a frame from the Rx FIFO after the complete frame has been written to it, ignoring the EMAC_DMA_OPMODE.RTC bits. When this bit is reset, the Rx FIFO operates in threshold mode, subject to the threshold specified by the EMAC_DMA_OPMODE.RTC bits.
24 (R/W)	DFF	Disable Flushing of received Frames. The EMAC_DMA_OPMODE.DFF bit, when set, directs the Rx DMA not to flush any frames because of the unavailability of receive descriptors/buffers as it does normally when this bit is reset.

Table 23-54: EMAC_DMA_OPMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W)	TSF	Transmit Store and Forward. The EMAC_DMA_OPMODE.TSF bit, when set, starts transmission when a full frame resides in the MFL Transmit FIFO. When this bit is set, the TTC values specified in Register 6[16:14] are ignored. This bit should be changed only when transmission is stopped.	
20 (R/W)	FTF	Flush Transmit FIFO. The EMAC_DMA_OPMODE.FTF bit, when set, directs the transmit FIFO controller logic to reset to its default values and thus all data in the Tx FIFO is lost/flushed. This bit is cleared internally when the flushing operation is completed fully. The Operation Mode register should not be written to until this bit is cleared. The data which is already accepted by the MAC transmitter is not flushed. It is scheduled for transmission and results in underflow and runt frame transmission. Note: The flush operation completes only after emptying the Tx FIFO of its contents and all the pending Transmit Status of the transmitted frames are accepted by the host. In order to complete this flush operation, the PHY transmit clock is required to be active.	
16:14 (R/W)	TTC	Transmit Threshold Control. The EMAC_DMA_OPMODE.TTC bits control the threshold level of the MFL Transmit FIFO. Transmission starts when the frame size within the MFL Transmit FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are also transmitted. These bits are used only when the EMAC_DMA_OPMODE.TSF bit is reset. The value =011 is not used.	
		0	64
		1	128
		2	192
		3	256
		4	40
		5	32
		6	24
		7	16
13 (R/W)	ST	Start/Stop Transmission. The EMAC_DMA_OPMODE.ST bit, when set, places transmission in the Running state, and the DMA checks the Transmit List at the current position for a frame to be transmitted. Descriptor acquisition is attempted either from the current position in the list, which is the Transmit List Base Address set by Transmit Descriptor List Address, or from the position retained when transmission was stopped previously. If the current descriptor is not owned by the DMA, transmission enters the Suspended state, and the EMAC_DMA_STAT.TU bit is set.	

Table 23-54: EMAC_DMA_OPMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		The Start Transmission command is effective only when transmission is stopped. If the command is issued before setting the EMAC_DMA_TXDSC_CUR address register, then the DMA behavior is unpredictable. When this bit is reset, the transmission process is placed in the Stopped state after completing the transmission of the current frame. The Next Descriptor position in the Transmit List is saved, and becomes the current position when transmission is restarted. The stop transmission command is effective only when the transmission of the current frame is complete or the transmission is in the Suspended state.
7 (R/W)	FEF	Forward Error Frames. The EMAC_DMA_OPMODE.FEF bit, when reset, directs the Rx FIFO to drop frames with error status (CRC error, collision error, giant frame, watchdog timeout, overflow). However, if the frames start byte (write) pointer is already transferred to the read controller side (in Threshold mode), then the frames are not dropped. When EMAC_DMA_OPMODE.FEF bit is set, all frames except runt error frames are forwarded to the DMA. But when Rx FIFO overflows when a partial frame is written, then such frames are dropped even when EMAC_DMA_OPMODE.FEF is set.
6 (R/W)	FUF	Forward Undersized good Frames. The EMAC_DMA_OPMODE.FUF bit, when set, directs the Rx FIFO to forward Undersized frames (frames with no Error and length less than 64 bytes) including pad-bytes and CRC). When reset, the Rx FIFO drops all frames of less than 64 bytes, unless it is already transferred because of lower value of Receive Threshold (for example, EMAC_DMA_OPMODE.RTC =01).
4:3 (R/W)	RTC	Receive Threshold Control. The EMAC_DMA_OPMODE.RTC bits control the threshold level of the MFL Receive FIFO. Transfer (request) to DMA starts when the frame size within the MFL Receive FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are transferred automatically. These bits are valid only when the EMAC_DMA_OPMODE.RSF bit is zero, and are ignored when the EMAC_DMA_OPMODE.RSF bit is set to 1. The value =11 is not used.
		0 64
		1 32
		2 96
		3 128
2 (R/W)	OSF	Operate on Second Frame. The EMAC_DMA_OPMODE.OSF bit, when set, instructs the DMA to process a second frame of Transmit data even before status for first frame is obtained.
1 (R/W)	SR	Start/Stop Receive.

Table 23-54: EMAC_DMA_OPMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		The EMAC_DMA_OPMODE.SR bit, when set, places the Receive process in the Running state. The DMA attempts to acquire the descriptor from the Receive list and processes incoming frames. Descriptor acquisition is attempted from the current position in the list, which is the address set by DMA Receive Descriptor List Address or the position retained when the Receive process was previously stopped. If no descriptor is owned by the DMA, reception is suspended, and the EMAC_DMA_STAT.RU bit is set. The Start Receive command is effective only when reception has stopped. If the command was issued before setting EMAC_DMA_RXDSC_CURaddress register, DMA behavior is unpredictable. When this bit is cleared, Rx DMA operation is stopped after the transfer of the current frame. The next descriptor position in the Receive list is saved and becomes the current position after the Receive process is restarted. The Stop Receive command is effective only when the Receive process is in either the Running (waiting for receive packet) or in the Suspended state.

DMA Rx Buffer Current Register

The EMAC_DMA_RXBUF_CUR register holds the pointer to the current receive DMA buffer.

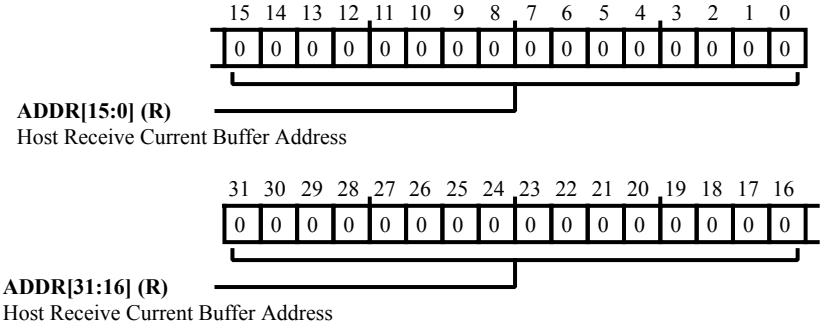

Figure 23-25: EMAC_DMA_RXBUF_CUR Register Diagram

Table 23-55: EMAC_DMA_RXBUF_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Receive Current Buffer Address. The EMAC_DMA_RXBUF_CUR.ADDR bit field points to the current Receive Buffer address being read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

DMA Rx Descriptor List Address Register

The EMAC_DMA_RXDSC_ADDR register holds the address for the DMA receive descriptor list. Writing to this Register is permitted only when reception is stopped. When stopped, this must be written to before the receive Start command is given. The processor can write to EMAC_DMA_RXDSC_ADDR only when Rx DMA has stopped (EMAC_DMA_OPMODE.SR bit =0). When stopped, it can be written with a new descriptor list address. When the processor sets the EMAC_DMA_OPMODE.SR bit to 1, the DMA takes the newly programmed descriptor base address. If this register is not changed when the EMAC_DMA_OPMODE.SR bit is cleared to 0, the DMA takes the descriptor address where it was stopped earlier.

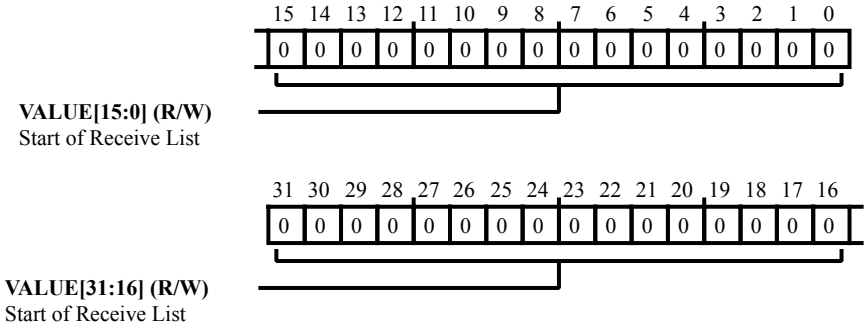

Figure 23-26: EMAC_DMA_RXDSC_ADDR Register Diagram

Table 23-56: EMAC_DMA_RXDSC_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Start of Receive List. The EMAC_DMA_RXDSC_ADDR.VALUE bit field contains the base address of the First Descriptor in the Receive Descriptor list. The LSB bits [1:0] for the 32bit bus width are ignored and are taken as all-zero by the DMA internally. Therefore, these LSB bits are Read-Only (RO).

DMA Rx Descriptor Current Register

The EMAC_DMA_RXDSC_CUR register contains the current DMA receive descriptor.

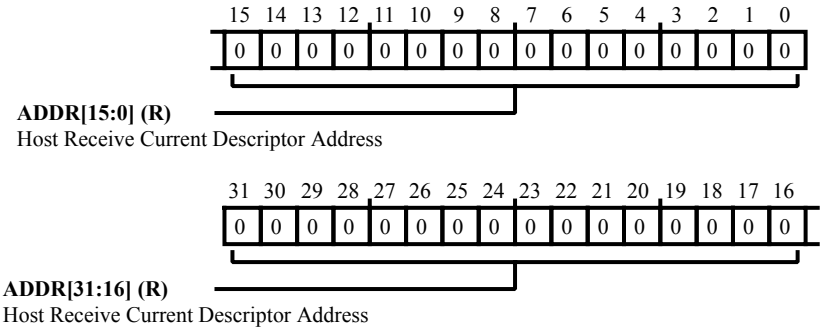

Figure 23-27: EMAC_DMA_RXDSC_CUR Register Diagram

Table 23-57: EMAC_DMA_RXDSC_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Receive Current Descriptor Address. The EMAC_DMA_RXDSC_CUR.ADDR bit field points to the start address of the current Receive Descriptor read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

DMA Rx Interrupt Watch Dog Register

The EMAC_DMA_RXIWDOG register contains the timeout value for the EMAC DMA receive interrupt watch dog timer.

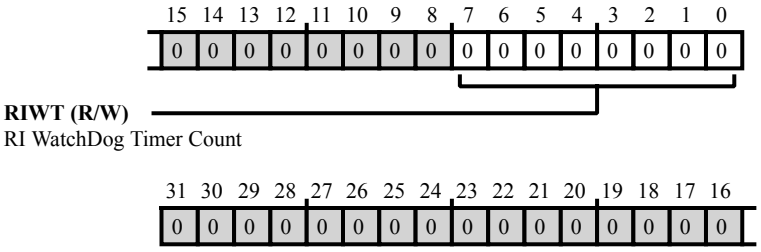

Figure 23-28: EMAC_DMA_RXIWDOG Register Diagram

Table 23-58: EMAC_DMA_RXIWDOG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	RIWT	RI WatchDog Timer Count. The EMAC_DMA_RXIWDOG.RIWT bit field indicates the number of system clock cycles multiplied by 256 for which the watchdog timer is set. The watchdog timer gets triggered with the programmed value after the Rx DMA completes the transfer of a frame for which the RI status bit is not set because of the setting in the corresponding descriptor RDES1[31]. When the watch-dog timer runs out, the RI bit is set and the timer is stopped. The watchdog timer is reset when EMAC_DMA_STAT.RI bit is set high because of automatic setting of EMAC_DMA_STAT.RI as per RDES1[31] of any received frame.

DMA Rx Poll Demand register

The EMAC_DMA_RXPOLL register directs the EMAC to poll the receive descriptor list.

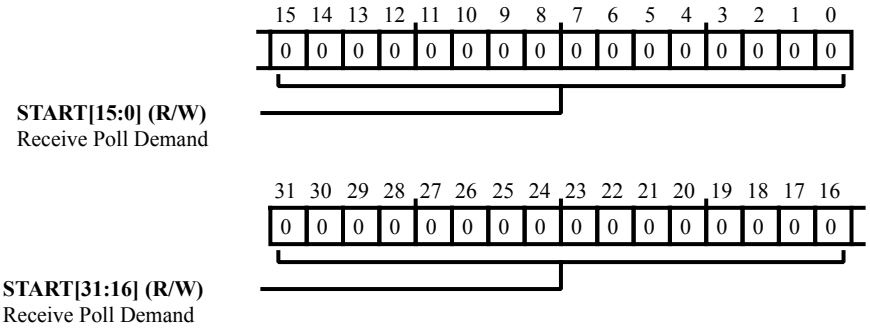

Figure 23-29: EMAC_DMA_RXPOLL Register Diagram

Table 23-59: EMAC_DMA_RXPOLL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	START	Receive Poll Demand. The EMAC_DMA_RXPOLL.START bits, when written with any value, cause the DMA to read the current descriptor pointed to by the EMAC_DMA_RXDSC_CUR register. If that descriptor is not available (owned by application), reception returns to the Suspended state, and the EMAC_DMA_STAT.RU bit is asserted. If the descriptor is available, the Receive DMA returns to the active state.

DMA Status Register

The EMAC_DMA_STAT register indicates EMAC DMA status.

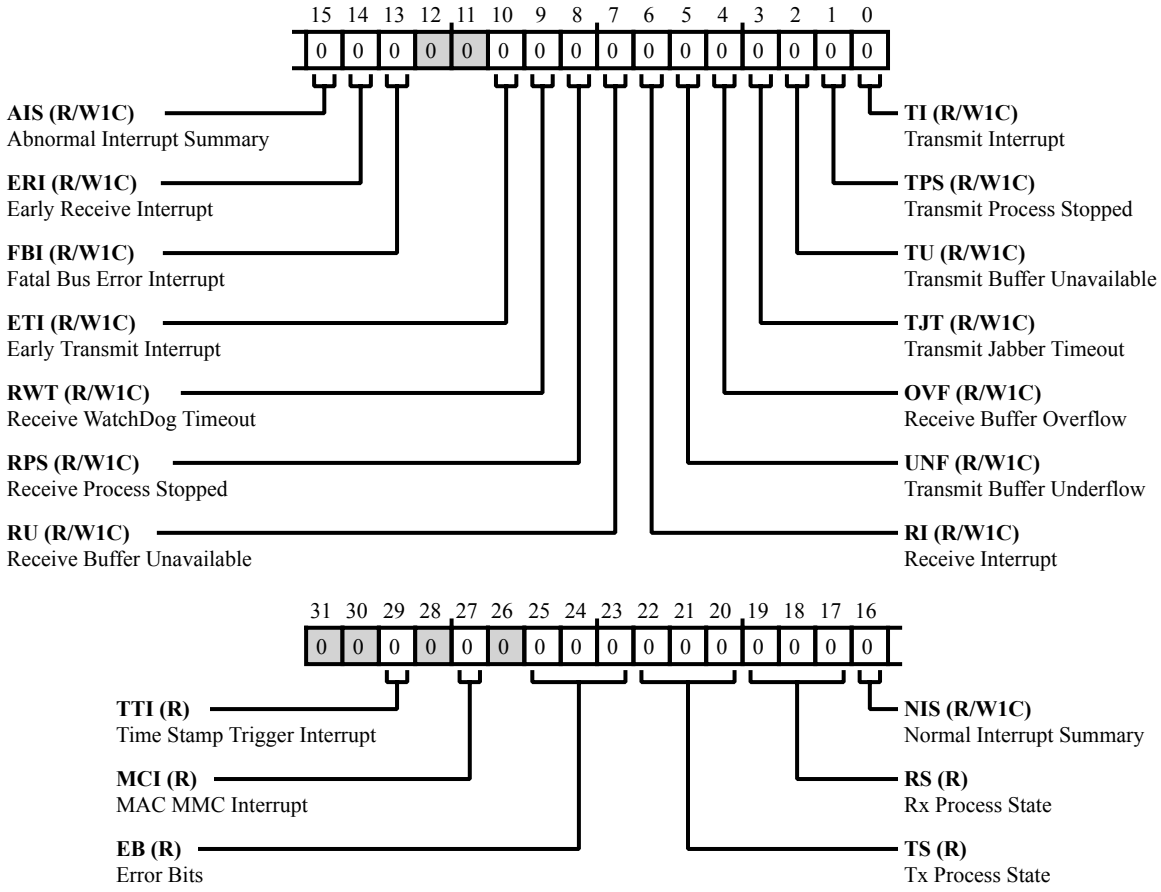

Figure 23-30: EMAC_DMA_STAT Register Diagram

Table 23-60: EMAC_DMA_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/NW)	TTI	Time Stamp Trigger Interrupt. The EMAC_DMA_STAT.TTI bit indicates an interrupt event in the MAC core's Time Stamp Generator block. The software must read the corresponding registers in the MAC core to get the exact cause of interrupt and clear its source to reset this bit to =0. When this bit is high, the interrupt signal from the MAC is high.
27 (R/NW)	MCI	MAC MMC Interrupt. The EMAC_DMA_STAT.MCI bit reflects an interrupt event in the MMC module of the MAC core. The software must read the corresponding registers in the MAC core to get the exact cause of interrupt and clear the source of interrupt to make this bit as =0. The interrupt signal from the MAC is high when this bit is high.

Table 23-60: EMAC_DMA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
25:23 (R/NW)	EB	Error Bits. The EMAC_DMA_STAT.EB bits indicate the type of error that caused a Bus Error (for example, error response on the SCB interface). These bits are valid only when the EMAC_DMA_STAT.FBI bit is set. This field does not generate an interrupt.	
		0	Error during data buffer access, write transfer, Rx DMA
		1	Error during data buffer access, write transfer, Tx DMA
		2	Error during data buffer access, read transfer, Rx DMA
		3	Error during data buffer access, read transfer, Tx DMA
		4	Error during descriptor access, write transfer, Rx DMA
		5	Error during descriptor access, write transfer, Tx DMA
		6	Error during descriptor access, read transfer, Rx DMA
		7	Error during descriptor access, read transfer, Tx DMA
22:20 (R/NW)	TS	Tx Process State. The EMAC_DMA_STAT.TS bits indicate the transmit DMA state. This field does not generate an interrupt.	
		0	Stopped; Reset or Stop Tx Command Issued
		1	Running; Fetching Tx Transfer Descriptor
		2	Running; Waiting for Status
		3	Reading Data from Host Memory Buffer and Queuing It to Tx Buffer
		4	TIME_STAMP Write State
		5	Reserved
		6	Suspended; Tx Descriptor Unavailable or Tx Buffer Underflow
		7	Closing Tx Descriptor
19:17 (R/NW)	RS	Rx Process State. The EMAC_DMA_STAT.RS bits indicate the receive DMA state. This field does not generate an interrupt.	
		0	Stopped: Reset or Stop Rx Command Issued.
		1	Running: Fetching Rx Transfer Descriptor.
		2	Reserved
		3	Running: Waiting for Rx Packet
		4	Suspended: Rx Descriptor Unavailable

Table 23-60: EMAC_DMA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		5	Running: Closing Rx Descriptor
		6	TIME_STAMP Write State
		7	Running: Transferring Rx Packet Data from Rx Buffer to Host Memory
16 (R/W1C)	NIS	Normal Interrupt Summary. The value of the EMAC_DMA_STAT.NIS bit field is the logical OR of the following when the corresponding interrupt bits are enabled in DMA Interrupt Enable Register: EMAC_DMA_STAT.TI, EMAC_DMA_STAT.TU, EMAC_DMA_STAT.RI, and EMAC_DMA_STAT.ERI. Only unmasked bits affect the Normal Interrupt Summary bit. This is a sticky bit and must be cleared (by writing a 1 to this bit) each time a corresponding bit that causes EMAC_DMA_STAT.NIS to be set is cleared.	
15 (R/W1C)	AIS	Abnormal Interrupt Summary. The value of the EMAC_DMA_STAT.AIS bit field is the logical OR of the following when the corresponding interrupt bits are enabled in DMA Interrupt Enable Register: EMAC_DMA_IEN.TPS, EMAC_DMA_IEN.TJT, EMAC_DMA_IEN.OVF, EMAC_DMA_IEN.UNF, EMAC_DMA_IEN.RU, EMAC_DMA_IEN.RPS, EMAC_DMA_IEN.RWT, EMAC_DMA_IEN.ETI, and EMAC_DMA_IEN.FBI. Only unmasked bits affect the Abnormal Interrupt Summary bit. This is a sticky bit and must be cleared each time a corresponding bit that causes EMAC_DMA_STAT.AIS to be set is cleared.	
14 (R/W1C)	ERI	Early Receive Interrupt. The EMAC_DMA_STAT.ERI bit indicates that the DMA had filled the first data buffer of the packet. The EMAC_DMA_STAT.RI bit automatically clears this bit.	
13 (R/W1C)	FBI	Fatal Bus Error Interrupt. The EMAC_DMA_STAT.FBI bit indicates that a bus error occurred, as detailed in the EMAC_DMA_STAT.EB field. When this bit is set, the corresponding DMA engine disables all its bus accesses.	
10 (R/W1C)	ETI	Early Transmit Interrupt. The EMAC_DMA_STAT.ETI bit indicates that the frame to be transmitted was fully transferred to the MFL Transmit FIFO.	
9 (R/W1C)	RWT	Receive WatchDog Timeout. The EMAC_DMA_STAT.RWT bit is asserted when a frame with a length greater than 2,048 bytes is received (10, 240 when Jumbo Frame mode is enabled).	
8 (R/W1C)	RPS	Receive Process Stopped. The EMAC_DMA_STAT.RPS bit is asserted when the Receive Process enters the Stopped state.	
7	RU	Receive Buffer Unavailable.	

Table 23-60: EMAC_DMA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(R/W1C)		The EMAC_DMA_STAT.RU bit indicates that the Next Descriptor in the Receive List is owned by the application and cannot be acquired by the DMA. Receive Process is suspended. To resume processing Receive descriptors, the application should change the ownership of the descriptor and issue a Receive Poll Demand command. If no Receive Poll Demand is issued, Receive Process resumes when the next recognized incoming frame is received. This bit is set only when the previous Receive Descriptor was owned by the DMA.
6 (R/W1C)	RI	Receive Interrupt. The EMAC_DMA_STAT.RI bit indicates the completion of frame reception. Specific frame status information has been posted in the descriptor. Reception remains in the Running state.
5 (R/W1C)	UNF	Transmit Buffer Underflow. The EMAC_DMA_STAT.UNF bit indicates that the Transmit Buffer had an Underflow during frame transmission. Transmission is suspended and an Underflow Error TDES0[1] is set.
4 (R/W1C)	OVF	Receive Buffer Overflow. The EMAC_DMA_STAT.OVF bit indicates that the Receive Buffer had an Overflow during frame reception. If the partial frame is transferred to application, the overflow status is set in RDES0[11].
3 (R/W1C)	TJT	Transmit Jabber Timeout. The EMAC_DMA_STAT.TJT bit indicates that the Transmit Jabber Timer expired, meaning that the transmitter had been excessively active. The transmission process is aborted and placed in the Stopped state. This causes the Transmit Jabber Timeout TDES0[14] flag to assert.
2 (R/W1C)	TU	Transmit Buffer Unavailable. The EMAC_DMA_STAT.TU bit indicates that the Next Descriptor in the Transmit List is owned by the application and cannot be acquired by the DMA. Transmission is suspended. The value in the EMAC_DMA_STAT.TS bits explain the Transmit Process state transitions. To resume processing transmit descriptors, the application should change the ownership of the bit of the descriptor and then issue a Transmit Poll Demand command.
1 (R/W1C)	TPS	Transmit Process Stopped. The EMAC_DMA_STAT.TPS bit is set when the transmission is stopped.
0 (R/W1C)	TI	Transmit Interrupt. The EMAC_DMA_STAT.TI bit indicates that frame transmission is finished and TDES1[31] is set in the First Descriptor.

DMA Tx Buffer Current Register

The EMAC_DMA_TXBUF_CUR register holds the pointer to the current transmit DMA buffer.

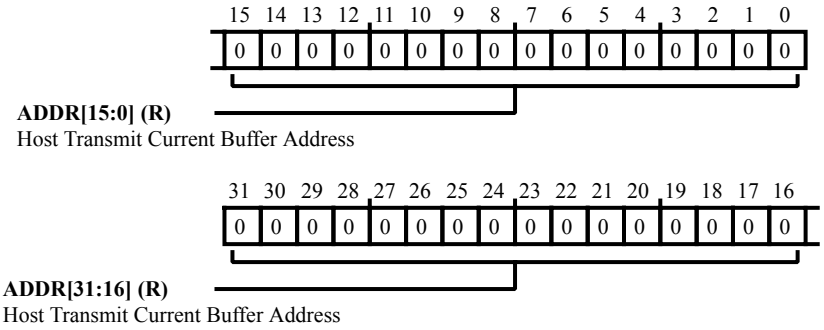

Figure 23-31: EMAC_DMA_TXBUF_CUR Register Diagram

Table 23-61: EMAC_DMA_TXBUF_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Transmit Current Buffer Address. The EMAC_DMA_TXBUF_CUR.ADDR bit field points to the current Transmit Buffer Address being read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

DMA Tx Descriptor List Address Register

The EMAC_DMA_TXDSC_ADDR register holds the address for the DMA transmit descriptor list. The processor can write to this Register only when Tx DMA has stopped (EMAC_DMA_OPMODE.ST bit =0). When stopped, this can be written with a new descriptor list address. When the processor sets the EMAC_DMA_OPMODE.ST bit to 1, the DMA takes the newly programmed descriptor base address. If this register is not changed when the EMAC_DMA_OPMODE.ST bit is cleared to 0, then the DMA takes the descriptor address where it was stopped earlier.

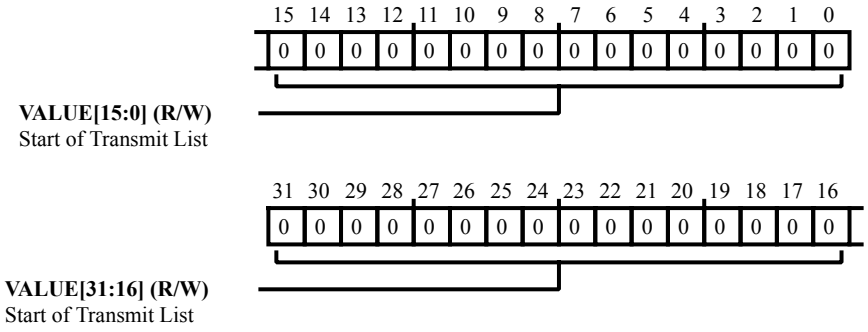

Figure 23-32: EMAC_DMA_TXDSC_ADDR Register Diagram

Table 23-62: EMAC_DMA_TXDSC_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Start of Transmit List. The EMAC_DMA_TXDSC_ADDR.VALUE bit field contains the base address of the First Descriptor in the Transmit Descriptor list. The LSB bits [1:0] for 32bit bus width are ignored and are taken as all-zero by the DMA internally. Therefore, these LSB bits are Read-Only (RO).

DMA Tx Descriptor Current Register

The EMAC_DMA_TXDSC_CUR register contains the current DMA transmit descriptor.

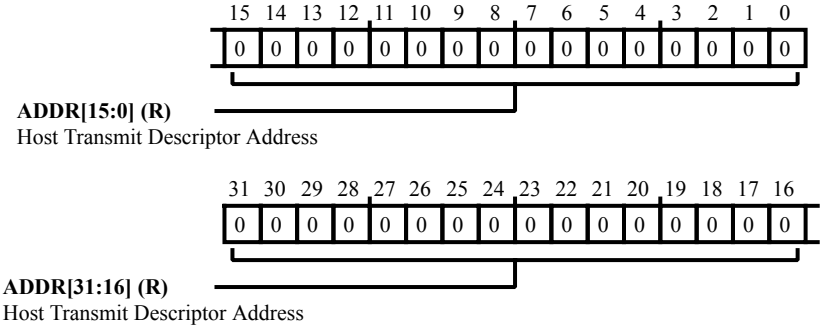

Figure 23-33: EMAC_DMA_TXDSC_CUR Register Diagram

Table 23-63: EMAC_DMA_TXDSC_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Transmit Descriptor Address. The EMAC_DMA_TXDSC_CUR.ADDR bit field points to the start address of the current Transmit Descriptor read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

DMA Tx Poll Demand Register

The EMAC_DMA_TXPOLL register directs the EMAC to poll the transmit descriptor list.

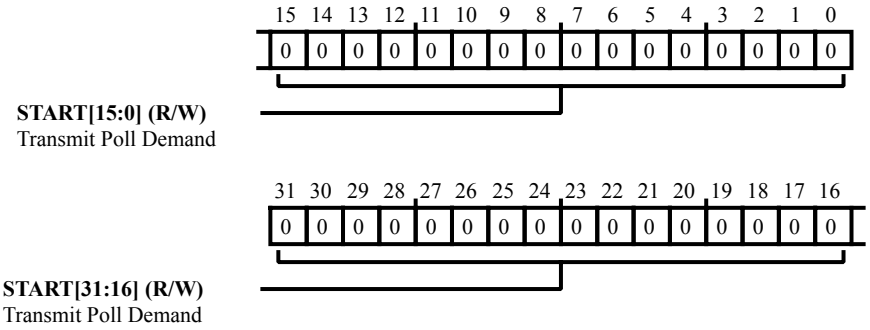

Figure 23-34: EMAC_DMA_TXPOLL Register Diagram

Table 23-64: EMAC_DMA_TXPOLL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	START	Transmit Poll Demand. The EMAC_DMA_TXPOLL.START bits, when written with any value, cause the DMA to read the current descriptor pointed to by EMAC_DMA_TXDSC_CUR register. If that descriptor is not available (owned by application), transmission returns to the Suspend state, and the EMAC_DMA_STAT.TU bit is asserted. If the descriptor is available, transmission resumes.

FLow Control Register

The EMAC_FLOWCTL register controls EMAC flow control features.

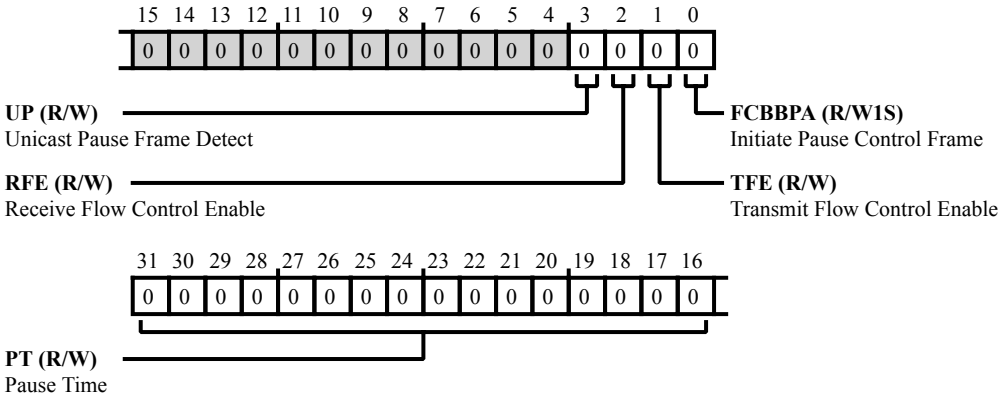

Figure 23-35: EMAC_FLOWCTL Register Diagram

Table 23-65: EMAC_FLOWCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	PT	Pause Time. The EMAC_FLOWCTL.PT bits hold the value to be used in the Pause Time field in the transmit control frame.
3 (R/W)	UP	Unicast Pause Frame Detect. The EMAC_FLOWCTL.UP bit, when set, directs the MAC to detect the Pause frames with the station's unicast address specified in EMAC_ADDR0_HI and EMAC_ADDR0_LO address registers. This bit also directs the MAC to the detect Pause frames with the unique multicast address. When this bit is reset, the MAC will detect only a Pause frame with the unique multicast address specified in the 802.3x standard.
2 (R/W)	RFE	Receive Flow Control Enable. The EMAC_FLOWCTL.RFE bit, when set, directs the MAC to decode the received Pause frame and disable its transmitter for a specified (Pause Time) time. When this bit is reset, the decode function of the Pause frame is disabled.
1 (R/W)	TFE	Transmit Flow Control Enable. In Full-Duplex mode, when the EMAC_FLOWCTL.TFE bit is set, the MAC enables the flow control operation to transmit Pause frames. When this bit is reset, the flow control operation in the MAC is disabled, and the MAC does not transmit any Pause frames. In Half-Duplex mode, when this bit is set, the MAC enables the back pressure operation. When this bit is reset, the back pressure feature is disabled.
0 (R/W1S)	FCBBPA	Initiate Pause Control Frame.

Table 23-65: EMAC_FLOWCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		The EMAC_FLOWCTL.FCBBPA bit initiates a Pause Control frame in Full-Duplex mode and activates the back pressure function in Half-Duplex mode if TFE bit is set. In Full-Duplex mode, this bit should be read as =0 before writing to the EMAC_FLOWCTL register. To initiate a Pause control frame, the Application must set this bit to =1. During a transfer of the Control Frame, this bit continues to be set to signify that a frame transmission is in progress. After the completion of Pause control frame transmission, the MAC resets this bit to =0. The EMAC_FLOWCTL register should not be written to until this bit is cleared. In Half-Duplex mode, when this bit is set (and EMAC_FLOWCTL.TFE is set), the back pressure is asserted by the MAC Core. During back pressure, when the MAC receives a new frame, the transmitter starts sending a JAM pattern resulting in a collision. The EMAC_FLOWCTL.FCBBPA bit is logically OR'ed with the flow control input signal for the back pressure function. When the MAC is configured to Full-Duplex mode, the back pressure function is automatically disabled.

Hash Table High Register

The EMAC_HASHTBL_HI register contains the upper 32 bits of the hash table.

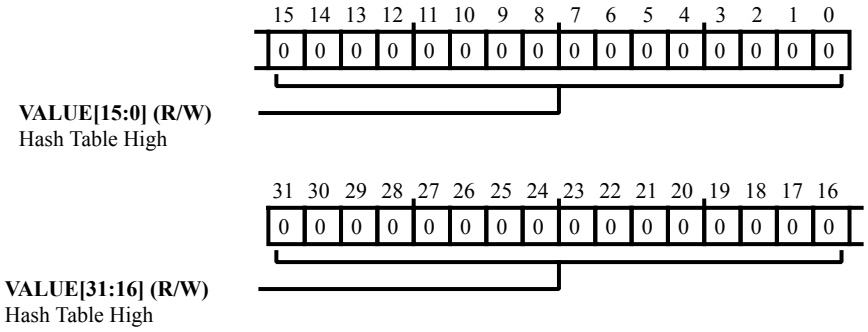

Figure 23-36: EMAC_HASHTBL_HI Register Diagram

Table 23-66: EMAC_HASHTBL_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Hash Table High. The EMAC_HASHTBL_HI.VALUE bits contain the upper 32 bits of Hash table.

Hash Table Low Register

The EMAC_HASHTBL_LO register contains the lower 32 bits of the hash table.

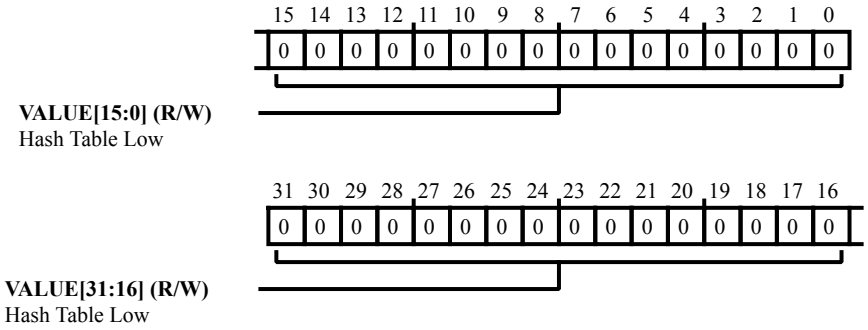

Figure 23-37: EMAC_HASHTBL_LO Register Diagram

Table 23-67: EMAC_HASHTBL_LO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Hash Table Low. The EMAC_HASHTBL_LO.VALUE bits contain the lower 32 bits of Hash table.

Interrupt Mask Register

The EMAC_IMSK register enables (unmasks) EMAC interrupts.

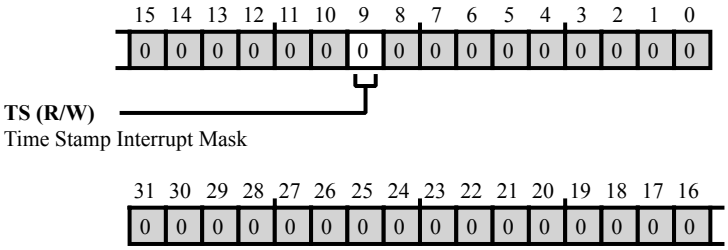

Figure 23-38: EMAC_IMSK Register Diagram

Table 23-68: EMAC_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W)	TS	Time Stamp Interrupt Mask. The EMAC_IMSK.TS bit, when set, disables the assertion of the interrupt signal, which is generated when the EMAC_ISTAT.TS bit is set.

MMC IPC Rx Interrupt Mask Register

The EMAC_IPC_RXIMSK register enables (unmasks) MMC IPC receive interrupts.

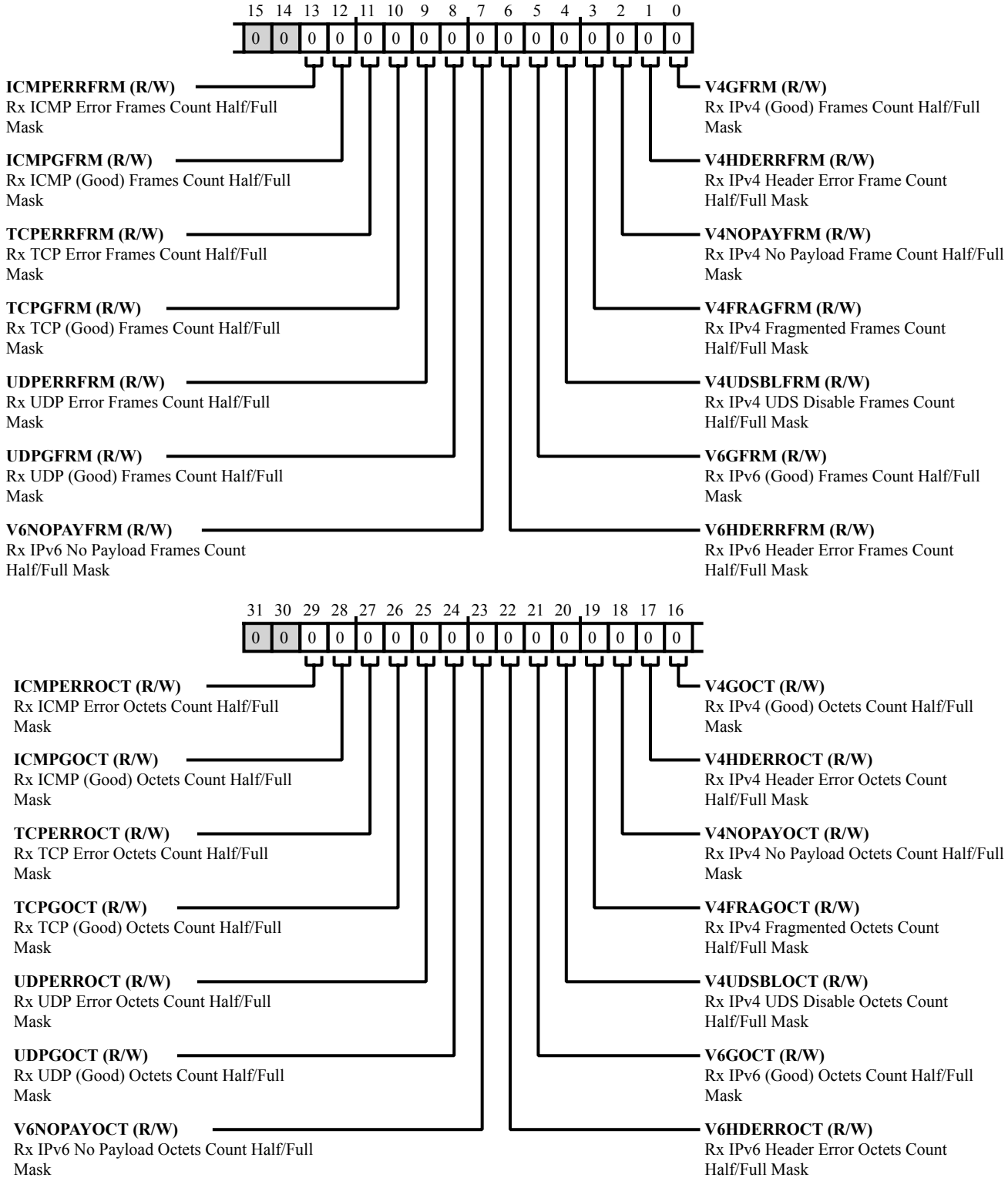

Figure 23-39: EMAC_IPC_RXIMSK Register Diagram

Table 23-69: EMAC_IPC_RXIMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/W)	ICMPERROCT	Rx ICMP Error Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.ICMPERROCT bit, when set, masks the interrupt when the EMAC_RXICMP_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
28 (R/W)	ICMPGOCT	Rx ICMP (Good) Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.ICMPGOCT bit, when set, masks the interrupt when the EMAC_RXICMP_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
27 (R/W)	TCPERROCT	Rx TCP Error Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.TCPERROCT bit, when set, masks the interrupt when the EMAC_RXTCP_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
26 (R/W)	TCPGOCT	Rx TCP (Good) Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.TCPGOCT bit, when set, masks the interrupt when the EMAC_RXTCP_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
25 (R/W)	UDPERROCT	Rx UDP Error Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.UDPERROCT bit, when set, masks the interrupt when the EMAC_RXUDP_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
24 (R/W)	UDPGOCT	Rx UDP (Good) Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.UDPGOCT bit, when set, masks the interrupt when the EMAC_RXUDP_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
23 (R/W)	V6NOPAYOCT	Rx IPv6 No Payload Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.V6NOPAYOCT bit, when set, masks the interrupt when the EMAC_RXIPV6_NOPAY_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
22 (R/W)	V6HDERROCT	Rx IPv6 Header Error Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.V6HDERROCT bit, when set, masks the interrupt when the EMAC_RXIPV6_HDR_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
21 (R/W)	V6GOCT	Rx IPv6 (Good) Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.V6GOCT bit, when set, masks the interrupt when the EMAC_RXIPV6_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
20	V4UDSBLOCT	Rx IPv4 UDS Disable Octets Count Half/Full Mask.

Table 23-69: EMAC_IPC_RXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(R/W)		The EMAC_IPC_RXIMSK.V4UDSBLOCT bit, when set, masks the interrupt when the EMAC_RXIPV4_UDSBL_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
19 (R/W)	V4FRAGOCT	Rx IPv4 Fragmented Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.V4FRAGOCT bit, when set, masks the interrupt when the EMAC_RXIPV4_FRAG_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
18 (R/W)	V4NOPAYOCT	Rx IPv4 No Payload Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.V4NOPAYOCT bit, when set, masks the interrupt when the EMAC_RXIPV4_NOPAY_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
17 (R/W)	V4HDERROCT	Rx IPv4 Header Error Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.V4HDERROCT bit, when set, masks the interrupt when the EMAC_RXIPV4_HDR_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
16 (R/W)	V4GOCT	Rx IPv4 (Good) Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.V4GOCT bit, when set, masks the interrupt when the EMAC_RXIPV4_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
13 (R/W)	ICMPERRFRM	Rx ICMP Error Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.ICMPERRFRM bit, when set, masks the interrupt when the EMAC_RXICMP_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
12 (R/W)	ICMPGFRM	Rx ICMP (Good) Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.ICMPGFRM bit, when set, masks the interrupt when the EMAC_RXICMP_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
11 (R/W)	TCPERRFRM	Rx TCP Error Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.TCPERRFRM bit, when set, masks the interrupt when the EMAC_RXTCP_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
10 (R/W)	TCPGFRM	Rx TCP (Good) Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.TCPGFRM bit, when set, masks the interrupt when the EMAC_RXTCP_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
9 (R/W)	UDPERRFRM	Rx UDP Error Frames Count Half/Full Mask.

Table 23-69: EMAC_IPC_RXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		The EMAC_IPC_RXIMSK.UDPERRFRM bit, when set, masks the interrupt when the EMAC_RXUDP_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
8 (R/W)	UDPGFRM	Rx UDP (Good) Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.UDPGFRM bit, when set, masks the interrupt when the EMAC_RXUDP_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
7 (R/W)	V6NOPAYFRM	Rx IPv6 No Payload Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.V6NOPAYFRM bit, when set, masks the interrupt when the EMAC_RXIPV6_NOPAY_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
6 (R/W)	V6HDERRFRM	Rx IPv6 Header Error Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.V6HDERRFRM bit, when set, masks the interrupt when the EMAC_RXIPV6_HDR_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
5 (R/W)	V6GFRM	Rx IPv6 (Good) Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.V6GFRM bit, when set, masks the interrupt when the EMAC_RXIPV6_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
4 (R/W)	V4UDSBLFRM	Rx IPv4 UDS Disable Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.V4UDSBLFRM bit, when set, masks the interrupt when the EMAC_RXIPV4_UDSBL_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
3 (R/W)	V4FRAGFRM	Rx IPv4 Fragmented Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.V4FRAGFRM bit, when set, masks the interrupt when the EMAC_RXIPV4_FRAG_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
2 (R/W)	V4NOPAYFRM	Rx IPv4 No Payload Frame Count Half/Full Mask. The EMAC_IPC_RXIMSK.V4NOPAYFRM bit, when set, masks the interrupt when the EMAC_RXIPV4_NOPAY_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
1 (R/W)	V4HDERRFRM	Rx IPv4 Header Error Frame Count Half/Full Mask. The EMAC_IPC_RXIMSK.V4HDERRFRM bit, when set, masks the interrupt when the EMAC_RXIPV4_HDR_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
0 (R/W)	V4GFRM	Rx IPv4 (Good) Frames Count Half/Full Mask.

Table 23-69: EMAC_IPC_RXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		The EMAC_IPC_RXIMSK.V4GFRM bit, when set, masks the interrupt when the EMAC_RXIPV4_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.

MMC IPC Rx Interrupt Register

The EMAC_IPC_RXINT register indicates status of MMC IPC receive interrupts.

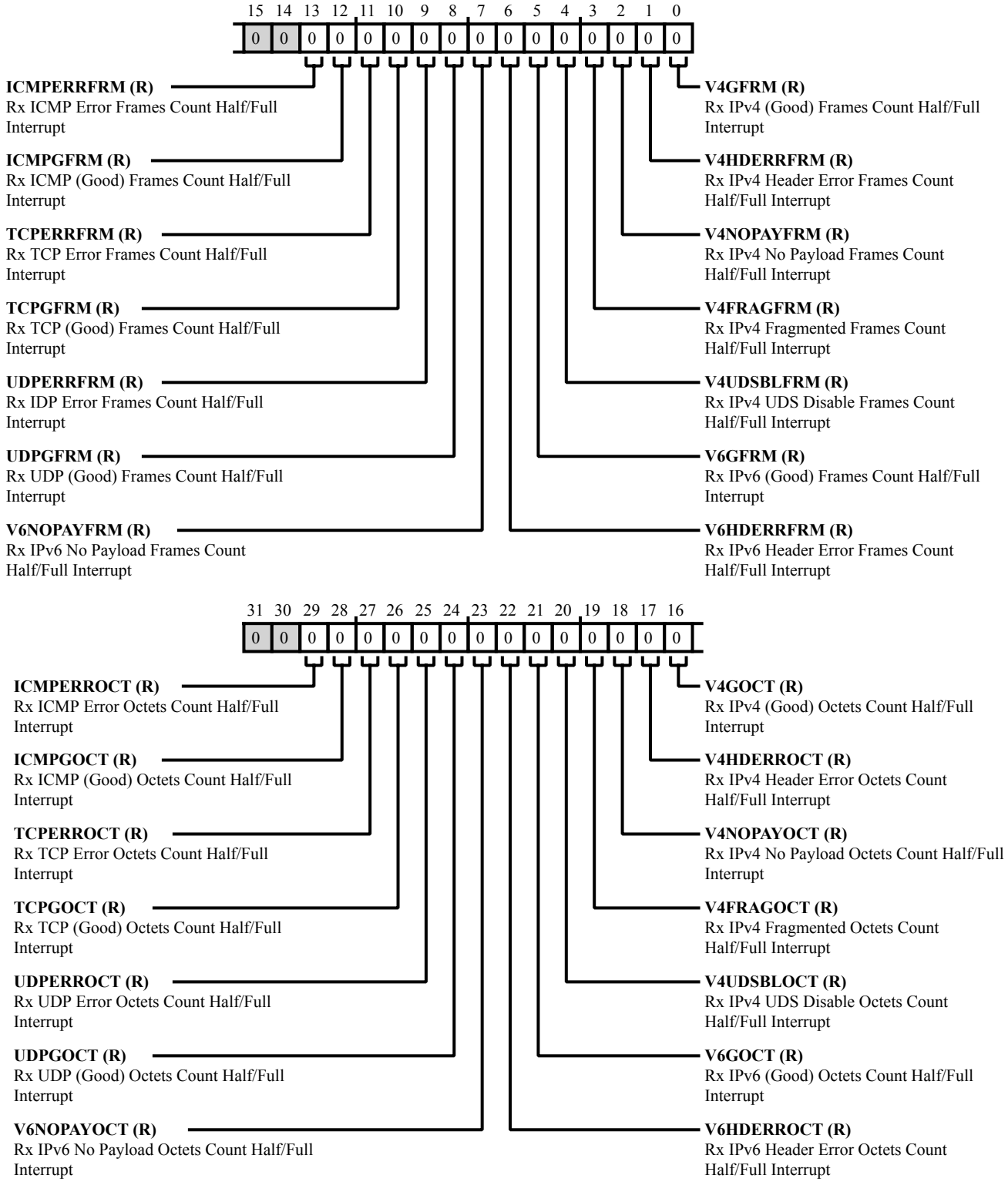

Figure 23-40: EMAC_IPC_RXINT Register Diagram

Table 23-70: EMAC_IPC_RXINT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/NW)	ICMPERROCT	Rx ICMP Error Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.ICMPERROCT bit is set when the EMAC_RXICMP_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
28 (R/NW)	ICMPGOCT	Rx ICMP (Good) Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.ICMPGOCT bit is set when the EMAC_RXICMP_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
27 (R/NW)	TCPERROCT	Rx TCP Error Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.TCPERROCT bit is set when the EMAC_RXTCP_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
26 (R/NW)	TCPGOCT	Rx TCP (Good) Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.TCPGOCT bit is set when the EMAC_RXTCP_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
25 (R/NW)	UDPERROCT	Rx UDP Error Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.UDPERROCT bit is set when the EMAC_RXUDP_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
24 (R/NW)	UDPGOCT	Rx UDP (Good) Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.UDPGOCT bit is set when the EMAC_RXUDP_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
23 (R/NW)	V6NOPAYOCT	Rx IPv6 No Payload Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.V6NOPAYOCT bit is set when the EMAC_RXIPV6_NOPAY_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
22 (R/NW)	V6HDERROCT	Rx IPv6 Header Error Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.V6HDERROCT bit is set when the EMAC_RXIPV6_HDR_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
21 (R/NW)	V6GOCT	Rx IPv6 (Good) Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.V6GOCT bit is set when the EMAC_RXIPV6_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
20 (R/NW)	V4UDSBLOCT	Rx IPv4 UDS Disable Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4UDSBLOCT bit is set when the EMAC_RXIPV4_UDSBL_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
19 (R/NW)	V4FRAGOCT	Rx IPv4 Fragmented Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4FRAGOCT bit is set when the EMAC_RXIPV4_FRAG_OCT counter reaches half the maximum value, and also when it reaches the maximum value.

Table 23-70: EMAC_IPC_RXINT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/NW)	V4NOPAYOCT	Rx IPv4 No Payload Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4NOPAYOCT bit set when the EMAC_RXIPV4_NOPAY_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
17 (R/NW)	V4HDERROCT	Rx IPv4 Header Error Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4HDERROCT bit is set when the EMAC_RXIPV4_HDR_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
16 (R/NW)	V4GOCT	Rx IPv4 (Good) Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4GOCT bit is set when the EMAC_RXIPV4_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
13 (R/NW)	ICMPERRFRM	Rx ICMP Error Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.ICMPERRFRM bit is set when the EMAC_RXICMP_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
12 (R/NW)	ICMPGFRM	Rx ICMP (Good) Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.ICMPGFRM bit is set when the EMAC_RXICMP_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
11 (R/NW)	TCPERRFRM	Rx TCP Error Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.TCPERRFRM bit is set when the EMAC_RXTCP_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
10 (R/NW)	TCPGFRM	Rx TCP (Good) Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.TCPGFRM bit is set when the EMAC_RXTCP_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
9 (R/NW)	UDPERRFRM	Rx IDP Error Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.UDPERRFRM bit is set when the EMAC_RXUDP_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
8 (R/NW)	UDPGFRM	Rx UDP (Good) Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.UDPGFRM bit is set when the EMAC_RXUDP_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
7 (R/NW)	V6NOPAYFRM	Rx IPv6 No Payload Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.V6NOPAYFRM bit is set when the EMAC_RXIPV6_NOPAY_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
6 (R/NW)	V6HDERRFRM	Rx IPv6 Header Error Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.V6HDERRFRM bit is set when the EMAC_RXIPV6_HDR_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.

Table 23-70: EMAC_IPC_RXINT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/NW)	V6GFRM	Rx IPv6 (Good) Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.V6GFRM bit is set when the EMAC_RXIPV6_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
4 (R/NW)	V4UDSBLFRM	Rx IPv4 UDS Disable Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4UDSBLFRM bit is set when the EMAC_RXIPV4_UDSBL_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
3 (R/NW)	V4FRAGFRM	Rx IPv4 Fragmented Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4FRAGFRM bit is set when the EMAC_RXIPV4_FRAG_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
2 (R/NW)	V4NOPAYFRM	Rx IPv4 No Payload Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4NOPAYFRM bit is set when the EMAC_RXIPV4_NOPAY_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
1 (R/NW)	V4HDERRFRM	Rx IPv4 Header Error Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4HDERRFRM bit is set when the EMAC_RXIPV4_HDR_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
0 (R/NW)	V4GFRM	Rx IPv4 (Good) Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4GFRM bit is set when the EMAC_RXIPV4_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.

Interrupt Status Register

The EMAC_ISTAT register indicates EMAC interrupt status.

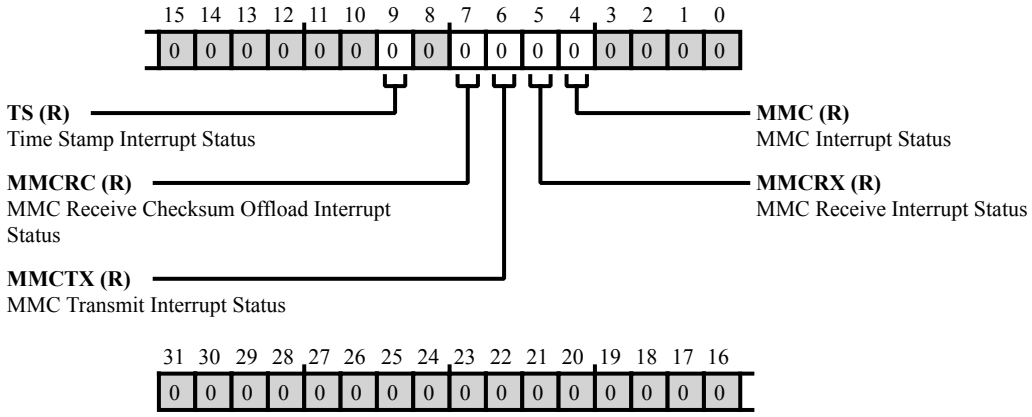

Figure 23-41: EMAC_ISTAT Register Diagram

Table 23-71: EMAC_ISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/NW)	TS	Time Stamp Interrupt Status. The EMAC_ISTAT.TS bit is set when: • The system time value equals or exceeds the value specified in the Time Stamp Target Time Seconds and nanoseconds registers (EMAC_TM_TGTM, EMAC_TM_NTGTM), or • There is an overflow in the EMAC_TM_SEC register, or • When the EMAC_TM_STMPSTAT.ATSTS bit is asserted. The EMAC_ISTAT.TS bit is cleared on reading the byte 0 of the EMAC_TM_STMPSTAT register. Otherwise, when default time-stamping is enabled, this bit, when set, indicates that the system time value equals or exceeds the value specified in the EMAC_TM_TGTM and EMAC_TM_NTGTM registers. In this mode, this bit is cleared after the completion of the read of the EMAC_ISTAT register. In all other modes, this bit is reserved.
7 (R/NW)	MMCRC	MMC Receive Checksum Offload Interrupt Status. The EMAC_ISTAT.MMCRC bit is set high whenever an interrupt is generated in the EMAC_IPC_RXINT. This bit is cleared when all the bits in this interrupt register are cleared.
6 (R/NW)	MMCTX	MMC Transmit Interrupt Status. The EMAC_ISTAT.MMCTX bit is set high whenever an interrupt is generated in the EMAC_MMC_TXINT register. This bit is cleared when all the bits in this interrupt register are cleared.
5	MMCRX	MMC Receive Interrupt Status.

Table 23-71: EMAC_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(R/NW)		The EMAC_ISTAT.MMCRX bit is set high whenever an interrupt is generated in the EMAC_MMC_RXINT register. This bit is cleared when all the bits in this interrupt register are cleared.
4 (R/NW)	MMC	MMC Interrupt Status. The EMAC_ISTAT.MMC bit is set high whenever any of EMAC_ISTAT bits [7:5] is set (=1) and is cleared only when all of these bits are cleared (=0).

MAC Configuration Register

The EMAC_MACCFG register configures MAC features.

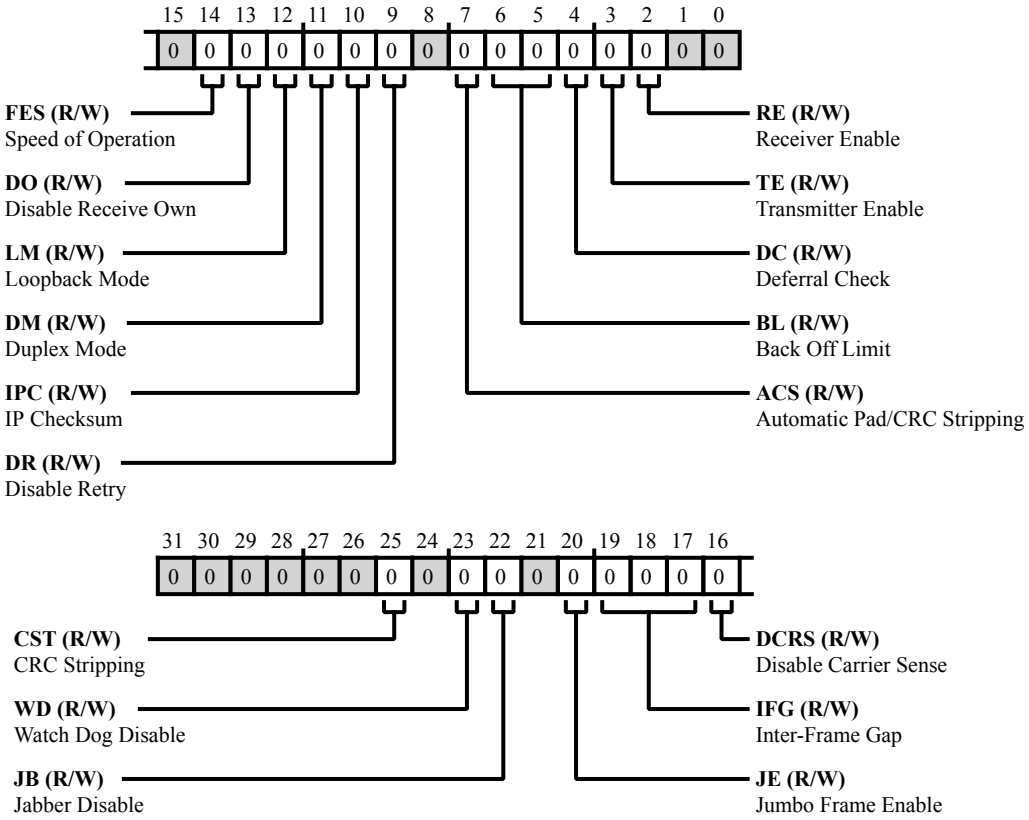

Figure 23-42: EMAC_MACCFG Register Diagram

Table 23-72: EMAC_MACCFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/W)	CST	CRC Stripping. The EMAC_MACCFG.CST bit, when set, directs the MAC to strip the last 4 bytes (FCS) of all frames of Ether type (Type field of frame greater than 0x0600) and drop these bytes before forwarding the frame to the application.
23 (R/W)	WD	Watch Dog Disable. The EMAC_MACCFG.WD bit, when set, disables the watchdog timer on the receiver, and can receive frames of up to 16,384 bytes. When this bit is reset, the MAC allows no more than 2,048 bytes (10,240 if EMAC_MACCFG.JE is set high) of the frame being received and cuts off any bytes received after that.
22 (R/W)	JB	Jabber Disable. The EMAC_MACCFG.JB bit, when set, disables the jabber timer on the transmitter, and can transfer frames of up to 16,384 bytes. When this bit is reset, the MAC cuts off the

Table 23-72: EMAC_MACCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		transmitter if the application sends out more than 2,048 bytes of data (10,240 if EMAC_MACCFG.JE is set high) during transmission.	
20 (R/W)	JE	Jumbo Frame Enable. The EMAC_MACCFG.JE bit, when set, directs the MAC to allow Jumbo frames of 9,018 bytes (9,022 bytes for VLAN tagged frames).	
19:17 (R/W)	IFG	Inter-Frame Gap. The EMAC_MACCFG.IFG bits control the minimum inter-frame gap between frames during transmission. Note that in Half-Duplex mode, the minimum gap can be configured for 64 bit times (EMAC_MACCFG.IFG =100) only. Lower values are not considered.	
		0	96 bit times
		1	88 bit times
		2	80 bit times
		3	72 bit times
		4	64 bit times
		5	56 bit times
		6	48 bit times
		7	40 bit times
16 (R/W)	DCRS	Disable Carrier Sense. The EMAC_MACCFG.DCRS bit, when set, makes the MAC transmitter ignore the CRS signal during frame transmission in Half-Duplex mode. This request results in no errors generated due to Loss of Carrier or No Carrier during such transmission. When this bit is low, the MAC transmitter generates such errors due to Carrier Sense and will even abort the transmissions.	
14 (R/W)	FES	Speed of Operation. The EMAC_MACCFG.FES bit indicates the Ethernet speed as 10 Mbps (bit =0) or 100 Mbps (bit =1).	
13 (R/W)	DO	Disable Receive Own. The EMAC_MACCFG.DO bit, when set, disables MAC reception of frames when MAC is transmitting in Half-Duplex mode. When this bit is reset, the MAC receives all packets that are given by the PHY while transmitting. This bit is not applicable if the MAC is operating in Full-Duplex mode.	
12 (R/W)	LM	Loopback Mode. The EMAC_MACCFG.LM bit, when set, directs the MAC to operate in internal loop back mode. (The media independent interface pins are not driven or sampled.)	
11	DM	Duplex Mode.	

Table 23-72: EMAC_MACCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
(R/W)		The EMAC_MACCFG.DM bit, when set, directs the MAC to operate in a Full-Duplex mode where it can transmit and receive simultaneously.	
10 (R/W)	IPC	IP Checksum. The EMAC_MACCFG.IPC bit, when set, directs the MAC to calculate the 16-bit one's complement of the one's complement sum of all received Ethernet frame payloads. It also checks whether the IPv4 Header checksum (assumed to be bytes 25-26 or 29-30 (VLAN-tagged) of the received Ethernet frame) is correct for the received frame and gives the status in the receive status word. The EMAC_MACCFG.IPC bit, when set, enables IPv4 checksum checking for received frame payloads' TCP/UDP/ICMP headers. When this bit is reset, the Checksum Offload Engine function in the receiver is disabled and the corresponding PCE and IP HCE status bits are always cleared.	
9 (R/W)	DR	Disable Retry. The EMAC_MACCFG.DR bit, when set, directs the MAC to attempt only 1 transmission. When a collision occurs on the media independent interface, the MAC ignores the current frame transmission and reports a Frame Abort with excessive collision error in the transmit frame status. When the EMAC_MACCFG.DR bit is reset, the MAC attempts retries based on the settings of BL. This bit is applicable only to Half-Duplex mode.	
		0	Retry enabled
		1	Retry disabled
7 (R/W)	ACS	Automatic Pad/CRC Stripping. The EMAC_MACCFG.ACS bit, when set, directs the MAC to strip the Pad/FCS field on incoming frames only if the length fields value is less than or equal to 1,500 bytes. All received frames with length field greater than or equal to 1,501 bytes are passed to the application without stripping the Pad/FCS field. When the EMAC_MACCFG.ACS bit is reset, the MAC passes all incoming frames to the Host unmodified.	
6:5 (R/W)	BL	Back Off Limit. The EMAC_MACCFG.BL bit selects the back-off limit, determining the random integer number (r) of slot time delays (512 bit times for 10/100 Mbps) the MAC waits before rescheduling a transmission attempt during retries after a collision. This bit is applicable only to Half-Duplex mode.The random integer r takes the value in the range: 0 less-than-equal-to r less-than 2^k Where k is the minimum of n (number of transmission attempts) or a limit value.	
		0	k = min (n, 10)
		1	k = min (n, 8)
		2	k = min (n, 4)
		3	k = min (n, 1)
4	DC	Deferral Check.	

Table 23-72: EMAC_MACCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(R/W)		The EMAC_MACCFG.DC bit, when set, enables the deferral check function in the MAC. The MAC issues a Frame Abort status, along with the excessive deferral error bit set in the transmit frame status when the transmit state machine is deferred for more than 24,288 bit times in 10/100-Mbps mode. If the Jumbo frame mode is enabled in 10/100-Mbps mode, the threshold for deferral is 155,680 bits times. Deferral begins when the transmitter is ready to transmit, but is prevented because of an active CRS (carrier sense) signal. Defer time is not cumulative. If the transmitter defers for 10,000 bit times, then transmits, collides, backs off, and then has to defer again after completion of back-off, the deferral timer resets to 0 and restarts. When the EMAC_MACCFG.DC bit is reset, the deferral check function is disabled and the MAC defers until the CRS signal goes inactive. This bit is applicable only in Half-Duplex mode.
3 (R/W)	TE	Transmitter Enable. The EMAC_MACCFG.TE bit, when set, enables the transmit state machine of the MAC for transmission. When this bit is reset, the MAC transmit state machine is disabled after the completion of the transmission of the current frame, and will not transmit any further frames.
2 (R/W)	RE	Receiver Enable. The EMAC_MACCFG.RE bit, when set, enables the receiver state machine of the MAC for receiving frames. When this bit is reset, the MAC receive state machine is disabled after the completion of the reception of the current frame, and does not receive any further frames..

MAC Rx Frame Filter Register

The EMAC_MACFRMFILT register controls receive frame filter features.

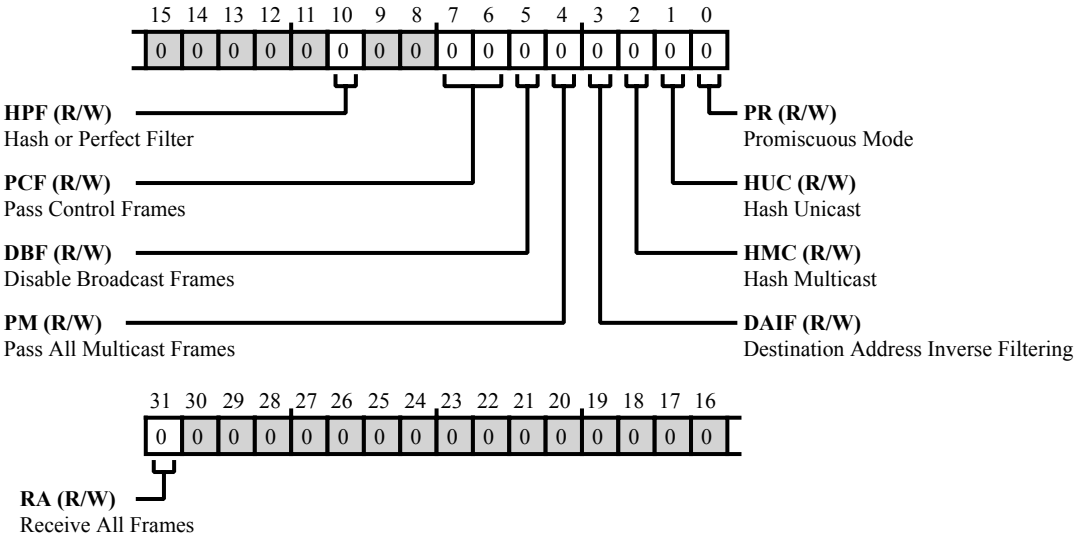

Figure 23-43: EMAC_MACFRMFILT Register Diagram

Table 23-73: EMAC_MACFRMFILT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	RA	Receive All Frames. The EMAC_MACFRMFILT.RA bit, when set, directs the MAC Receiver module to pass to the Application all frames received irrespective of whether they pass the address filter. The result of the DA filtering is updated (pass or fail) in the corresponding bits in the Receive Status Word. When this bit is reset, the Receiver module passes to the Application only those frames that pass the DA address filter.
10 (R/W)	HPF	Hash or Perfect Filter. The EMAC_MACFRMFILT.HPF bit. when set, configures the address filter to pass a frame if it matches either the perfect filtering or the hash filtering as set by EMAC_MACFRMFILT.HMC or EMAC_MACFRMFILT.HUC bits. When EMAC_MACFRMFILT.HPF is low and either the EMAC_MACFRMFILT.HUC bit or EMAC_MACFRMFILT.HMC bit is set, the frame is passed only if it matches the Hash filter.

Table 23-73: EMAC_MACFRMFILT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/W)	PCF	Pass Control Frames. The EMAC_MACFRMFILT.PCF bits control the forwarding of all control frames (including unicast and multicast PAUSE frames). Note that the processing of PAUSE control frames depends only on the value of the EMAC_FLOWCTL.RFE bit.	
		0	Pass no control frames All control frames are filtered from reaching the application.
		1	Pass no PAUSE frames All control frames are passed to the application (even if the fail the address filter), except for PAUSE frames.
		2	Pass all control frames All control frames are passed to the application (even if the fail the address filter).
		3	Pass address filtered control frames All control frames that pass the address filter are passed to the application.
5 (R/W)	DBF	Disable Broadcast Frames. The EMAC_MACFRMFILT.DBF bit, when set, directs the AFM module to filter all incoming broadcast frames. When this bit is reset, the AFM module passes all received broadcast frames.	
		0	AFM module passes all received broadcast frames
		1	AFM module filters all incoming broadcast frames
4 (R/W)	PM	Pass All Multicast Frames. The EMAC_MACFRMFILT.PM bit, when set, indicates that all received frames with a multicast destination address (first bit in the destination address field is =1) are passed. When this bit is reset, filtering of multicast frame depends on EMAC_MACFRMFILT.HMC bit.	
3 (R/W)	DAIF	Destination Address Inverse Filtering. The EMAC_MACFRMFILT.DAIF bit, when set, directs the Address Check block to operate in inverse filtering mode for the DA address comparison for both unicast and multicast frames. When this bit is reset, normal filtering of frames is performed.	
2 (R/W)	HMC	Hash Multicast. The EMAC_MACFRMFILT.HMC bit, when set, directs the EMAC to perform destination address filtering of received multicast frames according to the hash table. When this bit is reset, the MAC performs a perfect destination address filtering for multicast frames, that is, the MAC compares the DA field with the values programmed in the EMAC_ADDR0_HI and EMAC_ADDR0_LO address registers.	
1 (R/W)	HUC	Hash Unicast. The EMAC_MACFRMFILT.HUC bit, when set, directs the EMAC to perform destination address filtering of unicast frames according to the hash table. When this bit is	

Table 23-73: EMAC_MACFRMFILT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		reset, the MAC performs a perfect destination address filtering for unicast frames, that is, it compares the DA field with the values programmed in the EMAC_ADDR0_HI and EMAC_ADDR0_LO address registers.
0 (R/W)	PR	Promiscuous Mode. The EMAC_MACFRMFILT.PR bit, when set, directs the Address Filter module to pass all incoming frames regardless of its destination or source address. The DA Filter Fails status bits of the Receive Status Word is always cleared when EMAC_MACFRMFILT.PR is set.

MMC Control Register

The EMAC_MMC_CTL register selects the MMC operating mode.

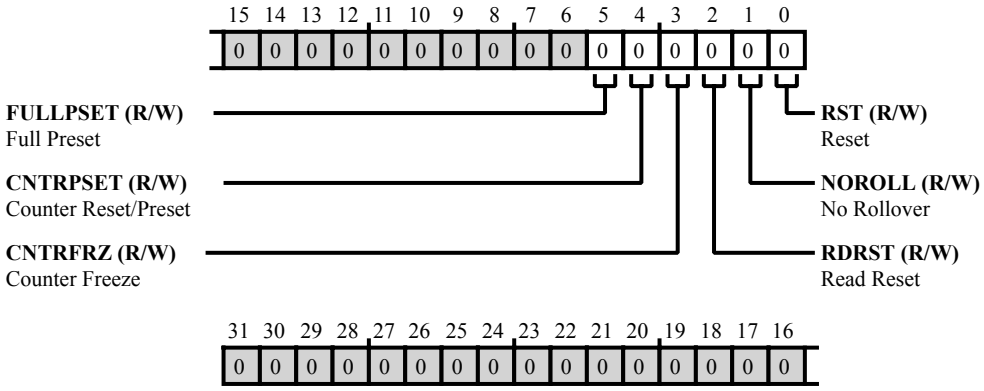

Figure 23-44: EMAC_MMC_CTL Register Diagram

Table 23-74: EMAC_MMC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W)	FULLPSET	Full Preset. The EMAC_MMC_CTL.FULLPSET bit, when =0 (and EMAC_MMC_CTL.CNTRPSET =1), presets all MMC counters to almost-half value. All octet counters get preset to 0x7FFF_F800 (half - 2KBytes) and all frame-counters gets preset to 0x7FFF_FFF0 (half - 16). When EMAC_MMC_CTL.FULLPSET =1 (and EMAC_MMC_CTL.CNTRPSET =1), all MMC counters get preset to almost-full value. All octet counters get preset to 0xFFFF_F800 (full - 2KBytes) and all frame-counters gets preset to 0xFFFF_FFF0 (full - 16). For 16-bit counters, the almost-half preset values are 0x7800 and 0x7FF0 for the respective octet and frame counters. Similarly, the almost-full preset values for the 16-bit counters are 0xF800 and 0xFFF0.
4 (R/W)	CNTRPSET	Counter Reset/Preset. The EMAC_MMC_CTL.CNTRPSET bit, when set, initializes all counters or presets counters to almost full or almost half as per EMAC_MMC_CTL.FULLPSET. The EMAC_MMC_CTL.CNTRPSET bit is cleared automatically after 1 clock cycle. This bit along with bit5 is useful for debugging and testing the assertion of interrupts because of MMC counter becoming half-full or full.
3 (R/W)	CNTRFRZ	Counter Freeze. The EMAC_MMC_CTL.CNTRFRZ bit, when set, freezes all the MMC counters to their current value. None of the MMC counters are updated due to any transmitted or received frame, until this bit is reset to 0. If any MMC counter is read with the EMAC_MMC_CTL.RDRST bit set, then that counter is also cleared in this mode.
2 (R/W)	RDRST	Read Reset.

Table 23-74: EMAC_MMC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		The EMAC_MMC_CTL.RDRST bit, when set, resets the MMC counters to zero after Read (self-clearing after reset). The counters are cleared when the least significant byte lane (bits[7:0]) is read.
1 (R/W)	NOROLL	No Rollover. The EMAC_MMC_CTL.NOROLL bit, when set, prevents counter rolls over to 0 after reaching max.
0 (R/W)	RST	Reset. The EMAC_MMC_CTL.RST bit, when set, resets all counters. This bit is cleared automatically after 1 clock cycle.

MMC Rx Interrupt Mask Register

The EMAC_MMC_RXIMSK register enables (unmasks) MMC receive interrupts.

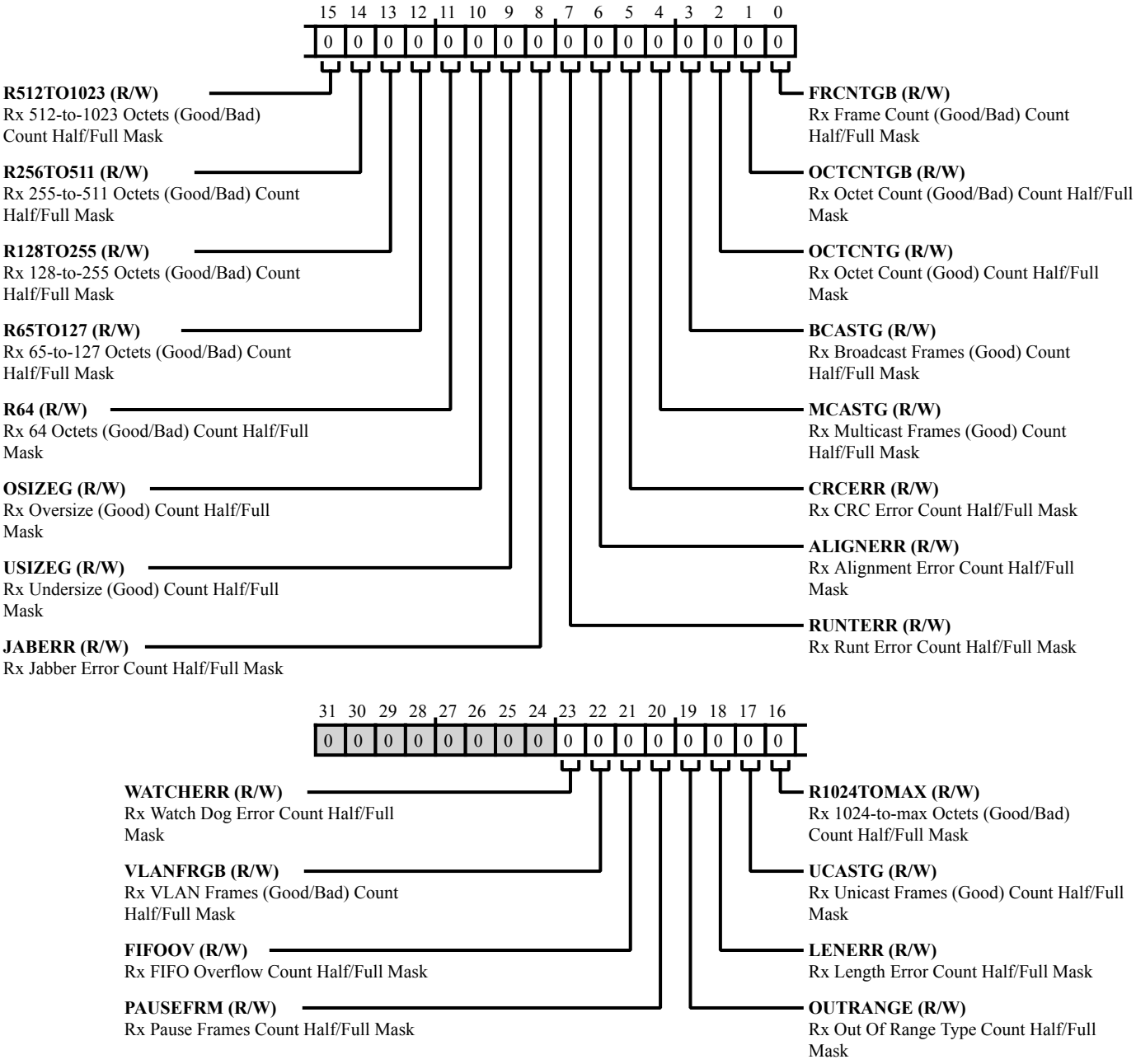

Figure 23-45: EMAC_MMC_RXIMSK Register Diagram

Table 23-75: EMAC_MMC_RXIMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23 (R/W)	WATCHERR	Rx Watch Dog Error Count Half/Full Mask. The EMAC_MMC_RXIMSK.WATCHERR bit, when set, masks the interrupt when EMAC_RXWDOG_ERR counter reaches full or half.
22 (R/W)	VLANFRGB	Rx VLAN Frames (Good/Bad) Count Half/Full Mask. The EMAC_MMC_RXIMSK.VLANFRGB bit, when set, masks the interrupt when EMAC_RXVLANFRM_GB counter reaches full or half.
21 (R/W)	FIFOOV	Rx FIFO Overflow Count Half/Full Mask. The EMAC_MMC_RXIMSK.FIFOOV bit, when set, masks the interrupt when EMAC_RXFIFO_OVF counter reaches full or half.
20 (R/W)	PAUSEFRM	Rx Pause Frames Count Half/Full Mask. The EMAC_MMC_RXIMSK.PAUSEFRM bit, when set, masks the interrupt when EMAC_RXPAUSEFRM counter reaches full or half.
19 (R/W)	OUTRANGE	Rx Out Of Range Type Count Half/Full Mask. The EMAC_MMC_RXIMSK.OUTRANGE bit, when set, masks the interrupt when EMAC_RXOORTYPE counter reaches full or half.
18 (R/W)	LENERR	Rx Length Error Count Half/Full Mask. The EMAC_MMC_RXIMSK.LENERR bit, when set, masks the interrupt when EMAC_RXLEN_ERR counter reaches full or half.
17 (R/W)	UCASTG	Rx Unicast Frames (Good) Count Half/Full Mask. The EMAC_MMC_RXIMSK.UCASTG bit, when set, masks the interrupt when EMAC_RXUCASTFRM_G counter reaches full or half.
16 (R/W)	R1024TOMAX	Rx 1024-to-max Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_RXIMSK.R1024TOMAX bit, when set, masks the interrupt when EMAC_RX1024TOMAX_GB counter reaches full or half.
15 (R/W)	R512TO1023	Rx 512-to-1023 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_RXIMSK.R512TO1023 bit, when set, masks the interrupt when EMAC_RX512TO1023_GB counter reaches full or half.
14 (R/W)	R256TO511	Rx 255-to-511 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_RXIMSK.R256TO511 bit, when set, masks the interrupt when EMAC_RX256TO511_GB counter reaches full or half.
13 (R/W)	R128TO255	Rx 128-to-255 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_RXIMSK.R128TO255 bit, when set, masks the interrupt when EMAC_RX128TO255_GB counter reaches full or half.
12 (R/W)	R65TO127	Rx 65-to-127 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_RXIMSK.R65TO127 bit, when set, masks the interrupt when EMAC_RX65TO127_GB counter reaches full or half.

Table 23-75: EMAC_MMC_RXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/W)	R64	Rx 64 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_RXIMSK.R64 bit, when set, masks the interrupt when EMAC_RX64_GB counter reaches full or half.
10 (R/W)	OSIZEG	Rx Oversize (Good) Count Half/Full Mask. The EMAC_MMC_RXIMSK.OSIZEG bit, when set, masks the interrupt when EMAC_RXOSIZE_G counter reaches full or half.
9 (R/W)	USIZEG	Rx Undersize (Good) Count Half/Full Mask. The EMAC_MMC_RXIMSK.USIZEG bit, when set, masks the interrupt when EMAC_RXUSIZE_G counter reaches full or half.
8 (R/W)	JABERR	Rx Jabber Error Count Half/Full Mask. The EMAC_MMC_RXIMSK.JABERR bit, when set, masks the interrupt when EMAC_RXJAB_ERR counter reaches full or half.
7 (R/W)	RUNTERR	Rx Runt Error Count Half/Full Mask. The EMAC_MMC_RXIMSK.RUNTERR bit, when set, masks the interrupt when EMAC_RXRUNT_ERR counter reaches full or half.
6 (R/W)	ALIGNERR	Rx Alignment Error Count Half/Full Mask. The EMAC_MMC_RXIMSK.ALIGNERR bit, when set, masks the interrupt when EMAC_RXALIGN_ERR counter reaches full or half.
5 (R/W)	CRCERR	Rx CRC Error Count Half/Full Mask. The EMAC_MMC_RXIMSK.CRCERR bit, when set, masks the interrupt when EMAC_RXCRC_ERR counter reaches full or half.
4 (R/W)	MCASTG	Rx Multicast Frames (Good) Count Half/Full Mask. The EMAC_MMC_RXIMSK.MCASTG bit, when set, masks the interrupt when EMAC_RXMCASTFRM_G counter reaches full or half.
3 (R/W)	BCASTG	Rx Broadcast Frames (Good) Count Half/Full Mask. The EMAC_MMC_RXIMSK.BCASTG bit, when set, masks the interrupt when EMAC_RXBCASTFRM_G counter reaches full or half.
2 (R/W)	OCTCNTG	Rx Octet Count (Good) Count Half/Full Mask. The EMAC_MMC_RXIMSK.OCTCNTG bit, when set, masks the interrupt when EMAC_RXOCTCNT_G counter reaches full or half.
1 (R/W)	OCTCNTGB	Rx Octet Count (Good/Bad) Count Half/Full Mask. The EMAC_MMC_RXIMSK.OCTCNTGB bit, when set, masks the interrupt when EMAC_RXOCTCNT_GB counter reaches half or full.
0 (R/W)	FRCNTGB	Rx Frame Count (Good/Bad) Count Half/Full Mask. The EMAC_MMC_RXIMSK.FRCNTGB bit, when set, masks the interrupt when EMAC_RXFRMCNT_GB counter reaches half or full.

MMC Rx Interrupt Register

The EMAC_MMC_RXINT register indicates status of MMC receive interrupts.

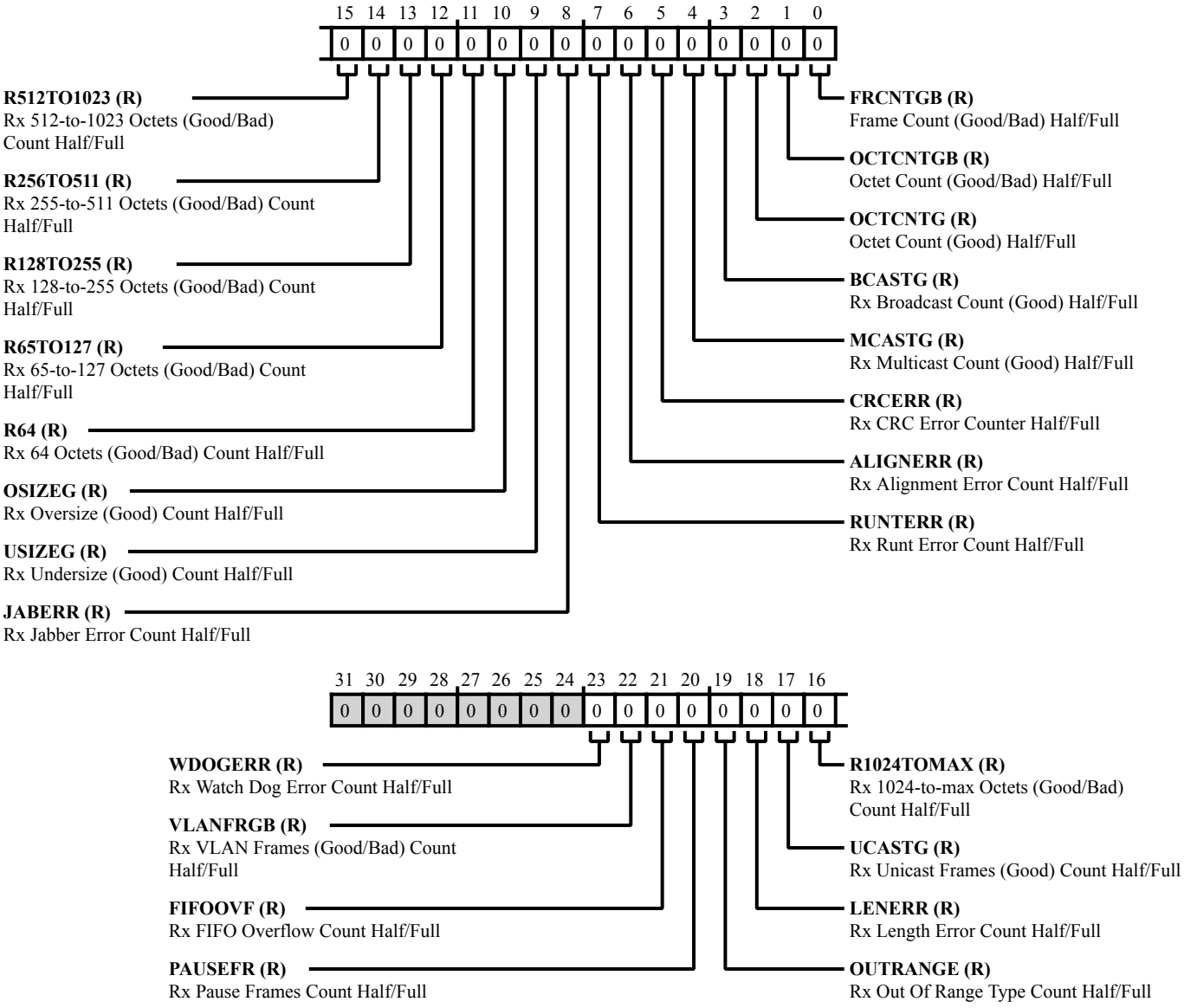

Figure 23-46: EMAC_MMC_RXINT Register Diagram

Table 23-76: EMAC_MMC_RXINT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23 (R/NW)	WDOGERR	Rx Watch Dog Error Count Half/Full. The EMAC_MMC_RXINT.WDOGERR bit is set when the EMAC_RXWDOG_ERR counter reaches full or half.

Table 23-76: EMAC_MMC_RXINT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
22 (R/NW)	VLANFRGB	Rx VLAN Frames (Good/Bad) Count Half/Full. The EMAC_MMC_RXINT.VLANFRGB bit is set when EMAC_RXVLANFRM_GB counter reaches full or half.
21 (R/NW)	FIFOOVF	Rx FIFO Overflow Count Half/Full. The EMAC_MMC_RXINT.FIFOOVF bit is set when EMAC_RXFIFO_OVF counter reaches full or half.
20 (R/NW)	PAUSEFR	Rx Pause Frames Count Half/Full. The EMAC_MMC_RXINT.PAUSEFR bit is set when EMAC_RXPAUSEFRM counter reaches full or half.
19 (R/NW)	OUTRANGE	Rx Out Of Range Type Count Half/Full. The EMAC_MMC_RXINT.OUTRANGE bit is set when EMAC_RXOORTYPE counter reaches full or half.
18 (R/NW)	LENERR	Rx Length Error Count Half/Full. The EMAC_MMC_RXINT.LENERR bit is set when EMAC_RXLEN_ERR counter reaches full or half.
17 (R/NW)	UCASTG	Rx Unicast Frames (Good) Count Half/Full. The EMAC_MMC_RXINT.UCASTG bit is set when EMAC_RXUCASTFRM_G counter reaches full or half.
16 (R/NW)	R1024TOMAX	Rx 1024-to-max Octets (Good/Bad) Count Half/Full. The EMAC_MMC_RXINT.R1024TOMAX bit is set when EMAC_RX1024TOMAX_GBcounter reaches full or half.
15 (R/NW)	R512TO1023	Rx 512-to-1023 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_RXINT.R512TO1023 bit is set when EMAC_RX512TO1023_GB counter reaches full or half.
14 (R/NW)	R256TO511	Rx 255-to-511 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_RXINT.R256TO511 bit is set when EMAC_RX256TO511_GB counter reaches full or half.
13 (R/NW)	R128TO255	Rx 128-to-255 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_RXINT.R128TO255 bit is set when EMAC_RX128TO255_GB counter reaches full or half.
12 (R/NW)	R65TO127	Rx 65-to-127 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_RXINT.R65TO127 bit is set when EMAC_RX65TO127_GB counter reaches full or half.
11 (R/NW)	R64	Rx 64 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_RXINT.R64 bit is set when EMAC_RX64_GB counter reaches full or half.

Table 23-76: EMAC_MMC_RXINT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/NW)	OSIZEG	Rx Oversize (Good) Count Half/Full. The EMAC_MMC_RXINT.OSIZEG bit is set when EMAC_RXOSIZE_G counter reaches full or half.
9 (R/NW)	USIZEG	Rx Undersize (Good) Count Half/Full. The EMAC_MMC_RXINT.USIZEG bit is set when EMAC_RXUSIZE_G counter reaches full or half.
8 (R/NW)	JABERR	Rx Jabber Error Count Half/Full. The EMAC_MMC_RXINT.JABERR bit is set when EMAC_RXJAB_ERR counter reaches full or half.
7 (R/NW)	RUNTERR	Rx Runt Error Count Half/Full. The EMAC_MMC_RXINT.RUNTERR bit is set when EMAC_RXRUNT_ERR counter reaches full or half.
6 (R/NW)	ALIGNERR	Rx Alignment Error Count Half/Full. The EMAC_MMC_RXINT.ALIGNERR bit is set when EMAC_RXALIGN_ERR counter reaches full or half
5 (R/NW)	CRCERR	Rx CRC Error Counter Half/Full. The EMAC_MMC_RXINT.CRCERR bit is set when EMAC_RXCRC_ERR counter reaches full or half.
4 (R/NW)	MCASTG	Rx Multicast Count (Good) Half/Full. The EMAC_MMC_RXINT.MCASTG bit is set when EMAC_RXMCASTFRM_G counter reaches full or half.
3 (R/NW)	BCASTG	Rx Broadcast Count (Good) Half/Full. The EMAC_MMC_RXINT.BCASTG bit is set when EMAC_RXBCASTFRM_G counter reaches full or half.
2 (R/NW)	OCTCNTG	Octet Count (Good) Half/Full. The EMAC_MMC_RXINT.OCTCNTG bit is set when EMAC_RXOCTCNT_G counter reaches full or half.
1 (R/NW)	OCTCNTGB	Octet Count (Good/Bad) Half/Full. The EMAC_MMC_RXINT.OCTCNTGB bit is set when EMAC_RXOCTCNT_GB counter reaches half or full.
0 (R/NW)	FRCNTGB	Frame Count (Good/Bad) Half/Full. The EMAC_MMC_RXINT.FRCNTGB bit is set when EMAC_RXFRMCNT_GB counter reaches half or full.

MMC TX Interrupt Mask Register

The EMAC_MMC_TXIMSK register enables (unmasks) MMC transmit interrupts.

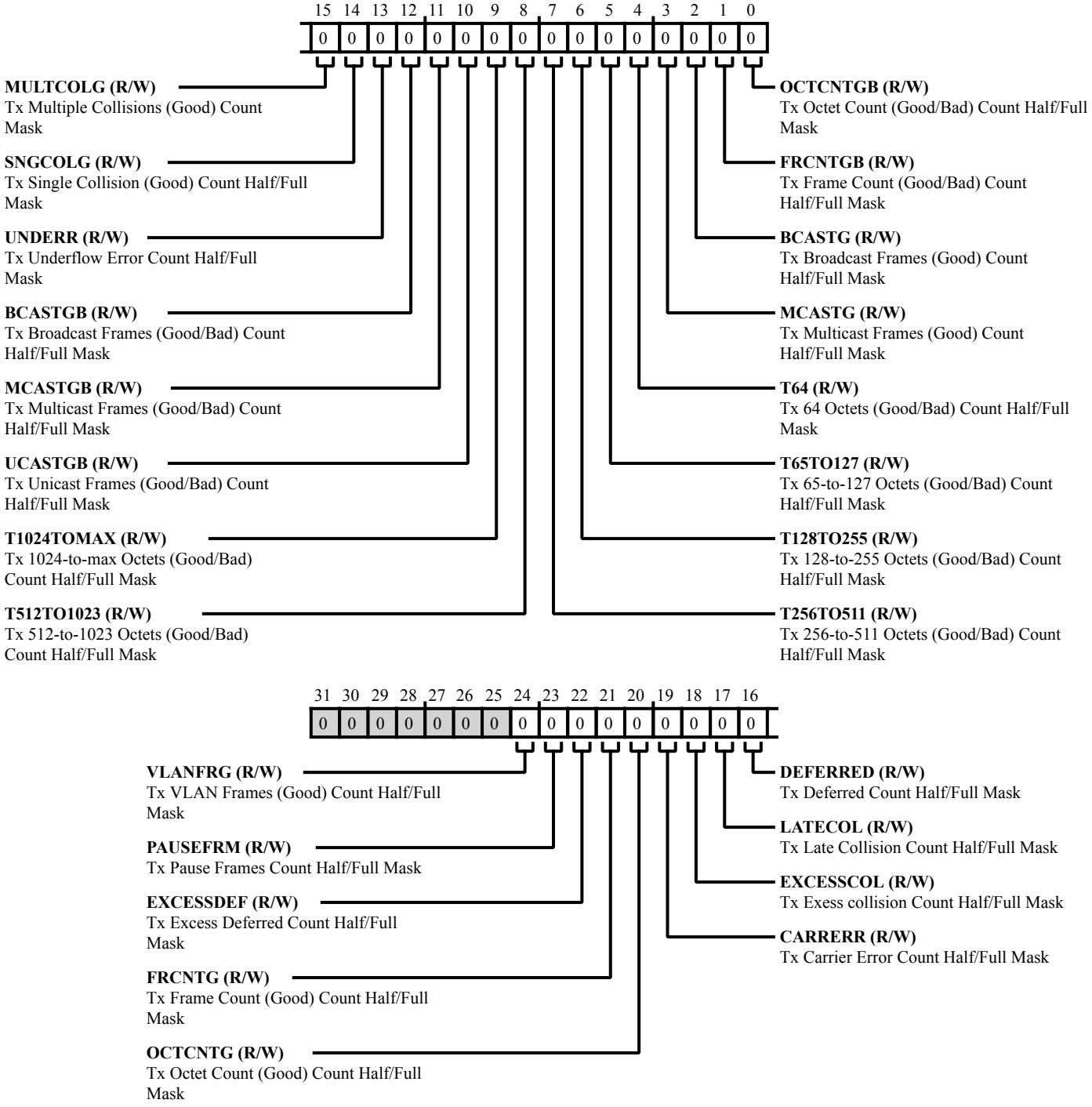

Figure 23-47: EMAC_MMC_TXIMSK Register Diagram

Table 23-77: EMAC_MMC_TXIMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24 (R/W)	VLANFRG	Tx VLAN Frames (Good) Count Half/Full Mask. The EMAC_MMC_TXIMSK.VLANFRG bit, when set, masks the interrupt when EMAC_TXVLANFRM_G counter reaches full or half.
23 (R/W)	PAUSEFRM	Tx Pause Frames Count Half/Full Mask. The EMAC_MMC_TXIMSK.PAUSEFRM bit, when set, masks the interrupt when EMAC_TXPAUSEFRM counter reaches full or half.
22 (R/W)	EXCESSDEF	Tx Excess Deferred Count Half/Full Mask. The EMAC_MMC_TXIMSK.EXCESSDEF bit, when set, masks the interrupt when EMAC_TXEXCESSDEF counter reaches full or half.
21 (R/W)	FRCNTG	Tx Frame Count (Good) Count Half/Full Mask. The EMAC_MMC_TXIMSK.FRCNTG bit, when set, masks the interrupt when EMAC_TXFRMCNT_G counter reaches full or half.
20 (R/W)	OCTCNTG	Tx Octet Count (Good) Count Half/Full Mask. The EMAC_MMC_TXIMSK.OCTCNTG bit, when set, masks the interrupt when EMAC_TXOCTCNT_G counter reaches full or half.
19 (R/W)	CARRERR	Tx Carrier Error Count Half/Full Mask. The EMAC_MMC_TXIMSK.CARRERR bit, when set, masks the interrupt when EMAC_TXCARR_ERR counter reaches full or half.
18 (R/W)	EXCESSCOL	Tx Exess collision Count Half/Full Mask. The EMAC_MMC_TXIMSK.EXCESSCOL bit, when set, masks the interrupt when EMAC_TXEXCESSCOL counter reaches full or half.
17 (R/W)	LATECOL	Tx Late Collision Count Half/Full Mask. The EMAC_MMC_TXIMSK.LATECOL bit, when set, masks the interrupt when EMAC_TXLATECOL counter reaches full or half.
16 (R/W)	DEFERRED	Tx Deferred Count Half/Full Mask. The EMAC_MMC_TXIMSK.DEFERRED bit, when set, masks the interrupt when EMAC_TXDEFERRED counter reaches full or half.
15 (R/W)	MULTCOLG	Tx Multiple Collisions (Good) Count Mask. The EMAC_MMC_TXIMSK.MULTCOLG bit, when set, masks the interrupt when EMAC_TXMULTCOL_G counter reaches full or half.
14 (R/W)	SNGCOLG	Tx Single Collision (Good) Count Half/Full Mask. The EMAC_MMC_TXIMSK.SNGCOLG bit, when set, masks the interrupt when EMAC_TXSNGCOL_G counter reaches full or half.
13 (R/W)	UNDERR	Tx Underflow Error Count Half/Full Mask. The EMAC_MMC_TXIMSK.UNDERR bit, when set, masks the interrupt when EMAC_TXUNDR_ERR counter reaches full or half.

Table 23-77: EMAC_MMC_TXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W)	BCASTGB	Tx Broadcast Frames (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.BCASTGB bit, when set, masks the interrupt when EMAC_TXBCASTFRM_GB counter reaches full or half.
11 (R/W)	MCASTGB	Tx Multicast Frames (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.MCASTGB bit, when set, masks the interrupt when EMAC_TXMCASTFRM_GB counter reaches full or half.
10 (R/W)	UCASTGB	Tx Unicast Frames (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.UCASTGB bit, when set, masks the interrupt when EMAC_TXUCASTFRM_GB counter reaches full or half.
9 (R/W)	T1024TOMAX	Tx 1024-to-max Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.T1024TOMAX bit, when set, masks the interrupt when EMAC_TX1024TOMAX_GB counter reaches full or half.
8 (R/W)	T512TO1023	Tx 512-to-1023 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.T512TO1023 bit, when set, masks the interrupt when EMAC_TX512TO1023_GB counter reaches full or half.
7 (R/W)	T256TO511	Tx 256-to-511 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.T256TO511 bit, when set, masks the interrupt when EMAC_TX256TO511_GB counter reaches full or half.
6 (R/W)	T128TO255	Tx 128-to-255 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.T128TO255 bit, when set, masks the interrupt when EMAC_TX128TO255_GB counter reaches full or half.
5 (R/W)	T65TO127	Tx 65-to-127 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.T65TO127 bit, when set, masks the interrupt when EMAC_TX65TO127_GB counter reaches full or half.
4 (R/W)	T64	Tx 64 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.T64 bit, when set, masks the interrupt when EMAC_TX64_GB counter reaches full or half.
3 (R/W)	MCASTG	Tx Multicast Frames (Good) Count Half/Full Mask. The EMAC_MMC_TXIMSK.MCASTG bit, when set, masks the interrupt when EMAC_TXMCASTFRM_G counter reaches full or half.
2 (R/W)	BCASTG	Tx Broadcast Frames (Good) Count Half/Full Mask. The EMAC_MMC_TXIMSK.BCASTG bit, when set, masks the interrupt when EMAC_TXBCASTFRM_G counter reaches full or half.
1 (R/W)	FRCNTGB	Tx Frame Count (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.FRCNTGB bit, when set, masks the interrupt when EMAC_TXFRMCNT_GB counter reaches full or half.

Table 23-77: EMAC_MMC_TXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	OCTCNTGB	Tx Octet Count (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_TXIMSK.OCTCNTGB` bit, when set, masks the interrupt when `EMAC_TXOCTCNT_GB` counter reaches full or half.

MMC Tx Interrupt Register

The EMAC_MMC_TXINT register indicates status of MMC transmit interrupts.

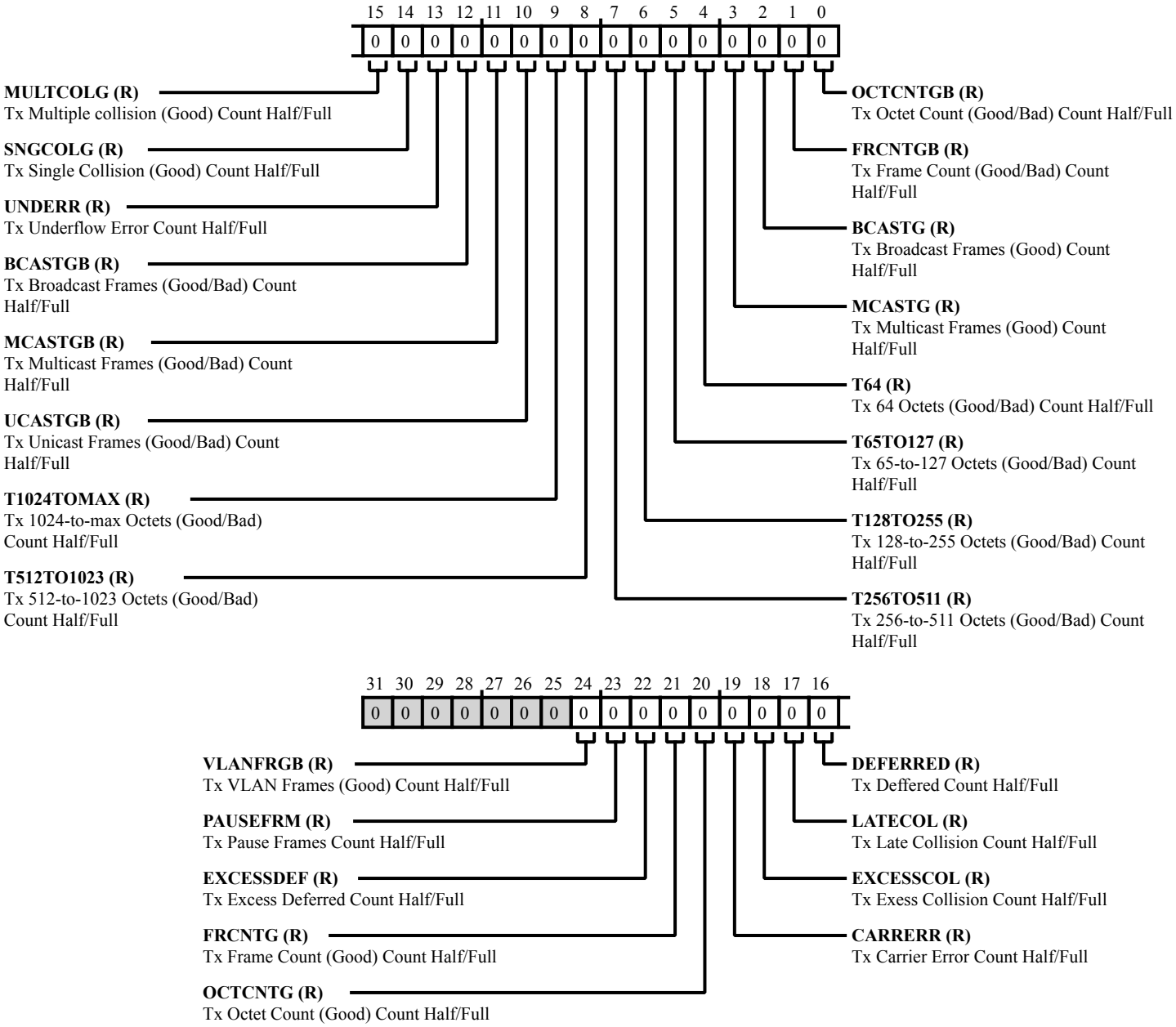

Figure 23-48: EMAC_MMC_TXINT Register Diagram

Table 23-78: EMAC_MMC_TXINT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24	VLANFRGB	Tx VLAN Frames (Good) Count Half/Full.

Table 23-78: EMAC_MMC_TXINT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(R/NW)		The EMAC_MMC_TXINT.VLANFRGB bit is set when EMAC_TXVLANFRM_G counter reaches full or half.
23 (R/NW)	PAUSEFRM	Tx Pause Frames Count Half/Full. The EMAC_MMC_TXINT.PAUSEFRM bit is set when EMAC_TXPAUSEFRM counter reaches full or half.
22 (R/NW)	EXCESSDEF	Tx Excess Deferred Count Half/Full. The EMAC_MMC_TXINT.EXCESSDEF bit is set when EMAC_TXEXCESSDEF counter reaches full or half.
21 (R/NW)	FRCNTG	Tx Frame Count (Good) Count Half/Full. The EMAC_MMC_TXINT.FRCNTG bit is set when EMAC_TXFRMCNT_G counter reaches full or half.
20 (R/NW)	OCTCNTG	Tx Octet Count (Good) Count Half/Full. The EMAC_MMC_TXINT.OCTCNTG bit is set when EMAC_TXOCTCNT_G counter reaches full or half.
19 (R/NW)	CARRERR	Tx Carrier Error Count Half/Full. The EMAC_MMC_TXINT.CARRERR bit is set when EMAC_TXCARR_ERR counter reaches full or half.
18 (R/NW)	EXCESSCOL	Tx Exess Collision Count Half/Full. The EMAC_MMC_TXINT.EXCESSCOL bit is set when EMAC_TXEXCESSCOL counter reaches full or half.
17 (R/NW)	LATECOL	Tx Late Collision Count Half/Full. The EMAC_MMC_TXINT.LATECOL bit is set when EMAC_TXLATECOL counter reaches full or half.
16 (R/NW)	DEFERRED	Tx Deffered Count Half/Full. The EMAC_MMC_TXINT.DEFERRED bit is set when EMAC_TXDEFERRED counter reaches full or half.
15 (R/NW)	MULTCOLG	Tx Multiple collision (Good) Count Half/Full. The EMAC_MMC_TXINT.MULTCOLG bit is set when EMAC_TXMULTCOL_G counter reaches full or half.
14 (R/NW)	SNGCOLG	Tx Single Collision (Good) Count Half/Full. The EMAC_MMC_TXINT.SNGCOLG bit is set when EMAC_TXSNGCOL_G counter reaches full or half.
13 (R/NW)	UNDERR	Tx Underflow Error Count Half/Full. The EMAC_MMC_TXINT.UNDERR bit is set when EMAC_TXUNDR_ERR counter reaches full or half.
12	BCASTGB	Tx Broadcast Frames (Good/Bad) Count Half/Full.

Table 23-78: EMAC_MMC_TXINT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(R/NW)		The EMAC_MMC_TXINT.BCASTGB bit is set when EMAC_TXBCASTFRM_GB counter reaches full or half.
11 (R/NW)	MCASTGB	Tx Multicast Frames (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.MCASTGB bit is set when EMAC_TXMCASTFRM_GB counter reaches full or half.
10 (R/NW)	UCASTGB	Tx Unicast Frames (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.UCASTGB bit is set when EMAC_TXUCASTFRM_GB counter reaches full or half.
9 (R/NW)	T1024TOMAX	Tx 1024-to-max Octets (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.T1024TOMAX bit is set when EMAC_TX1024TOMAX_GB counter reaches full or half.
8 (R/NW)	T512TO1023	Tx 512-to-1023 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.T512TO1023 bit is set when EMAC_TX512TO1023_GB counter reaches full or half.
7 (R/NW)	T256TO511	Tx 256-to-511 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.T256TO511 bit is set when EMAC_TX256TO511_GB counter reaches full or half.
6 (R/NW)	T128TO255	Tx 128-to-255 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.T128TO255 bit is set when EMAC_TX128TO255_GB counter reaches full or half.
5 (R/NW)	T65TO127	Tx 65-to-127 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.T65TO127 bit is set when EMAC_TX65TO127_GB counter reaches full or half.
4 (R/NW)	T64	Tx 64 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.T64 bit is set when EMAC_TX64_GB counter reaches full or half.
3 (R/NW)	MCASTG	Tx Multicast Frames (Good) Count Half/Full. The EMAC_MMC_TXINT.MCASTG bit is set when EMAC_TXMCASTFRM_G counter reaches full or half.
2 (R/NW)	BCASTG	Tx Broadcast Frames (Good) Count Half/Full. The EMAC_MMC_TXINT.BCASTG bit is set when EMAC_TXBCASTFRM_G counter reaches full or half.
1 (R/NW)	FRCNTGB	Tx Frame Count (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.FRCNTGB bit is set when EMAC_TXFRMCNT_GB counter reaches full or half.
0	OCTCNTGB	Tx Octet Count (Good/Bad) Count Half/Full.

Table 23-78: EMAC_MMC_TXINT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(R/NW)		The EMAC_MMC_TXINT.OCTCNTGB bit is set when EMAC_TXOCTCNT_GB counter reaches full or half.

Rx 1024- to Max-Byte Frames (Good/Bad) Register

The EMAC_RX1024TOMAX_GB register contains a count of the number of good and bad frames received with length between 1024 and maxsize (inclusive) bytes, exclusive of preamble.

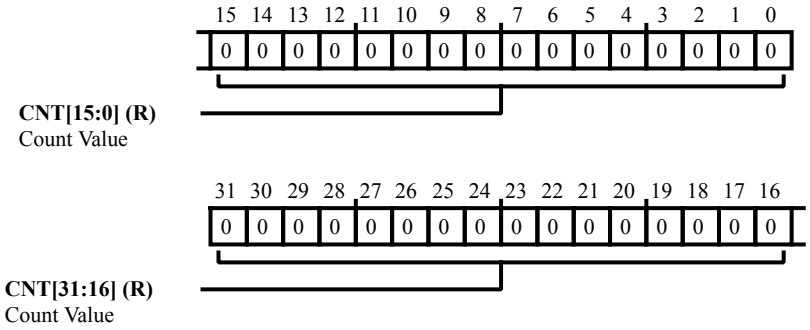

Figure 23-49: EMAC_RX1024TOMAX_GB Register Diagram

Table 23-79: EMAC_RX1024TOMAX_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx 128- to 255-Byte Frames (Good/Bad) Register

The EMAC_RX128TO255_GB register contains a count of the number of good and bad frames received with length between 128 and 255 (inclusive) bytes, exclusive of preamble.

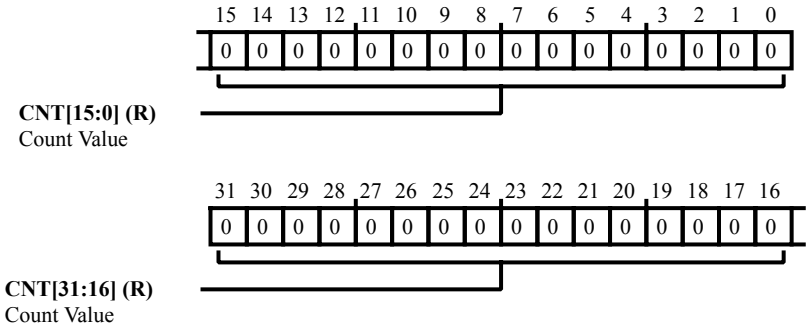

Figure 23-50: EMAC_RX128TO255_GB Register Diagram

Table 23-80: EMAC_RX128TO255_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx 256- to 511-Byte Frames (Good/Bad) Register

The EMAC_RX256TO511_GB register contains a count of the number of good and bad frames received with length between 256 and 511 (inclusive) bytes, exclusive of preamble.

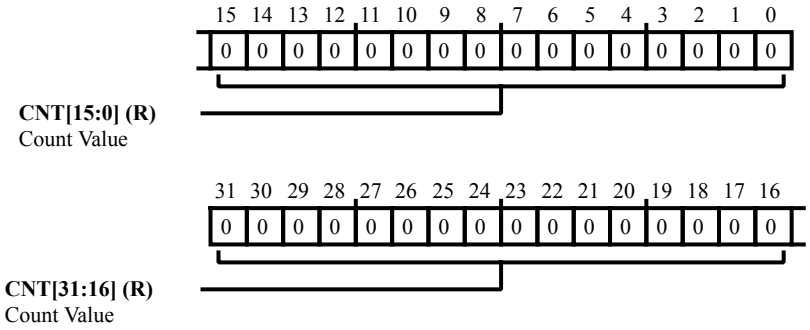

Figure 23-51: EMAC_RX256TO511_GB Register Diagram

Table 23-81: EMAC_RX256TO511_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx 512- to 1023-Byte Frames (Good/Bad) Register

The EMAC_RX512TO1023_GB register contains a count of the number of good and bad frames received with length between 512 and 1023 (inclusive) bytes, exclusive of preamble.

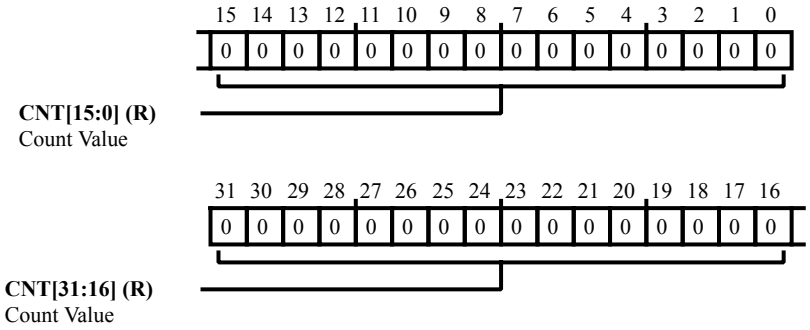

Figure 23-52: EMAC_RX512TO1023_GB Register Diagram

Table 23-82: EMAC_RX512TO1023_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx 64-Byte Frames (Good/Bad) Register

The EMAC_RX64_GB register contains a count of the number of good and bad frames received with length 64 bytes, exclusive of preamble.

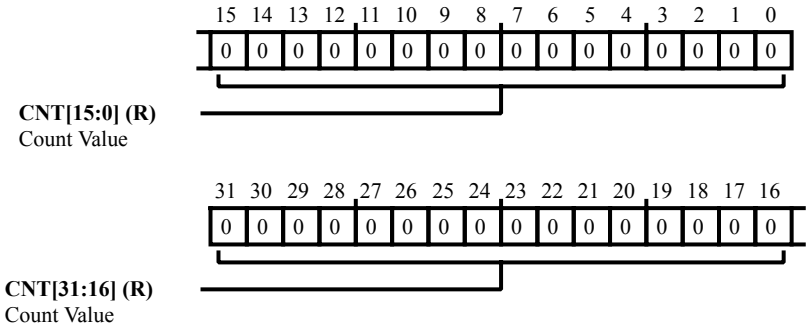

Figure 23-53: EMAC_RX64_GB Register Diagram

Table 23-83: EMAC_RX64_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx 65- to 127-Byte Frames (Good/Bad) Register

The EMAC_RX65TO127_GB register contains a count of the number of good and bad frames received with length between 65 and 127 (inclusive) bytes, exclusive of preamble.

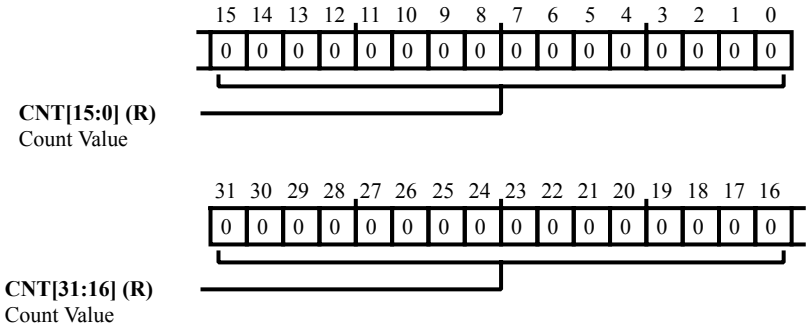

Figure 23-54: EMAC_RX65TO127_GB Register Diagram

Table 23-84: EMAC_RX65TO127_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx alignment Error Register

The EMAC_RXALIGN_ERR register contains a count of the number of frames received with alignment (dribble) error. Valid only in 10/100 mode.

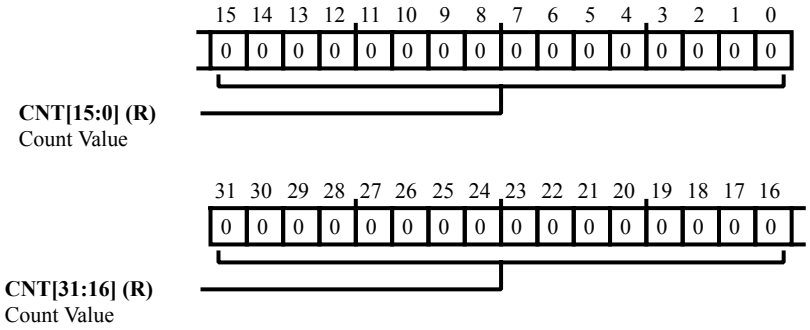

Figure 23-55: EMAC_RXALIGN_ERR Register Diagram

Table 23-85: EMAC_RXALIGN_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Broadcast Frames (Good) Register

The EMAC_RXBCASTFRM_G register contains a count of the number of good broadcast frames received.

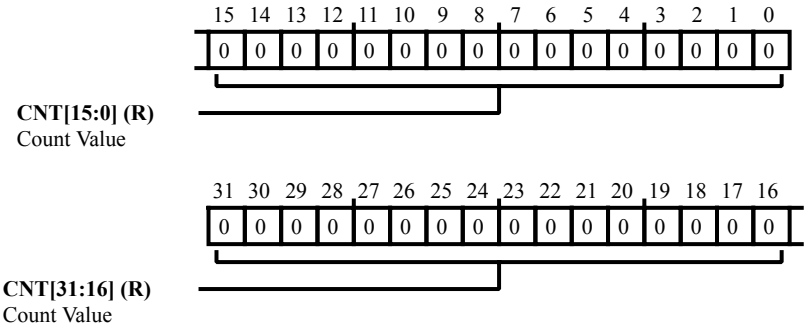

Figure 23-56: EMAC_RXBCASTFRM_G Register Diagram

Table 23-86: EMAC_RXBCASTFRM_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx CRC Error Register

The EMAC_RXCRC_ERR register contains a count of the number of frames received with CRC error.

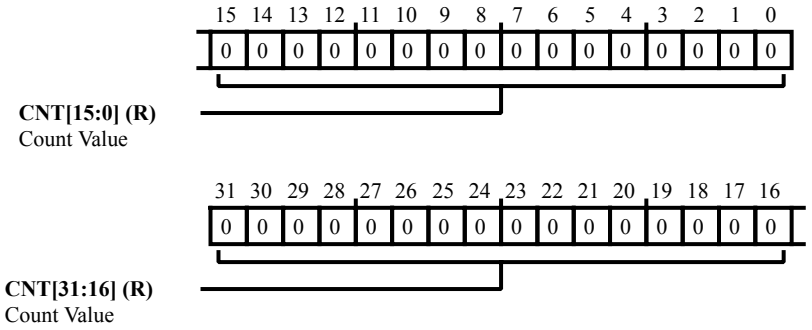

Figure 23-57: EMAC_RXCRC_ERR Register Diagram

Table 23-87: EMAC_RXCRC_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx FIFO Overflow Register

The EMAC_RXFIFO_OVF register contains a count of the number of missed received frames due to FIFO overflow.

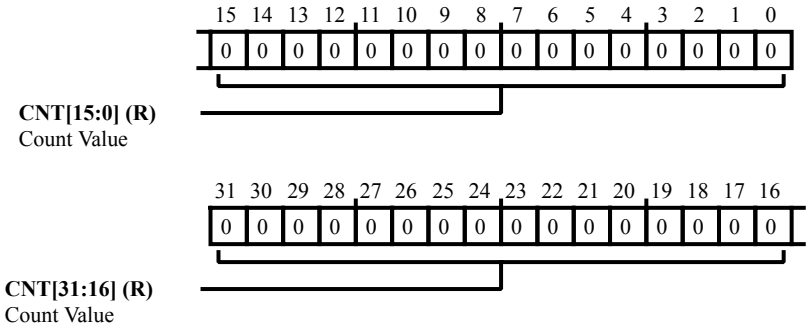

Figure 23-58: EMAC_RXFIFO_OVF Register Diagram

Table 23-88: EMAC_RXFIFO_OVF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Frame Count (Good/Bad) Register

The EMAC_RXFRMCNT_GB register contains a count of the number of good and bad frames received.

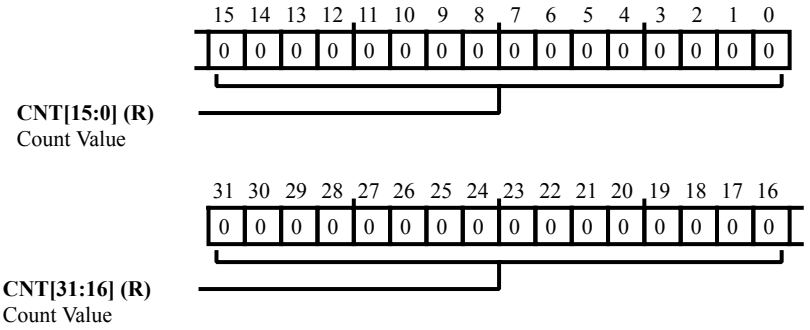

Figure 23-59: EMAC_RXFRMCNT_GB Register Diagram

Table 23-89: EMAC_RXFRMCNT_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx ICMP Error Frames Register

The EMAC_RXICMP_ERR_FRM register contains a count of the number of good IP datagrams whose ICMP payload has a checksum error.

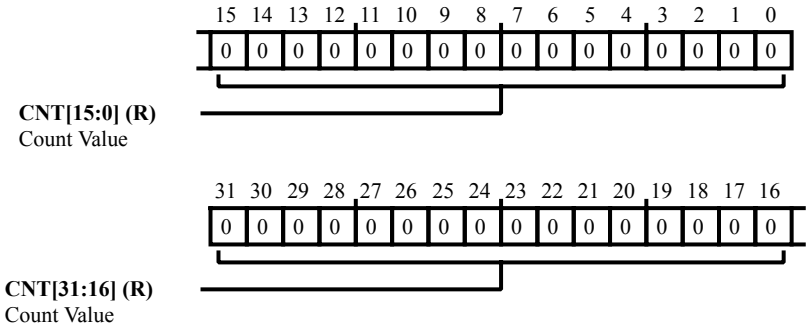

Figure 23-60: EMAC_RXICMP_ERR_FRM Register Diagram

Table 23-90: EMAC_RXICMP_ERR_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx ICMP Error Octets Register

The EMAC_RXICMP_ERR_OCT register contains a count of the number of bytes received in an ICMP segment with checksum errors.

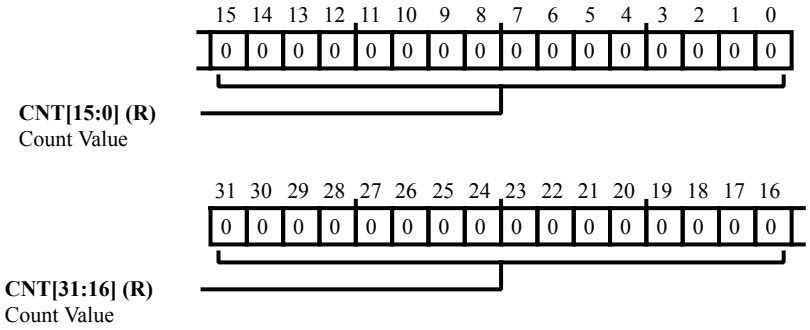

Figure 23-61: EMAC_RXICMP_ERR_OCT Register Diagram

Table 23-91: EMAC_RXICMP_ERR_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx ICMP Good Frames Register

The EMAC_RXICMP_GD_FRM register contains a count of the number of good IP datagrams with a good ICMP payload.

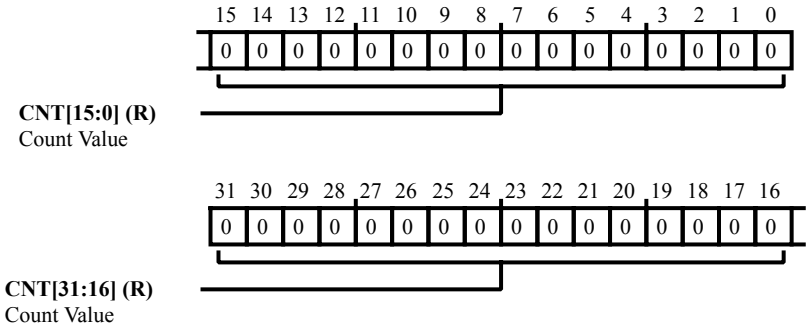

Figure 23-62: EMAC_RXICMP_GD_FRM Register Diagram

Table 23-92: EMAC_RXICMP_GD_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx ICMP Good Octets Register

The EMAC_RXICMP_GD_OCT register contains a count of the Number of bytes received in a good ICMP segment.

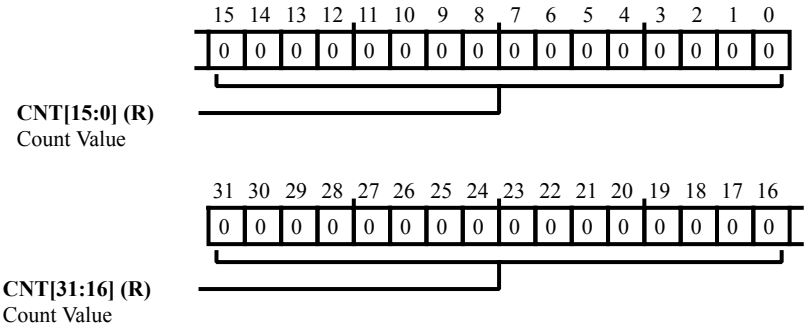

Figure 23-63: EMAC_RXICMP_GD_OCT Register Diagram

Table 23-93: EMAC_RXICMP_GD_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams Fragmented Frames Register

The EMAC_RXIPV4_FRAG_FRM register contains a count of the number of good IPv4 datagrams with fragmentation.

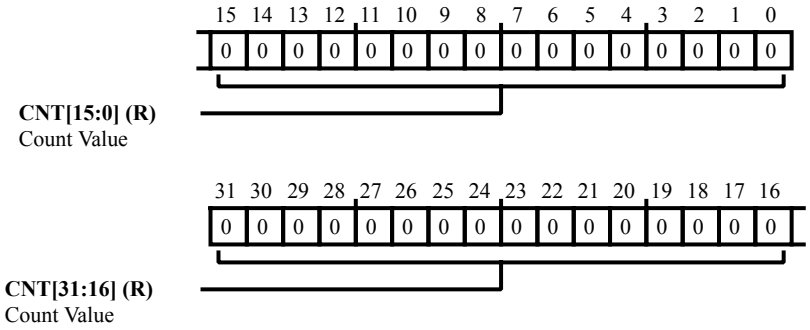

Figure 23-64: EMAC_RXIPV4_FRAG_FRM Register Diagram

Table 23-94: EMAC_RXIPV4_FRAG_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams Fragmented Octets Register

The EMAC_RXIPV4_FRAG_OCT register contains a count of the number of bytes received in fragmented IPv4 datagrams. The value in the IPv4 headers Length field is used to update this counter.

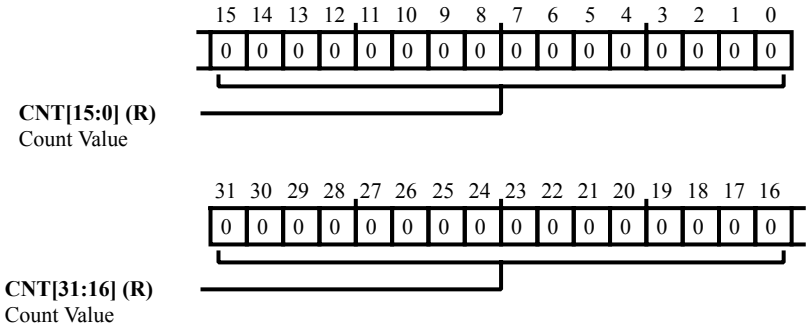

Figure 23-65: EMAC_RXIPV4_FRAG_OCT Register Diagram

Table 23-95: EMAC_RXIPV4_FRAG_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams (Good) Register

The EMAC_RXIPV4_GD_FRM register contains a count of the number of good IPv4 datagrams received with the TCP, UDP, or ICMP payload.

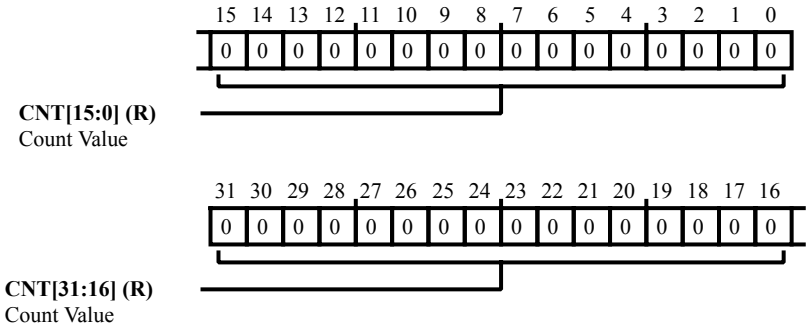

Figure 23-66: EMAC_RXIPV4_GD_FRM Register Diagram

Table 23-96: EMAC_RXIPV4_GD_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams Good Octets Register

The EMAC_RXIPV4_GD_OCT register contains a count of the number of bytes received in good IPv4 datagrams encapsulating TCP, UDP, or ICMP data.

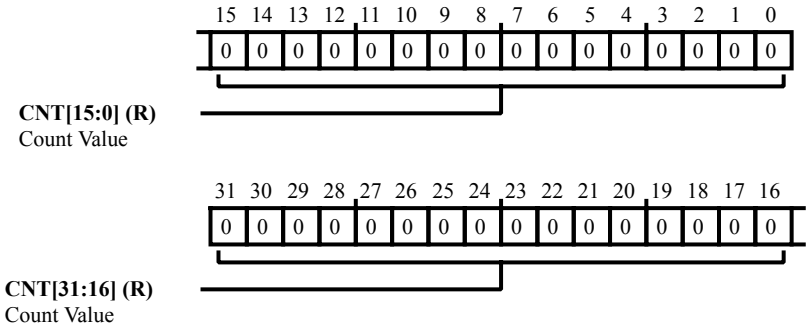

Figure 23-67: EMAC_RXIPV4_GD_OCT Register Diagram

Table 23-97: EMAC_RXIPV4_GD_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams Header Errors Register

The EMAC_RXIPV4_HDR_ERR_FRM register contains a count of the number of IPv4 datagrams received with header (checksum, length, or version mismatch) errors.

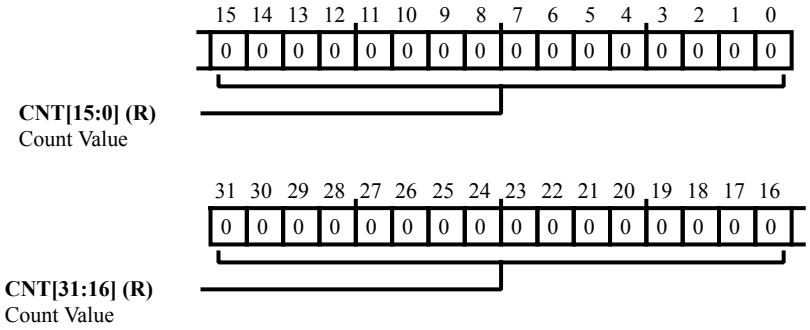

Figure 23-68: EMAC_RXIPV4_HDR_ERR_FRM Register Diagram

Table 23-98: EMAC_RXIPV4_HDR_ERR_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams Header Errors Register

The EMAC_RXIPV4_HDR_ERR_OCT register contains a count of the number of bytes received in IPv4 datagrams with header errors (checksum, length, version mismatch). The value in the Length field of IPv4 header is used to update this counter.

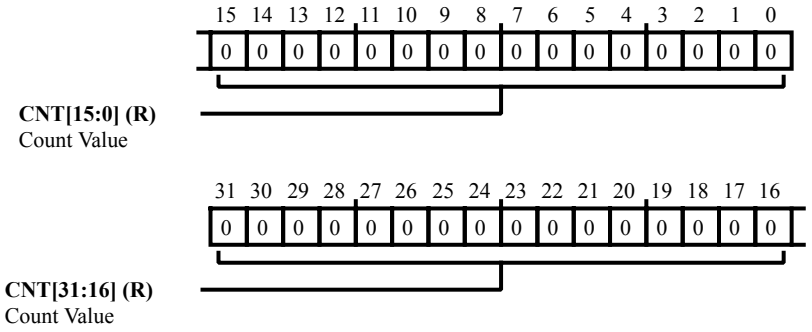

Figure 23-69: EMAC_RXIPV4_HDR_ERR_OCT Register Diagram

Table 23-99: EMAC_RXIPV4_HDR_ERR_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams No Payload Frame Register

The EMAC_RXIPV4_NOPAY_FRM register contains a count of the number of IPv4 datagram frames received that did not have a TCP, UDP, or ICMP payload processed by the Checksum engine.

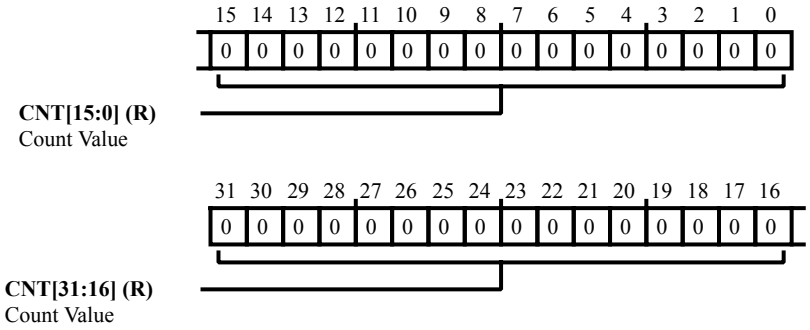

Figure 23-70: EMAC_RXIPV4_NOPAY_FRM Register Diagram

Table 23-100: EMAC_RXIPV4_NOPAY_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams No Payload Octets Register

The EMAC_RXIPV4_NOPAY_OCT register contains a count of the number of bytes received in IPv4 datagrams that did not have a TCP, UDP, or ICMP payload. The value in the IPv4 headers Length field is used to update this counter.

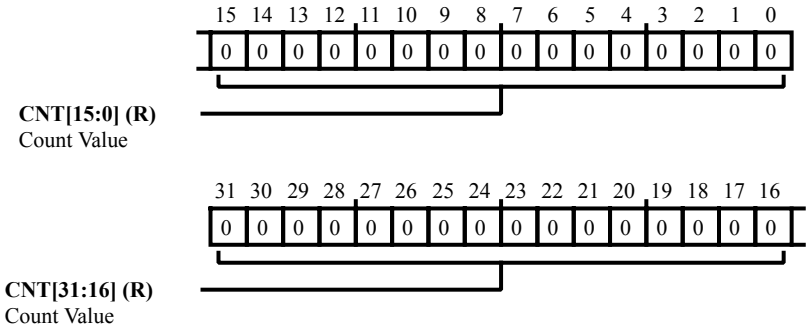

Figure 23-71: EMAC_RXIPV4_NOPAY_OCT Register Diagram

Table 23-101: EMAC_RXIPV4_NOPAY_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 UDP Disabled Frames Register

The EMAC_RXIPV4_UDSBL_FRM register contains a count of the number of good IPv4 datagrams received that had a UDP payload with checksum disabled.

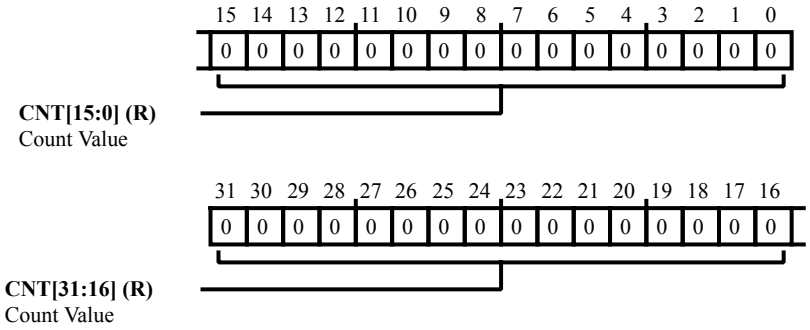

Figure 23-72: EMAC_RXIPV4_UDSBL_FRM Register Diagram

Table 23-102: EMAC_RXIPV4_UDSBL_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 UDP Disabled Octets Register

The EMAC_RXIPV4_UDSBL_OCT register contains a count of the number of bytes received in a UDP segment that had the UDP checksum disabled. This counter does not count IP Header bytes.

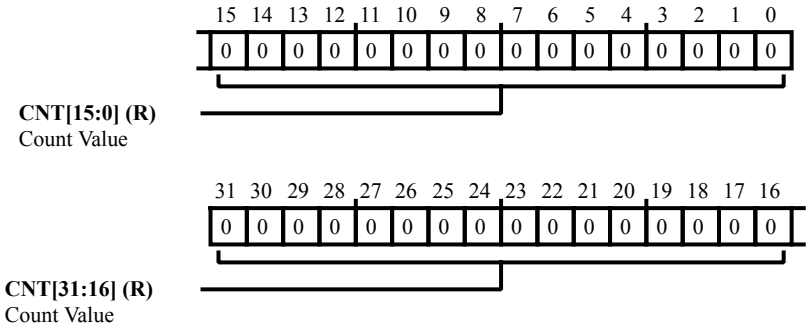

Figure 23-73: EMAC_RXIPV4_UDSBL_OCT Register Diagram

Table 23-103: EMAC_RXIPV4_UDSBL_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv6 Datagrams Good Frames Register

The EMAC_RXIPV6_GD_FRM register contains a count of the number of good IPv6 datagrams received with TCP, UDP, or ICMP payloads.

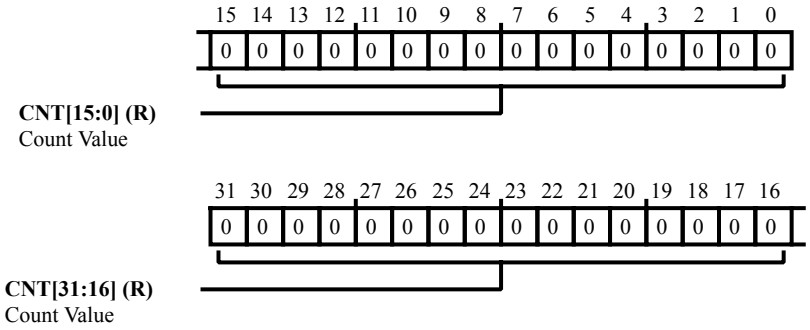

Figure 23-74: EMAC_RXIPV6_GD_FRM Register Diagram

Table 23-104: EMAC_RXIPV6_GD_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv6 Good Octets Register

The EMAC_RXIPV6_GD_OCT register contains a count of the number of bytes received in good IPv6 datagrams encapsulating TCP, UDP or ICMPv6 data

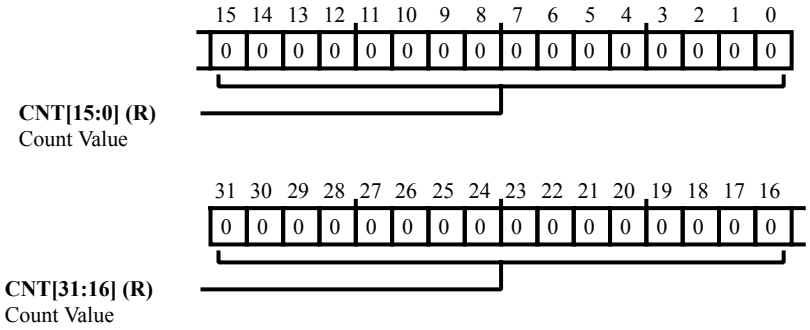

Figure 23-75: EMAC_RXIPV6_GD_OCT Register Diagram

Table 23-105: EMAC_RXIPV6_GD_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv6 Datagrams Header Error Frames Register

The EMAC_RXIPV6_HDR_ERR_FRM register contains a count of the number of IPv6 datagrams received with header errors (length or version mismatch).

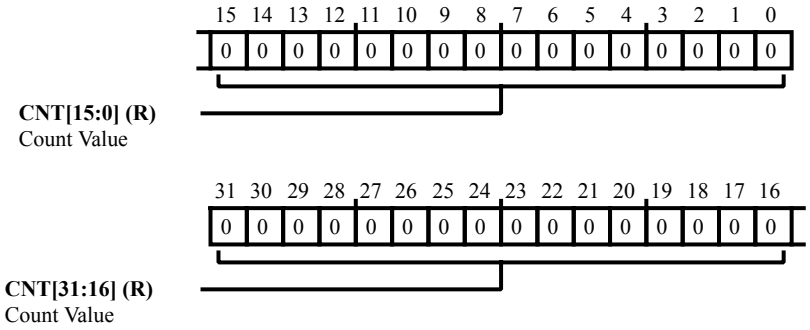

Figure 23-76: EMAC_RXIPV6_HDR_ERR_FRM Register Diagram

Table 23-106: EMAC_RXIPV6_HDR_ERR_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv6 Header Errors Register

The EMAC_RXIPV6_HDR_ERR_OCT register contains a count of the number of bytes received in IPv6 datagrams with header errors (length, version mismatch). The value in the IPv6 headers Length field is used to update this counter.

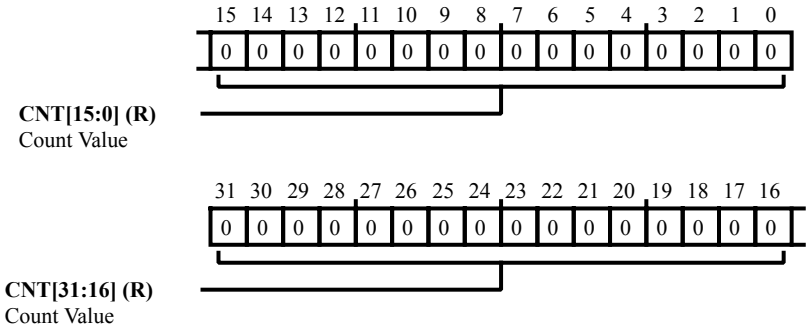

Figure 23-77: EMAC_RXIPV6_HDR_ERR_OCT Register Diagram

Table 23-107: EMAC_RXIPV6_HDR_ERR_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv6 Datagrams No Payload Frames Register

The EMAC_RXIPV6_NOPAY_FRM register contains a count of the number of IPv6 datagram frames received that did not have a TCP, UDP, or ICMP payload. This includes all IPv6 datagrams with fragmentation or security extension headers.

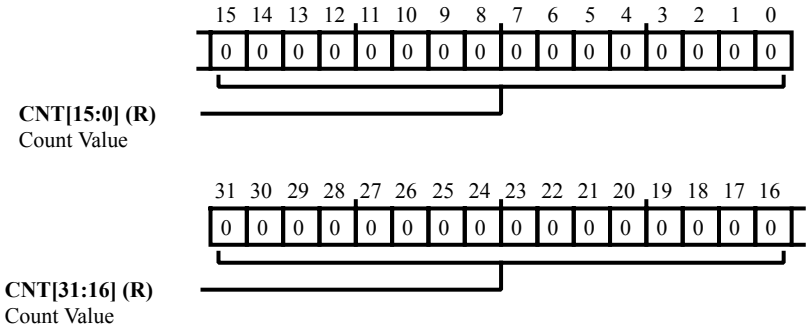

Figure 23-78: EMAC_RXIPV6_NOPAY_FRM Register Diagram

Table 23-108: EMAC_RXIPV6_NOPAY_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv6 No Payload Octets Register

The EMAC_RXIPV6_NOPAY_OCT register contains a count of the number of bytes received in IPv6 datagrams that did not have a TCP, UDP, or ICMP payload. The value in the IPv6 headers Length field is used to update this counter.

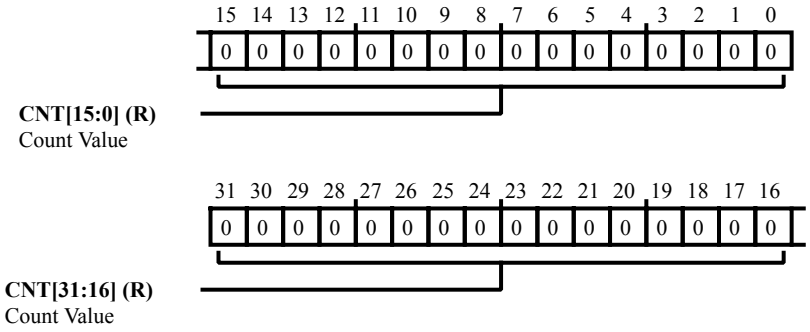

Figure 23-79: EMAC_RXIPV6_NOPAY_OCT Register Diagram

Table 23-109: EMAC_RXIPV6_NOPAY_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Jab Error Register

The EMAC_RXJAB_ERR register contains a count of the number of giant frames received with length greater than 1,518 bytes and with CRC error.

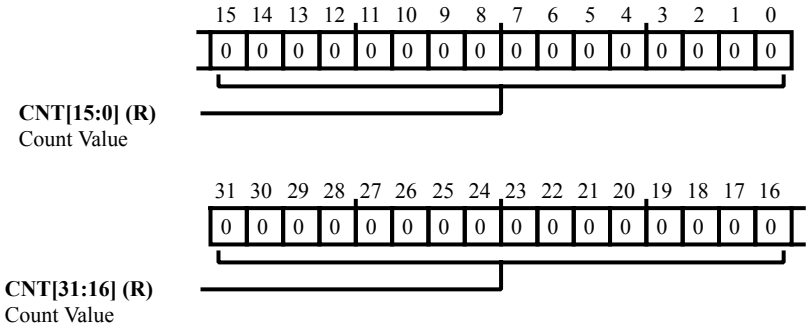

Figure 23-80: EMAC_RXJAB_ERR Register Diagram

Table 23-110: EMAC_RXJAB_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Length Error Register

The EMAC_RXLEN_ERR register contains a count of the number of frames received with length error (Length type field frame size), for all frames with valid length field.

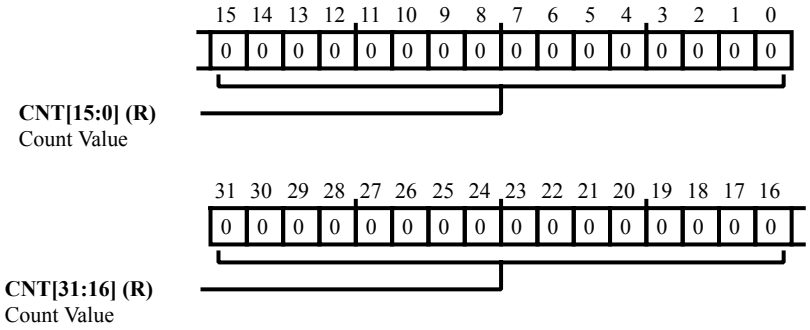

Figure 23-81: EMAC_RXLEN_ERR Register Diagram

Table 23-111: EMAC_RXLEN_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Multicast Frames (Good) Register

The EMAC_RXMCASTFRM_G register contains a count of the number of good multicast frames received.

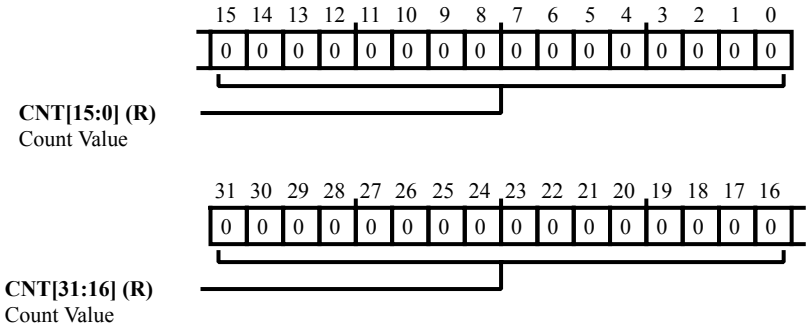

Figure 23-82: EMAC_RXMCASTFRM_G Register Diagram

Table 23-112: EMAC_RXMCASTFRM_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Octet Count (Good) Register

The EMAC_RXOCTCNT_G register contains a count of the number of bytes received, exclusive of preamble, only in good frames.

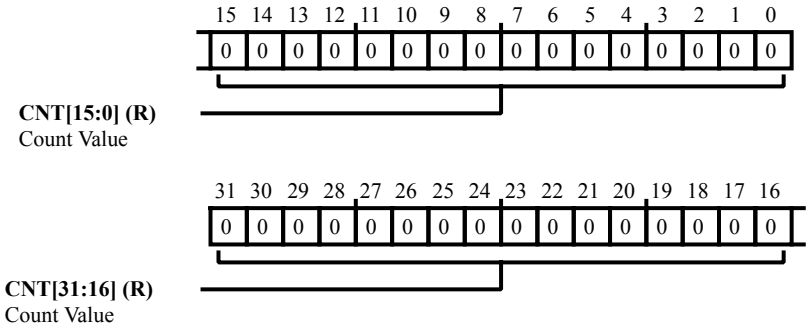

Figure 23-83: EMAC_RXOCTCNT_G Register Diagram

Table 23-113: EMAC_RXOCTCNT_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Octet Count (Good/Bad) Register

The EMAC_RXOCTCNT_GB register contains a count of the number of bytes received, exclusive of preamble, in good and bad frames.

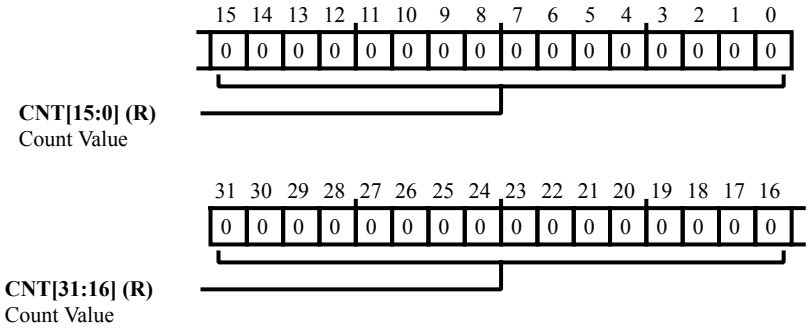

Figure 23-84: EMAC_RXOCTCNT_GB Register Diagram

Table 23-114: EMAC_RXOCTCNT_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Out Of Range Type Register

The EMAC_RXOORTYPE register contains a count of the number of frames received with length field not equal to the valid frame size (greater than 1,500 but less than 1,536).

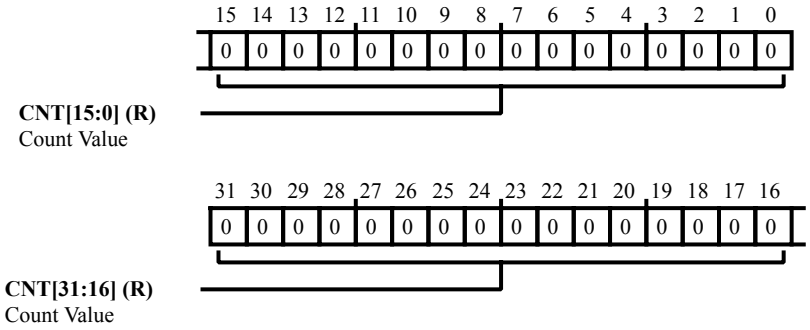

Figure 23-85: EMAC_RXOORTYPE Register Diagram

Table 23-115: EMAC_RXOORTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Oversize (Good) Register

The EMAC_RXOSIZE_G register contains a count of the number of frames received with length greater than the maxsize, without errors.

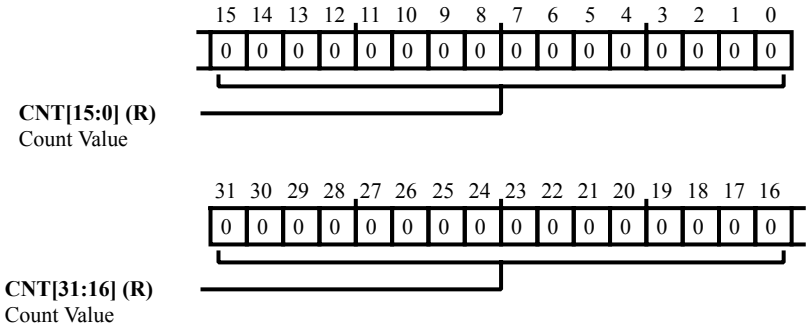

Figure 23-86: EMAC_RXOSIZE_G Register Diagram

Table 23-116: EMAC_RXOSIZE_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Pause Frames Register

The EMAC_RXPAUSEFRM register contains a count of the number of good and valid PAUSE frames received.

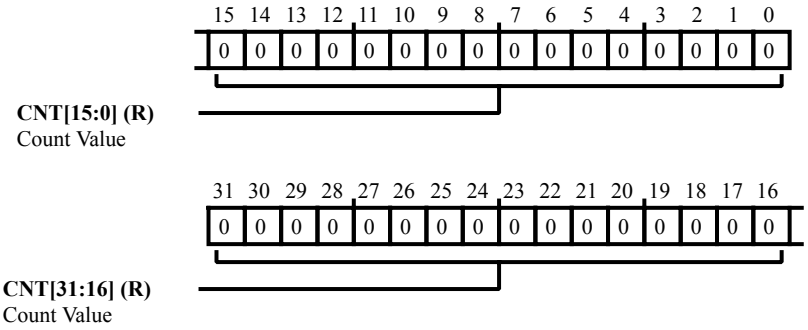

Figure 23-87: EMAC_RXPAUSEFRM Register Diagram

Table 23-117: EMAC_RXPAUSEFRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Runt Error Register

The EMAC_RXRUNT_ERR register contains a count of the number of frames received with runt error.

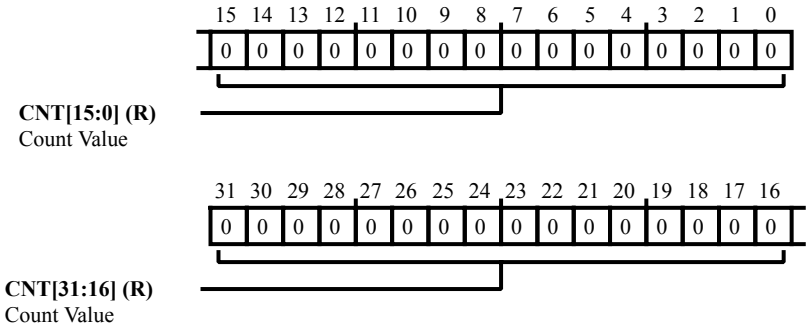

Figure 23-88: EMAC_RXRUNT_ERR Register Diagram

Table 23-118: EMAC_RXRUNT_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx TCP Error Frames Register

The EMAC_RXTCP_ERR_FRM register contains a count of the number of good IP datagrams whose TCP payload has a checksum error.

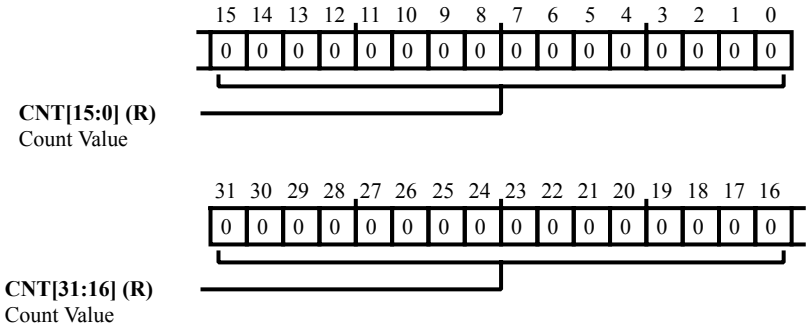

Figure 23-89: EMAC_RXTCP_ERR_FRM Register Diagram

Table 23-119: EMAC_RXTCP_ERR_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx TCP Error Octets Register

The EMAC_RXTCP_ERR_OCT register contains a count of the number of bytes received in a TCP segment with checksum errors.

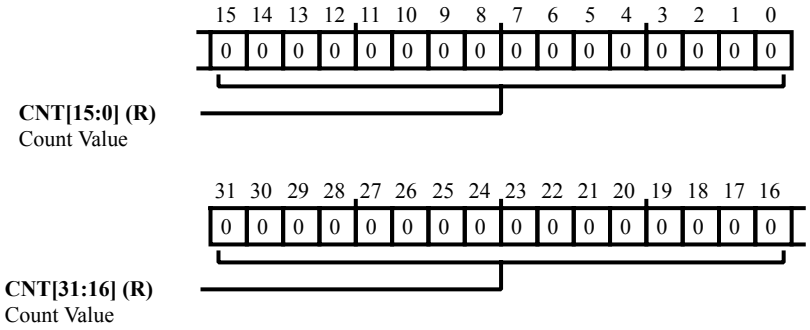

Figure 23-90: EMAC_RXTCP_ERR_OCT Register Diagram

Table 23-120: EMAC_RXTCP_ERR_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx TCP Good Frames Register

The EMAC_RXTCP_GD_FRM register contains a count of the number of good IP datagrams with a good TCP payload.

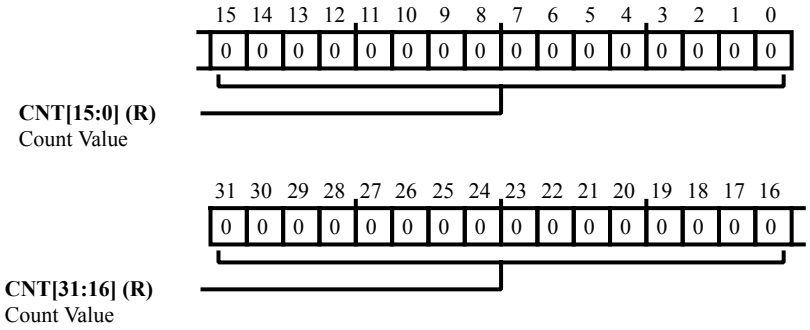

Figure 23-91: EMAC_RXTCP_GD_FRM Register Diagram

Table 23-121: EMAC_RXTCP_GD_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx TCP Good Octets Register

The EMAC_RXTCP_GD_OCT register contains a count of the number of bytes received in a good TCP segment.

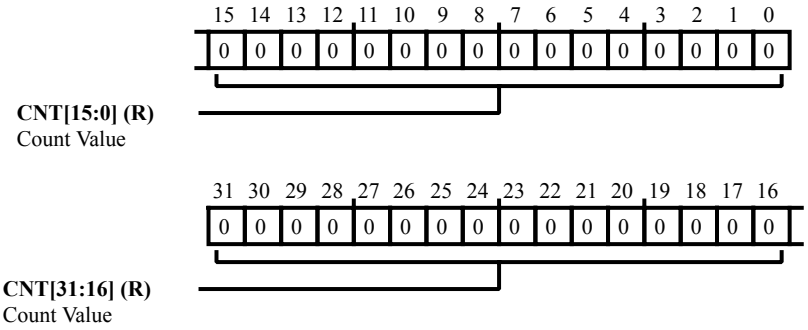

Figure 23-92: EMAC_RXTCP_GD_OCT Register Diagram

Table 23-122: EMAC_RXTCP_GD_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Unicast Frames (Good) Register

The EMAC_RXUCASTFRM_G register contains a count of the number of good unicast frames received.

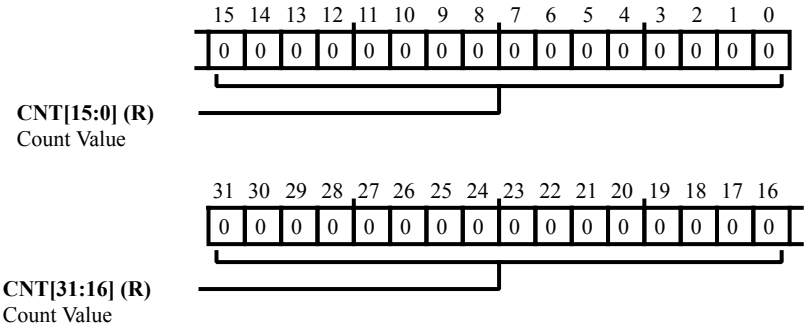

Figure 23-93: EMAC_RXUCASTFRM_G Register Diagram

Table 23-123: EMAC_RXUCASTFRM_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx UDP Error Frames Register

The EMAC_RXUDP_ERR_FRM register contains a count of the number of good IP datagrams whose UDP payload has a checksum error.

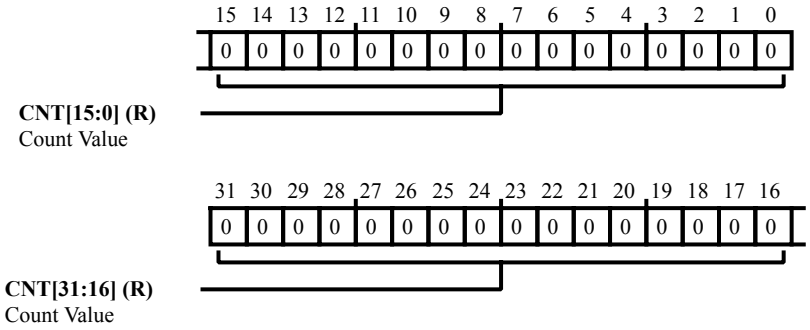

Figure 23-94: EMAC_RXUDP_ERR_FRM Register Diagram

Table 23-124: EMAC_RXUDP_ERR_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx UDP Error Octets Register

The EMAC_RXUDP_ERR_OCT register contains a count of the number of bytes received in a UDP segment that had checksum errors.

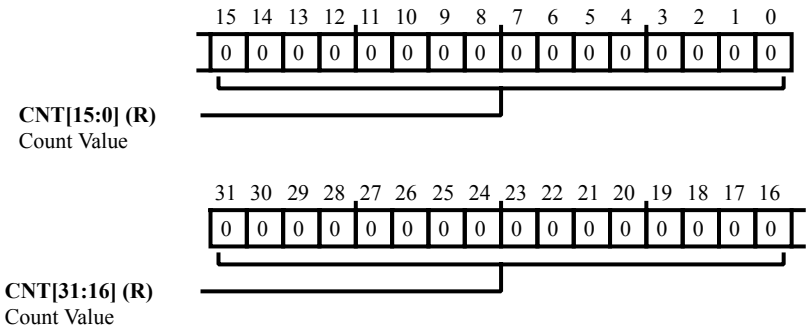

Figure 23-95: EMAC_RXUDP_ERR_OCT Register Diagram

Table 23-125: EMAC_RXUDP_ERR_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx UDP Good Frames Register

The EMAC_RXUDP_GD_FRM register contains a count of the number of good IP datagrams with a good UDP payload. This counter is not updated when the rxipv4_udsbl_frms counter is incremented.

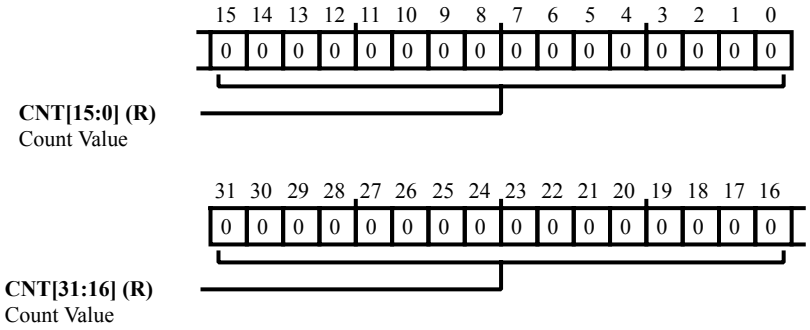

Figure 23-96: EMAC_RXUDP_GD_FRM Register Diagram

Table 23-126: EMAC_RXUDP_GD_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx UDP Good Octets Register

The EMAC_RXUDP_GD_OCT register contains a count of the number of bytes received in a good UDP segment. This counter (and the counters below) does not count IP header bytes.

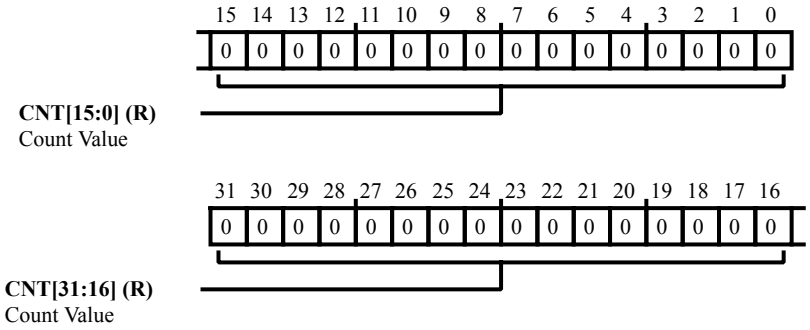

Figure 23-97: EMAC_RXUDP_GD_OCT Register Diagram

Table 23-127: EMAC_RXUDP_GD_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Undersize (Good) Register

The EMAC_RXUSIZE_G register contains a count of the number of frames received with length less than 64 bytes, without any errors.

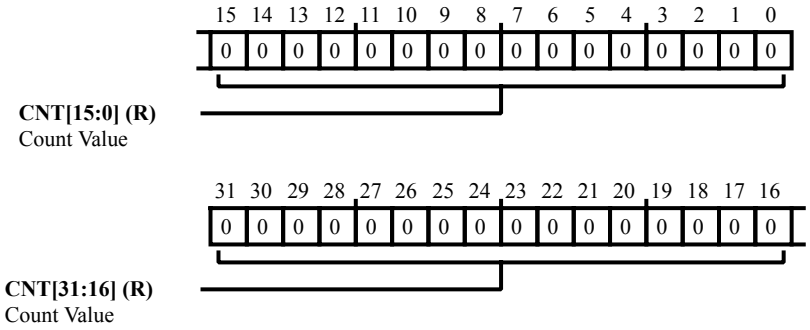

Figure 23-98: EMAC_RXUSIZE_G Register Diagram

Table 23-128: EMAC_RXUSIZE_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx VLAN Frames (Good/Bad) Register

The EMAC_RXVLANFRM_GB register contains a count of the number of good and bad VLAN frames received.

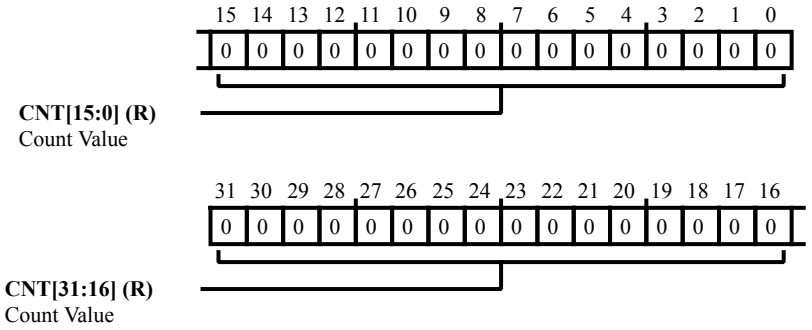

Figure 23-99: EMAC_RXVLANFRM_GB Register Diagram

Table 23-129: EMAC_RXVLANFRM_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Watch Dog Error Register

The EMAC_RXWDOG_ERR register contains a count of the number of frames received with error due to watchdog timeout error (frames with a data load larger than 2,048 bytes).

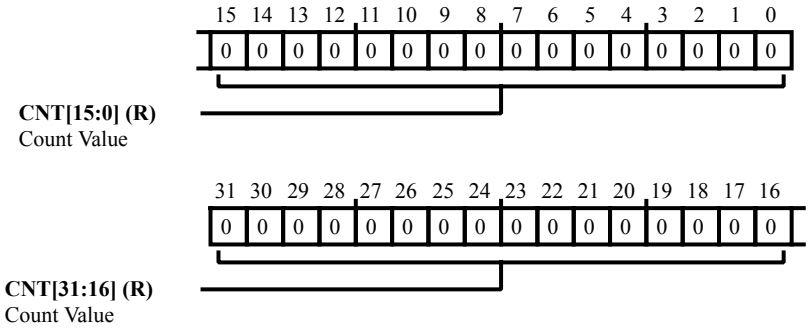

Figure 23-100: EMAC_RXWDOG_ERR Register Diagram

Table 23-130: EMAC_RXWDOG_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

SMI Address Register

The EMAC_SMI_ADDR register contains the station management interface address and feature settings.

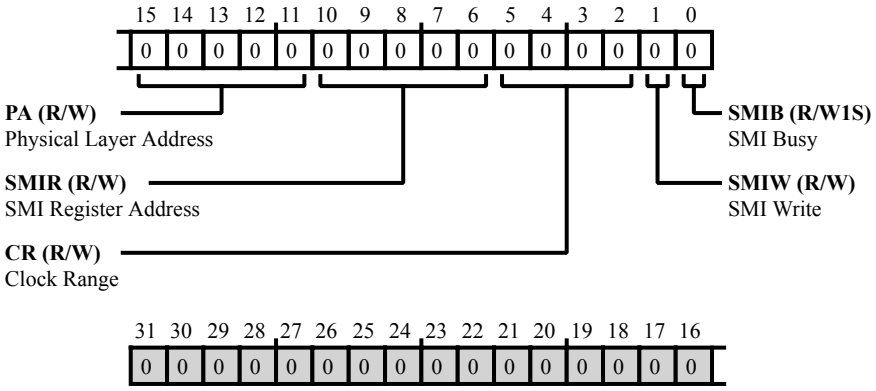

Figure 23-101: EMAC_SMI_ADDR Register Diagram

Table 23-131: EMAC_SMI_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:11 (R/W)	PA	Physical Layer Address. The EMAC_SMI_ADDR.PA bits select the PHY. This field tells which of the 32 possible PHY devices are being accessed.	
10:6 (R/W)	SMIR	SMI Register Address. The EMAC_SMI_ADDR.SMIR bits select the desired Station Management Interface register in the selected PHY device.	
5:2 (R/W)	CR	Clock Range. The EMAC_SMI_ADDR.CR bits select the Clock Range, determining the frequency of the MDC clock as per the SYSCLK frequency. The suggested range of SYSCLK frequency applicable for each value below (when Bit[5] =0) ensures that the MDC clock is approximately between the frequency range 1.0 MHz - 2.5 MHz. When the MSB of this field is set, you can achieve MDC clock of frequency higher than the IEEE 802.3 specified frequency limit of 2.5 MHz and program a clock divider of lower value. For example, when SYSCLK=100 MHz and you program these bits to b#1010, the resulting MDC clock is 12.5 MHz, which is outside the limit of IEEE 802.3 specified range. Use the values shown only if the interface chips support faster MDC clocks.	
		0	MDC Clock= SYSCLK/42 (for SYSCLK=60-100MHz)
		1	MDC Clock= SYSCLK/62 (for SYSCLK=100-125 MHz)
		2	MDC Clock= SYSCLK/16 (for SYSCLK=20-35 MHz)
		3	MDC Clock= SYSCLK/26 (for SYSCLK=35-60 MHz)
		8	MDC Clock= SYSCLK/4

Table 23-131: EMAC_SMI_ADDR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		9	MDC Clock= SYSCLK/6
		10	MDC Clock= SYSCLK/8
		11	MDC Clock= SYSCLK/10
		12	MDC Clock= SYSCLK/12
		13	MDC Clock= SYSCLK/14
		14	MDC Clock= SYSCLK/16
		15	MDC Clock= SYSCLK/18
1 (R/W)	SMIW	SMI Write. The EMAC_SMI_ADDR.SMIW bit, when set, tells the PHY this is a Write operation using the Station Management Interface Data register. If this bit is not set, this is a Read operation.	
0 (R/W1S)	SMIB	SMI Busy. The EMAC_SMI_ADDR.SMIB bit should read low (=0) before writing to the EMAC_SMI_ADDR and EMAC_SMI_DATA registers. This bit must also =0 during a Write to EMAC_SMI_ADDR. During a PHY register access, this bit is set (=1) by the Application to indicate that a Read or Write access is in progress. The EMAC_SMI_DATA register should be kept valid until this bit is cleared by the MAC during a PHY Write operation. EMAC_SMI_DATA is invalid until this bit is cleared by the MAC during a PHY Read operation. The EMAC_SMI_ADDR should not be written to until this bit is cleared.	

SMI Data Register

The EMAC_SMI_DATA register contains the station management interface data.

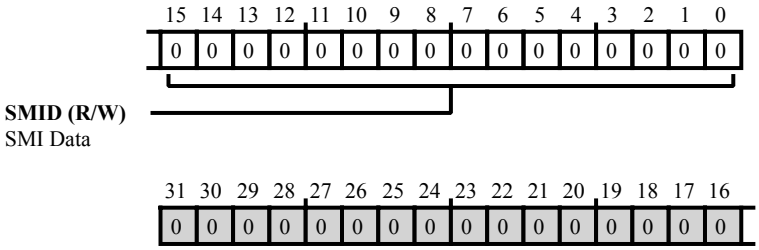

Figure 23-102: EMAC_SMI_DATA Register Diagram

Table 23-132: EMAC_SMI_DATA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	SMID	SMI Data. The EMAC_SMI_DATA.SMID bits contain the 16-bit data value read from the PHY after a Management Read operation or the 16-bit data value to be written to the PHY before a Management Write operation.

Time Stamp Addend Register

The EMAC_TM_ADDEND register lets software adjust the clock frequency linearly to match the master clock frequency.

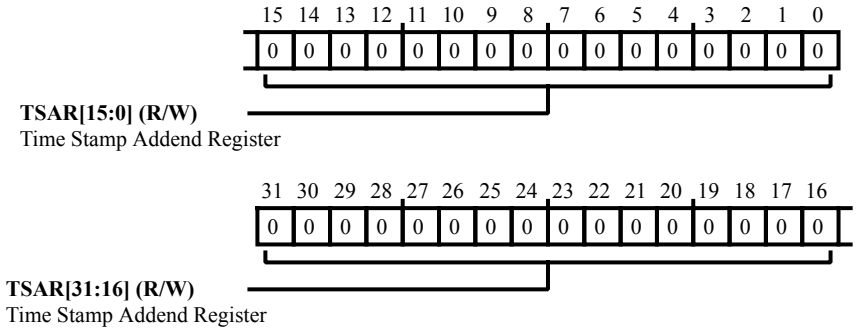

Figure 23-103: EMAC_TM_ADDEND Register Diagram

Table 23-133: EMAC_TM_ADDEND Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TSAR	Time Stamp Addend Register. The EMAC_TM_ADDEND.TSAR bits indicate the 32-bit time value to be added to the Accumulator register to achieve time synchronization.

Time Stamp Auxiliary TS Nano Seconds Register

The EMAC_TM_AUXSTMP_NSEC register contains the low 32 bits (nanoseconds field) of the auxiliary time stamp.

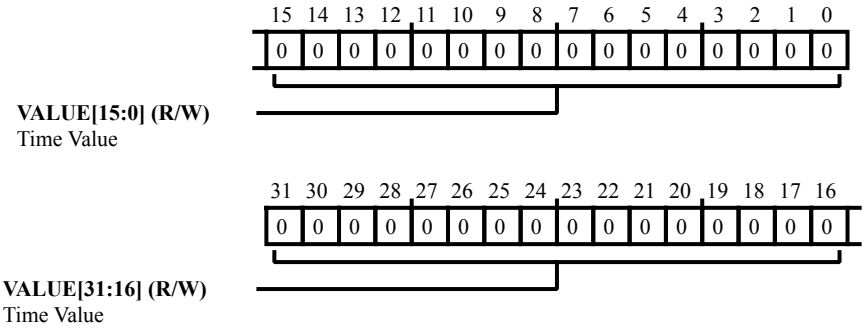

Figure 23-104: EMAC_TM_AUXSTMP_NSEC Register Diagram

Table 23-134: EMAC_TM_AUXSTMP_NSEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Time Value.

Time Stamp Auxiliary TM Seconds Register

The EMAC_TM_AUXSTMP_SEC register contains the low 32 bits of the seconds field of the auxiliary time stamp.

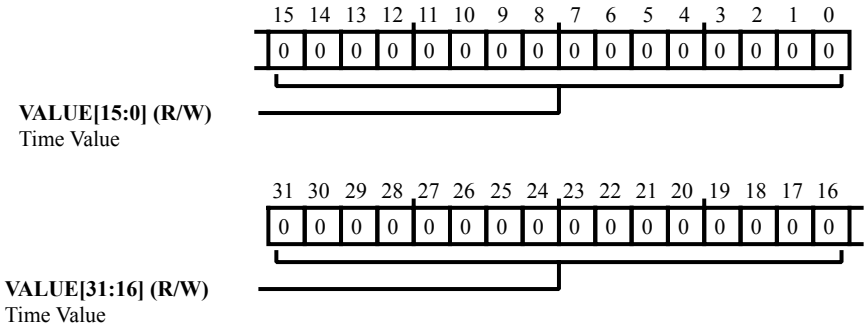

Figure 23-105: EMAC_TM_AUXSTMP_SEC Register Diagram

Table 23-135: EMAC_TM_AUXSTMP_SEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Time Value.

Time Stamp Control Register

The EMAC_TM_CTL register controls time stamp generation and update. The EMAC_TM_CTL.SNAPTYPSEL, EMAC_TM_CTL.TSMSTRENA, and EMAC_TM_CTL.TSEVNTENA bits work together to decide the set of PTP packet types for which snapshot needs to be taken. (Encoding shown in table.)

SNAPTYPSEL()	TSMSTRENA	TSEVNTENA	Messages for which snapshot is taken
00	X	0	SYNC, Follow_Up, Delay_Req, Delay_Resp
00	0	1	SYNC
00	1	1	Delay_Req
01	X	0	SYNC, Follow_Up, Delay_Req, Delay_Resp, Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up
01	0	1	SYNC, Pdelay_Req, Pdelay_Resp
01	1	1	Delay_Req, Pdelay_Req, Pdelay_Resp
10	X	X	SYNC, Delay_Req
11	X	X	Pdelay_Req, Pdelay_Resp

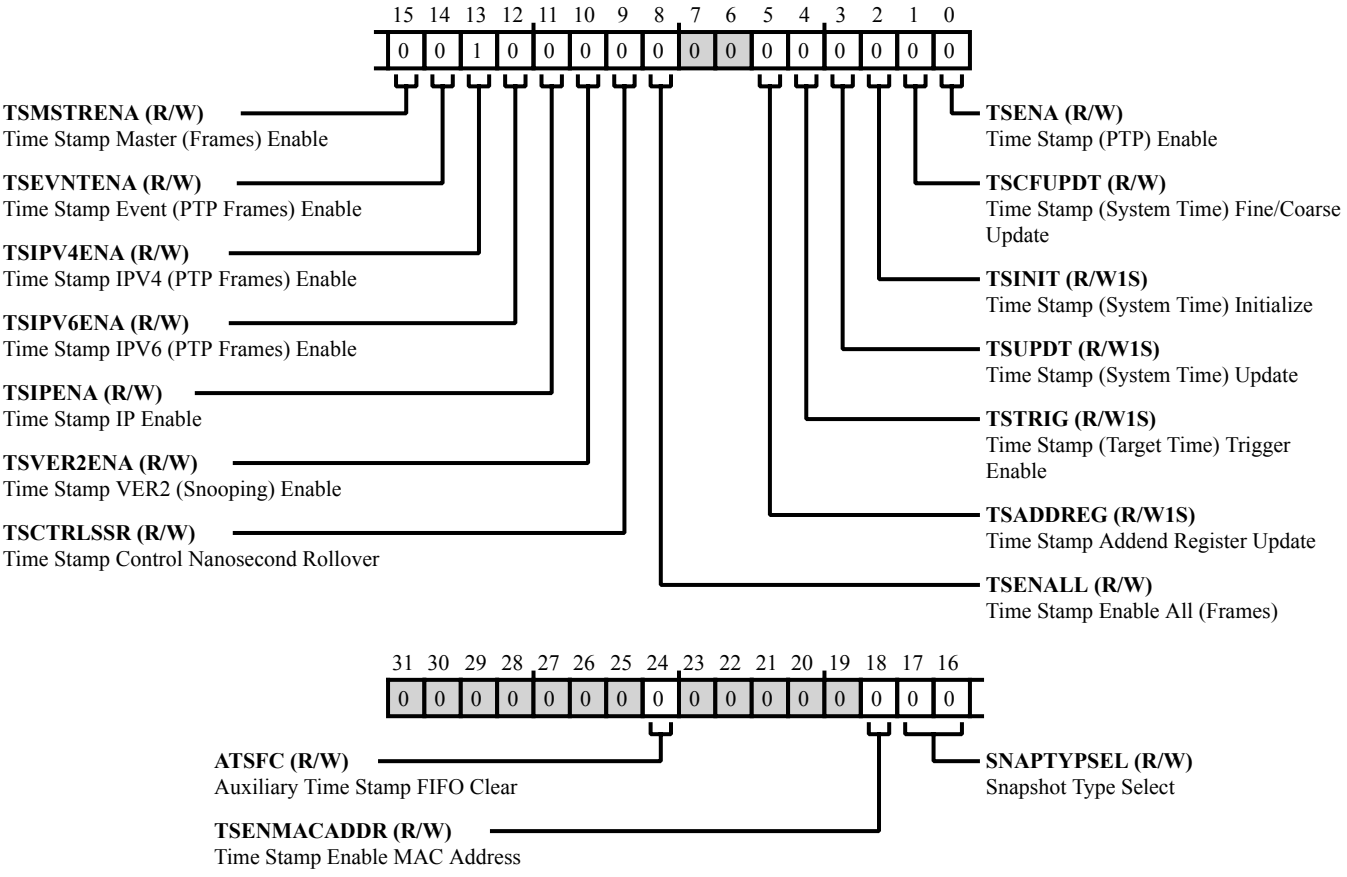

Figure 23-106: EMAC_TM_CTL Register Diagram

Table 23-136: EMAC_TM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
24 (R/W)	ATSFC	Auxiliary Time Stamp FIFO Clear. The EMAC_TM_CTL.ATSFC bit, when set, resets the pointers of the Auxiliary Snapshot FIFO. This bit is cleared when the pointers are reset and the FIFO is empty. When this bit is cleared, auxiliary snapshots gets stored in the FIFO.	
18 (R/W)	TSENMACADDR	Time Stamp Enable MAC Address. The EMAC_TM_CTL.TSENMACADDR bit, when set, uses the DA MAC address (that matches the EMAC_ADDR0_LO and EMAC_ADDR0_HI registers) to filter the PTP frames when PTP is sent directly over Ethernet.	
		0	Disable PTP MAC address filter
		1	Enable PTP MAC address filter
17:16 (R/W)	SNAPTYPSEL	Snapshot Type Select. The EMAC_TM_CTL.SNAPTYPSEL bits along with bit 15 and 14 decide the set of PTP packet types for which snapshot needs to be taken. (See the table in the EMAC_TM_CTL register description.)	
15 (R/W)	TSMSTRENA	Time Stamp Master (Frames) Enable. The EMAC_TM_CTL.TSMSTRENA bit, when set, takes the snapshot for messages relevant to master node only else snapshot is taken for PTP messages relevant to slave node.	
		0	Enable Snapshot for Slave Messages
		1	Enable Snapshot for Master Messages
14 (R/W)	TSEVNTENA	Time Stamp Event (PTP Frames) Enable. The EMAC_TM_CTL.TSEVNTENA bit, when set, takes the time stamp snapshot for PTP event messages only (SYNC, Delay_Req, Pdelay_Req, or Pdelay_Resp). When reset, the snapshot is taken for all PTP messages except Announce, Management, and Signaling.	
		0	Enable Time Stamp for All Messages
		1	Enable Time Stamp for Event Messages Only
13 (R/W)	TSIPV4ENA	Time Stamp IPV4 (PTP Frames) Enable. The EMAC_TM_CTL.TSIPV4ENA bit, when set, directs the EMAC receiver to process the PTP packets encapsulated in UDP over IPv4 packets. When this bit is clear, the MAC ignores the PTP transported over UDP-IPv4 packets. This bit is set by default.	
		0	Disable Time Stamp for PTP Over IPv4 Frames
		1	Enable Time Stamp for PTP Over IPv4 Frames

Table 23-136: EMAC_TM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	TSIPV6ENA	Time Stamp IPV6 (PTP Frames) Enable. The EMAC_TM_CTL.TSIPV6ENA bit, when set, directs the EMAC receiver to process PTP packets encapsulated in UDP over IPv6 packets. When this bit is clear, the MAC ignores the PTP transported over UDP-IPv6 packets.	
		0	Disable Time Stamp for PTP Over IPv6 frames
		1	Enable Time Stamp for PTP Over IPv6 Frames
11 (R/W)	TSIPENA	Time Stamp IP Enable. The EMAC_TM_CTL.TSIPENA bit, when set, directs the EMAC receiver to process the PTP packets encapsulated directly in the Ethernet frames. When this bit is clear, the MAC ignores PTP over Ethernet packets.	
		0	Disable PTP Over Ethernet Frames
		1	Enable PTP Over Ethernet Frames
10 (R/W)	TSVER2ENA	Time Stamp VER2 (Snooping) Enable. The EMAC_TM_CTL.TSVER2ENA bit, when set, processes the PTP packets using the 1588 version 2 format (enables PTP packet snooping for VER2) else processed using the version 1 format.	
		0	Disable packet snooping for V2 frames
		1	Enable packet snooping for V2 frames
9 (R/W)	TSCTRLSSR	Time Stamp Control Nanosecond Rollover. The EMAC_TM_CTL.TSCTRLSSR bit, when set, rolls over the EMAC_TM_NSEC register after 0x3B9A_C9FF value (10^9-1) and increments the EMAC_TM_SEC register. When reset, the roll over value of EMAC_TM_NSEC register is 0x7FFF_FFFF. The nanosecond increment has to be programmed correctly depending on the PTP reference clock frequency and this bit value.	
		0	Roll Over Nanosecond After 0x7FFFFFFF
		1	Roll Over Nanosecond After 0x3B9AC9FF
8 (R/W)	TSENALL	Time Stamp Enable All (Frames). The EMAC_TM_CTL.TSENALL bit, when set, enables the time stamp snapshot for all frames received by the core.	
		0	Disable timestamp for all frames
		1	Enable timestamp for all frames
5 (R/W1S)	TSADDREG	Time Stamp Addend Register Update. The EMAC_TM_CTL.TSADDREG bit, when set, updates the contents of the EMAC_TM_ADDEND register for fine correction. This bit is cleared when the update is completed. This bit should be zero before setting it.	

Table 23-136: EMAC_TM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W1S)	TSTRIG	Time Stamp (Target Time) Trigger Enable. The EMAC_TM_CTL.TSTRIG bit, when set, generates the time stamp interrupt when the System Time becomes greater than the value written in Time Stamp Target Time Seconds register. This bit is reset after the generation of the Time Stamp Trigger Interrupt.	
		1	Interrupt (TS) if system time is greater than target time register
3 (R/W1S)	TSUPDT	Time Stamp (System Time) Update. The EMAC_TM_CTL.TSUPDT bit, when set, updates (adds/subtracts) the system time with the value specified in the EMAC_TM_SECUPDT and EMAC_TM_NSECUPDT registers. This bit should read =0 before updating it. This bit is reset when the update is completed in hardware. The EMAC_TM_NSEC register is not updated.	
		1	System time updated with Time stamp register values
2 (R/W1S)	TSINIT	Time Stamp (System Time) Initialize. The EMAC_TM_CTL.TSINIT bit, when set, initializes (over-writes) the system time with the value specified in the EMAC_TM_SECUPDT and EMAC_TM_NSECUPDT registers. This bit should read =0 before updating it. This bit is reset when the initialization is complete. Only the EMAC_TM_NSEC register can be initialized.	
		1	System time initialized with Time stamp register values
1 (R/W)	TSCFUPDT	Time Stamp (System Time) Fine/Coarse Update. The EMAC_TM_CTL.TSCFUPDT bit, when set, indicates that the system time update is done using the fine correction method. When reset, it indicates the system time correction to be done using Coarse method.	
		0	Use Coarse Correction Method for System Time Update
		1	Use Fine Correction Method for System Time Update
0 (R/W)	TSENA	Time Stamp (PTP) Enable. The EMAC_TM_CTL.TSENA bit, when set, enables PTP module for time stamping transmitted and received frames. It also enables System Time which is used for time stamping the frames. Programs should initialize the System Time after setting this bit.	
		0	Disable PTP Module
		1	Enable PTP Module

Time Stamp High Second Register

The EMAC_TM_HISEC register contains the upper 32 bits of the seconds field of the system time.

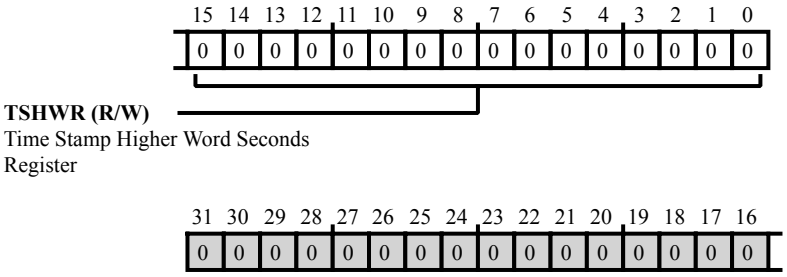

Figure 23-107: EMAC_TM_HISEC Register Diagram

Table 23-137: EMAC_TM_HISEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	TSHWR	Time Stamp Higher Word Seconds Register. The EMAC_TM_HISEC.TSHWR bit field contains the most significant 16-bits of the time stamp seconds value. The register is directly written to initialize the value. This register is incremented when there is an overflow from the 32-bits of the EMAC_TM_SEC register.

Time Stamp Nanoseconds Register

The EMAC_TM_NSEC register contains the nanoseconds field of the system time.

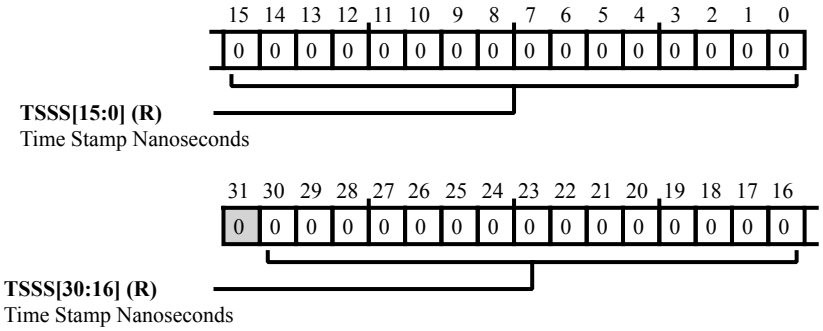

Figure 23-108: EMAC_TM_NSEC Register Diagram

Table 23-138: EMAC_TM_NSEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30:0 (R/NW)	TSSS	Time Stamp Nanoseconds. The value in the EMAC_TM_NSEC.TSSS bit field has the nanosecond representation of time, with an accuracy of 0.46 nanosecond. (When EMAC_TM_CTL.TSCTRLSSR is set, each bit represents 1 ns and the maximum value will be 0x3B9A_C9FF, after which it rolls-over to zero).

Time Stamp Nanoseconds Update Register

The EMAC_TM_NSECUPDT register contains the low 32 bits to be added to, subtracted from, or written to the nanoseconds field of the system time.

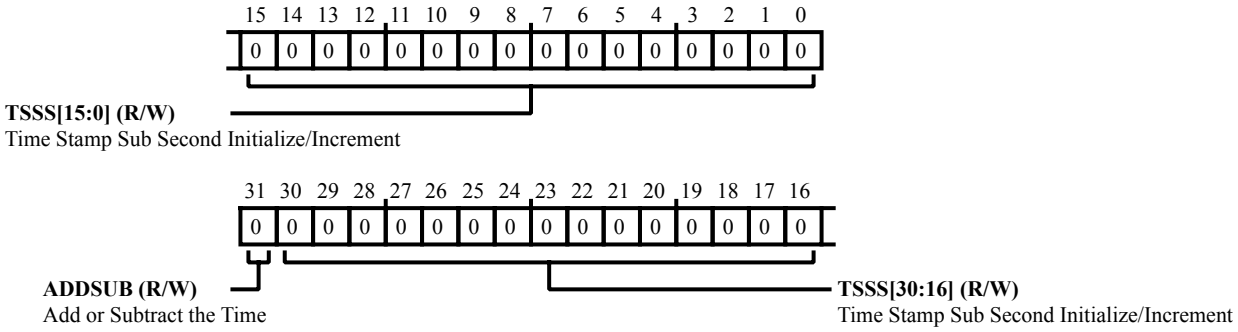

Figure 23-109: EMAC_TM_NSECUPDT Register Diagram

Table 23-139: EMAC_TM_NSECUPDT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	ADDSUB	Add or Subtract the Time. The EMAC_TM_NSECUPDT.ADDSUB bit, when set, subtracts the time value with the contents of the update registers. When this bit is reset, the time value is added with the contents of the update registers.
30:0 (R/W)	TSSS	Time Stamp Sub Second Initialize/Increment. The value in the EMAC_TM_NSECUPDT.TSSS bit field indicates the time, in nanoseconds, to be initialized or added to or subtracted from the system time nanoseconds.

Time Stamp Target Time Nanoseconds Register

The EMAC_TM_NTGTM register contains the high 32 bits of the target nanoseconds field for comparison to the corresponding system time field.

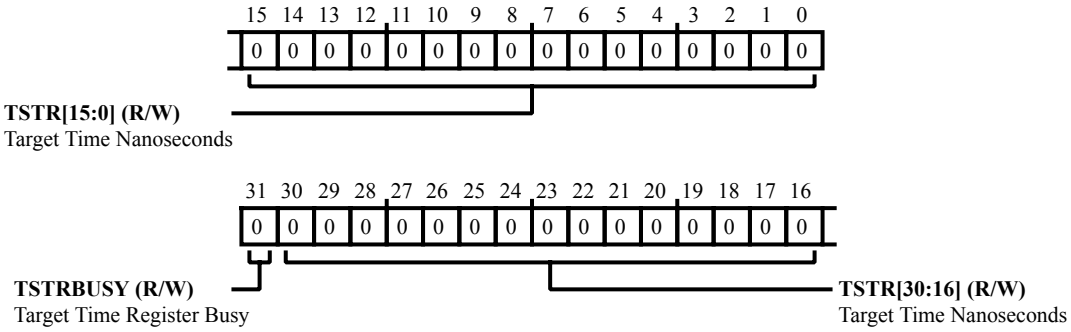

Figure 23-110: EMAC_TM_NTGTM Register Diagram

Table 23-140: EMAC_TM_NTGTM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	TSTRBUSY	Target Time Register Busy. The EMAC_TM_NTGTM.TSTRBUSY bit is set when Flexible PPS is enabled and the EMAC_TM_PPSCTL.PPSCTL field is programmed to 0001, 0010 or 0100. Programming the EMAC_TM_PPSCTL.PPSCTL field to 0001, 0010 or 0100, instructs the core to synchronize the EMAC_TM_TGTM and EMAC_TM_NTGTM registers to the PTP clock domain. The EMAC clears this bit after synchronizing the EMAC_TM_TGTM and EMAC_TM_NTGTM registers to the PTP clock domain The application must not update the EMAC_TM_TGTM and EMAC_TM_NTGTM registers when this bit is read as 1. Otherwise, the synchronization of the previous programmed time gets corrupted.
30:0 (R/W)	TSTR	Target Time Nanoseconds. The EMAC_TM_NTGTM.TSTR bit field stores the time in (signed) nanoseconds. When the value of the time stamp matches the both EMAC_TM_TGTM and EMAC_TM_NTGTM registers, based on the EMAC_TM_PPSCTL.TRGTMODSEL field, the MAC starts or stops the PPS signal output and generates an interrupt (if enabled). This value should not exceed 0x3B9A_C9FF when EMAC_TM_PPSCTL.TRGTMODSEL is set. The actual start or stop time of the PPS signal output may have an error margin up to one unit of sub-second increment value.

PPS Control Register

The EMAC_TM_PPSCTL register controls the interval of PPS output.

When the EMAC_TM_PPSCTL.PPSEN bit is disabled (=0, fixed PPS output), the PPS Frequency Control (PPSCTL0) bits control the behavior of the PPS output signal. The default value of PPSCTRL is 0000 and the PPS output is 1 pulse every second. For other values of PPSCTRL, the PPS output becomes a generated clock. (See bit enumerations for frequencies.) In the binary rollover mode, the PPS output has a duty cycle of 50 percent with these frequencies. In the digital rollover mode, the PPS output frequency is an average number. The actual clock is of different frequency that gets synchronized every second. This behavior is because of the non-linear toggling of the bits in the digital rollover mode in System Time - Nanoseconds Register.

When the EMAC_TM_PPSCTL.PPSEN bit is enabled (=1, flexible PPS output), the PPS Frequency Control bits function as PPSCMD. (See bit enumerations for commands.) Programming these bits with a non-zero value instructs the core to initiate an event. After the command is transferred or synchronized to the PTP clock domain, these bits gets cleared automatically. Software should ensure that these bits are programmed only when they are all-zero.

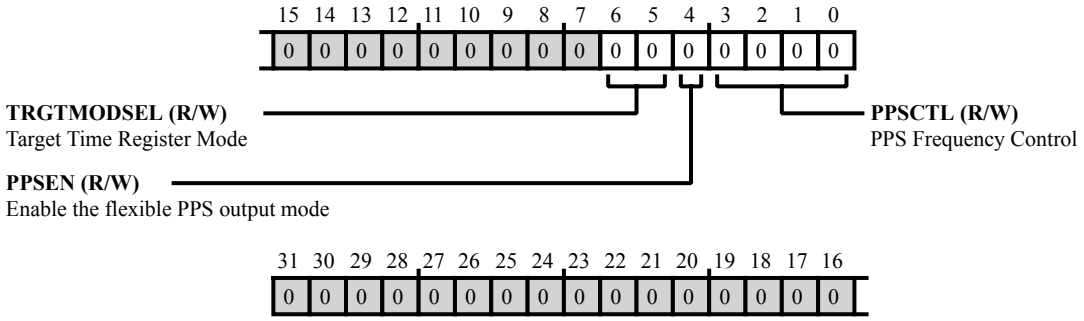

Figure 23-111: EMAC_TM_PPSCTL Register Diagram

Table 23-141: EMAC_TM_PPSCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6:5 (R/W)	TRGTMODSEL	Target Time Register Mode. The EMAC_TM_PPSCTL.TRGTMODSEL bits select the target time register mode.	
		0	Interrupt Only The Target Time registers are programmed only for interrupt event generation.
		1	Reserved
		2	Interrupt and PPS Start/Stop The Target Time registers are programmed for interrupt event and for starting or stopping the PPS output signal generation.
		3	PPS Start/Stop Only The Target Time registers are programmed only for starting or stopping the PPS output signal generation. No interrupt is asserted.
4 (R/W)	PPSEN	Enable the flexible PPS output mode. The EMAC_TM_PPSCTL.PPSEN bit enables PPS operation. When set low, the PPS Frequency Control field controls frequency of Fixed PPS output. When set high, PPS Frequency Control field is used to command Flexible PPS output.	
3:0 (R/W)	PPSCTL	PPS Frequency Control. When the EMAC_TM_PPSCTL.PPSEN bit is disabled (=0, fixed PPS output), the EMAC_TM_PPSCTL.PPSCTL bits control the behavior of the PPS output signal. When the EMAC_TM_PPSCTL.PPSEN bit is enabled (=1, flexible PPS output), the EMAC_TM_PPSCTL.PPSCTL bits function as PPSCMD. (See bit enumerations for PPS output frequency, rollover, and PPS commands.)Programming these bits with a non-zero value instructs the core to initiate an event. After the command is transferred or synchronized to the PTP clock domain, these bits gets cleared automatically. Software should ensure that these bits are programmed only when they are "all-zero". All values not shown in the bit enumerations are reserved. For more information about the EMAC_TM_PPSCTL.PPSCTL bits, see the pulse-per-second functional description.	
		0	CMD=No Command
		1	CMD=START Single; BR=2kHz; DR=1kHz For more info, see register description.
		2	CMD=START Pulse; BR=4kHz; DR=2kHz For more info, see register description.
		3	CMD=Cancel START; BR=8kHz; DR=4kHz For more info, see register description.
		4	CMD=STOP Pulse Time; BR=16kHz; DR=8kHz For more info, see register description.
		5	CMD=STOP Pulse Now For more info, see register description.

Table 23-141: EMAC_TM_PPSCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		6	CMD=Cancel STOP Pulse For more info, see register description.

Time Stamp PPS Interval Register

The EMAC_TM_PPSINTVL register contains the interval value for the time between rising edges (period) of PPS output.

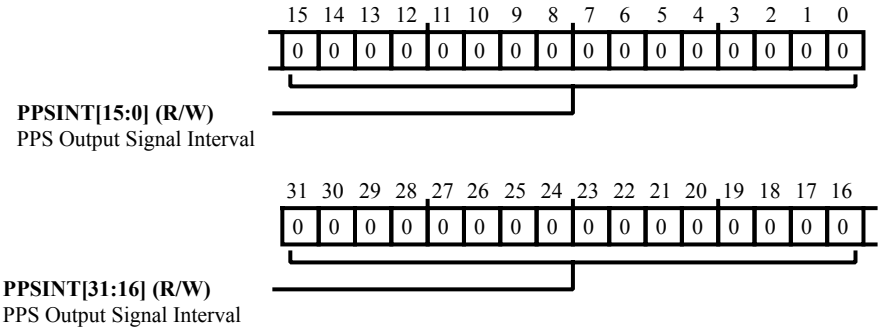

Figure 23-112: EMAC_TM_PPSINTVL Register Diagram

Table 23-142: EMAC_TM_PPSINTVL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	PPSINT	PPS Output Signal Interval. The EMAC_TM_PPSINTVL.PPSINT bits store the interval between the rising edges of PPS signal output in terms of units of sub-second increment value. You need to program one value less than the required interval. For example, if the PTP reference clock is 50 MHz (period of 20ns), and desired interval between rising edges of PPS signal output is 100ns (that is, 5 units of sub-second increment value), then you should program value 4 (5-1) in this register.

PPS Width Register

The EMAC_TM_PPSWIDTH register contains the interval value for the time between a rising and the next falling edge (width) of PPS output.

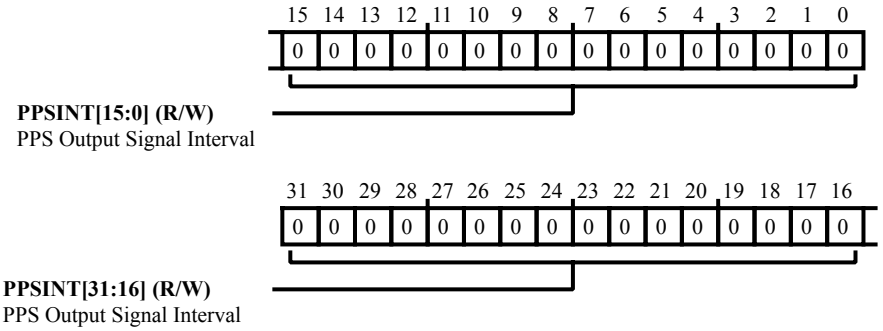

Figure 23-113: EMAC_TM_PPSWIDTH Register Diagram

Table 23-143: EMAC_TM_PPSWIDTH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	PPSINT	PPS Output Signal Interval. The EMAC_TM_PPSWIDTH.PPSINT bits store the interval between a rising edge and the next falling edge (width) of PPS output in terms of units of sub second increment value. The user needs to program one value less than the required interval. For example, if the PTP reference clock is 50 MHz (period of 20ns) and the desired width of the PPS signal output is 60ns (that is, 3 units of sub-second increment value), the user should program value 2 (3-1) in this register.

Time Stamp Low Seconds Register

The EMAC_TM_SEC register contains the lower 32 bits of the seconds field of the system time.

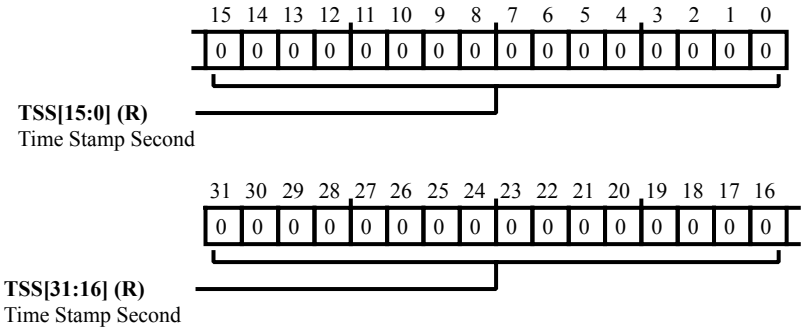

Figure 23-114: EMAC_TM_SEC Register Diagram

Table 23-144: EMAC_TM_SEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	TSS	Time Stamp Second. The value in the EMAC_TM_SEC.TSS bit field indicates the current value in seconds of the System Time maintained by the core.

Time Stamp Seconds Update Register

The EMAC_TM_SECUPDT register contains the low 32 bits to be added to, subtracted from, or written to the seconds field of the system time.

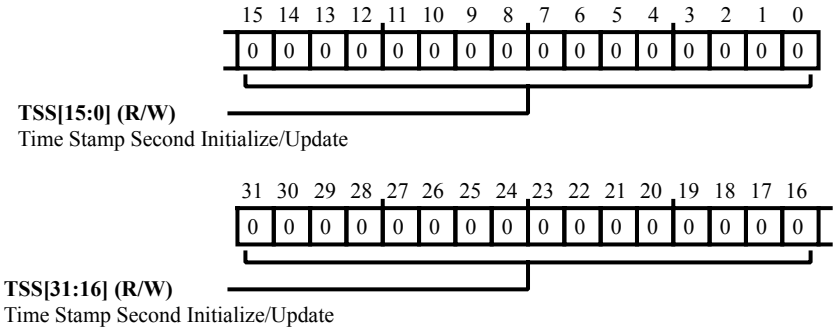

Figure 23-115: EMAC_TM_SECUPDT Register Diagram

Table 23-145: EMAC_TM_SECUPDT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TSS	Time Stamp Second Initialize/Update. The value in the EMAC_TM_SECUPDT.TSS bit field indicates the time, in seconds, to be initialized or added to or subtracted from the system time seconds.

Time Stamp Status Register

The EMAC_TM_STMPSTAT register contains the PTP status.

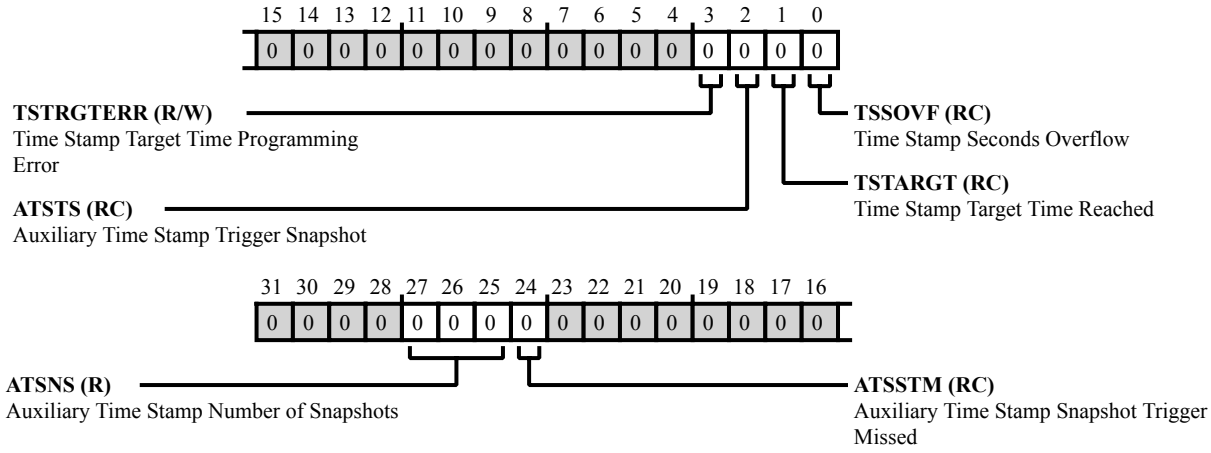

Figure 23-116: EMAC_TM_STMPSTAT Register Diagram

Table 23-146: EMAC_TM_STMPSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:25 (R/NW)	ATSNS	Auxiliary Time Stamp Number of Snapshots. The EMAC_TM_STMPSTAT.ATSNS bits indicate the number of Snapshots available in the FIFO. A value of 4 (100) indicates that the Auxiliary Snapshot FIFO is full. These bits are cleared (to 000) when the Auxiliary snapshot FIFO clear bit is set.
24 (RC/NW)	ATSSTM	Auxiliary Time Stamp Snapshot Trigger Missed. The EMAC_TM_STMPSTAT.ATSSTM bit is set when the Auxiliary time stamp snapshot FIFO is full and external trigger was set. This indicates that the latest snapshot was not stored in the FIFO.
3 (R/W)	TSTRGTERR	Time Stamp Target Time Programming Error. The EMAC_TM_STMPSTAT.TSTRGTERR bit is set when the target time, which is being programmed in the EMAC_TM_SEC and EMAC_TM_NSEC registers, has already elapsed. This bit is cleared when read by the application.
2 (RC/NW)	ATSTS	Auxiliary Time Stamp Trigger Snapshot. The EMAC_TM_STMPSTAT.ATSTS bit is set high when the auxiliary snapshot is written to the FIFO.
1 (RC/NW)	TSTARGT	Time Stamp Target Time Reached. The EMAC_TM_STMPSTAT.TSTARGT bit, when set, indicates the value of system time has reached or passed the value specified in the EMAC_TM_TGTM and EMAC_TM_NTGTM registers.
0	TSSOVF	Time Stamp Seconds Overflow.

Table 23-146: EMAC_TM_STMPSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(RC/NW)		The `EMAC_TM_STMPSTAT.TSSOVF` bit, when set, indicates that the seconds value of the time stamp (when supporting PTP version 2 format) has overflowed beyond 0xFFFF_FFFF.

Time Stamp Sub Second Increment Register

The EMAC_TM_SUBSEC register contains the value by which the system time nano second is incremented.

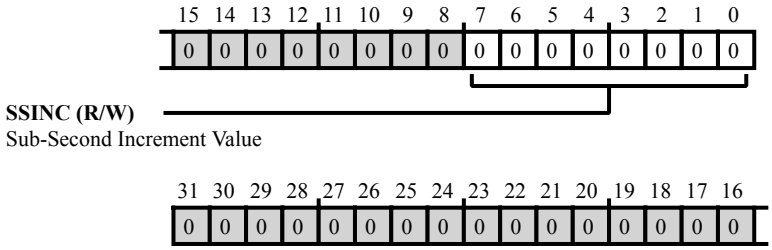

Figure 23-117: EMAC_TM_SUBSEC Register Diagram

Table 23-147: EMAC_TM_SUBSEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	SSINC	Sub-Second Increment Value. The value in the EMAC_TM_SUBSEC.SSINC bits is accumulated every PTP clock cycle with the contents of the nanosecond register. For example, when PTP clock is 50 MHz (period is 20 ns), the processor should program 20 (0x14) when the EMAC_TM_NSEC register has an accuracy of 1 ns (EMAC_TM_CTL.TSCTRLSSR bit is set). When EMAC_TM_CTL.TSCTRLSSR is clear, the EMAC_TM_NSEC register has a resolution of ~0.465ns. In this case, the processor should program a value of 43 (0x2B) that is derived by 20ns/0.465.

Time Stamp Target Time Seconds Register

The EMAC_TM_TGTM register contains the high 32 bits of the target seconds field for comparison to the corresponding system time field.

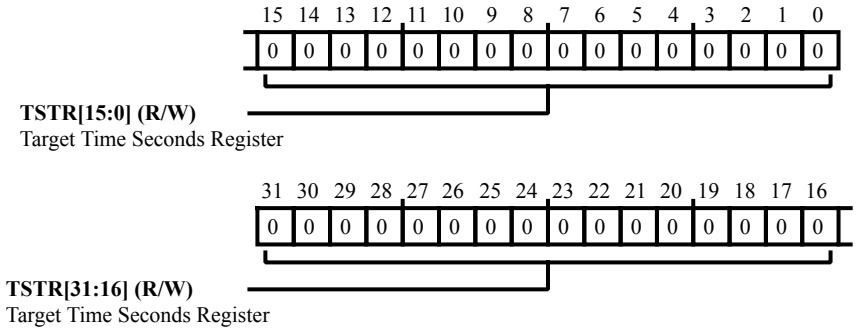

Figure 23-118: EMAC_TM_TGTM Register Diagram

Table 23-148: EMAC_TM_TGTM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TSTR	Target Time Seconds Register. The EMAC_TM_TGTM.TSTR bit field stores the time in seconds. When the time stamp value matches or exceeds both EMAC_TM_TGTM and EMAC_TM_NTGTM registers, based on the selection in the EMAC_TM_PPSCTL.TRGTMODSEL bits, the MAC starts or stops the PPS signal output and generates an interrupt (if enabled).

Tx 1024- to Max-Byte Frames (Good/Bad) Register

The EMAC_TX1024TOMAX_GB register contains the count of the number of good and bad frames transmitted with length between 1024 and maxsize (inclusive) bytes, exclusive of preamble and retried frames.

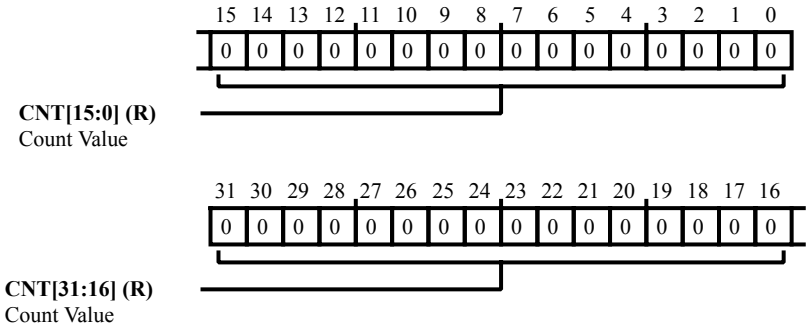

Figure 23-119: EMAC_TX1024TOMAX_GB Register Diagram

Table 23-149: EMAC_TX1024TOMAX_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx 128- to 255-Byte Frames (Good/Bad) Register

The EMAC_TX128TO255_GB register contains the count of the number of good and bad frames transmitted with length between 128 and 255 (inclusive) bytes, exclusive of preamble and retried frames.

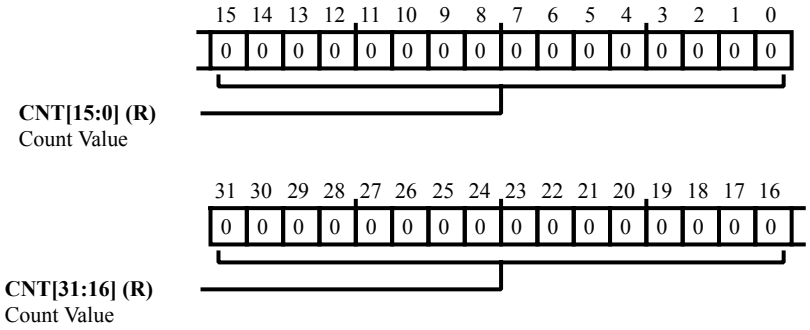

Figure 23-120: EMAC_TX128TO255_GB Register Diagram

Table 23-150: EMAC_TX128TO255_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx 256- to 511-Byte Frames (Good/Bad) Register

The EMAC_TX256TO511_GB register contains the count of the number of good and bad frames transmitted with length between 256 and 511 (inclusive) bytes, exclusive of preamble and retried frames.

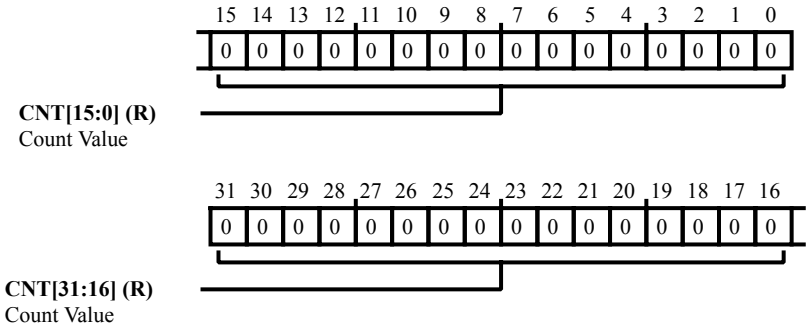

Figure 23-121: EMAC_TX256TO511_GB Register Diagram

Table 23-151: EMAC_TX256TO511_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx 512- to 1023-Byte Frames (Good/Bad) Register

The EMAC_TX512TO1023_GB register contains the count of the number of good and bad frames transmitted with length between 512 and 1023 (inclusive) bytes, exclusive of preamble and retried frames.

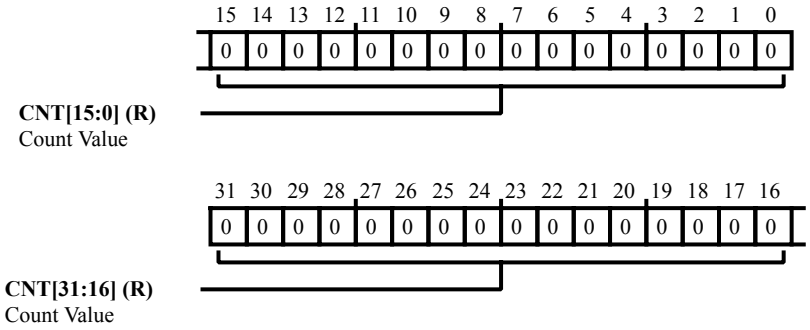

Figure 23-122: EMAC_TX512TO1023_GB Register Diagram

Table 23-152: EMAC_TX512TO1023_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx 64-Byte Frames (Good/Bad) Register

The EMAC_TX64_GB register contains the count of the number of good and bad frames transmitted with length 64 bytes, exclusive of preamble and retried frames.

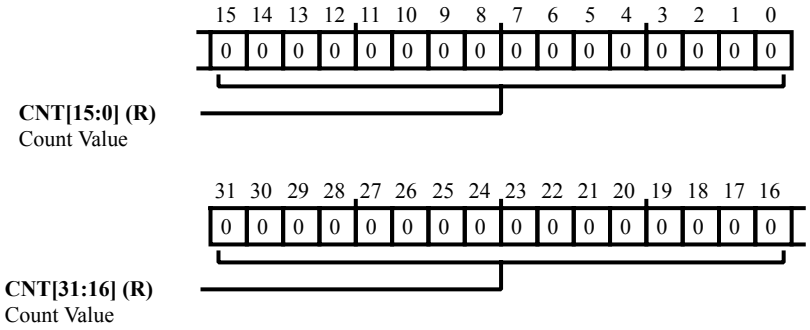

Figure 23-123: EMAC_TX64_GB Register Diagram

Table 23-153: EMAC_TX64_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx 65- to 127-Byte Frames (Good/Bad) Register

The EMAC_TX65TO127_GB register contains the count of the number of good and bad frames transmitted with length between 65 and 127 (inclusive) bytes, exclusive of preamble and retried frames.

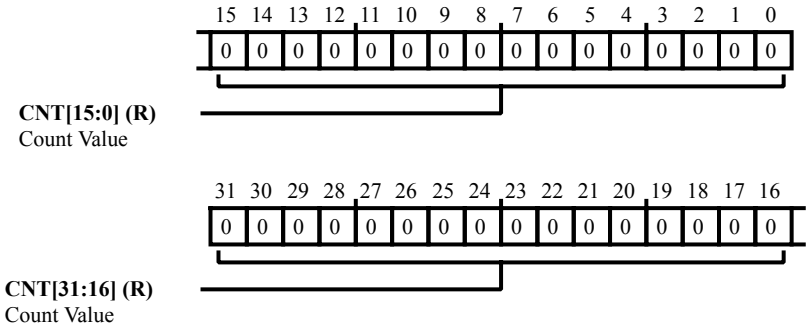

Figure 23-124: EMAC_TX65TO127_GB Register Diagram

Table 23-154: EMAC_TX65TO127_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Broadcast Frames (Good) Register

The EMAC_TXBCASTFRM_G register contains the count of the number of good broadcast frames transmitted.

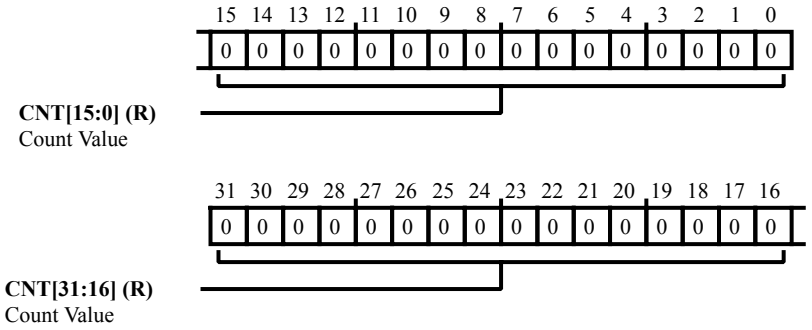

Figure 23-125: EMAC_TXBCASTFRM_G Register Diagram

Table 23-155: EMAC_TXBCASTFRM_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Broadcast Frames (Good/Bad) Register

The EMAC_TXBCASTFRM_GB register contains the count of the number of good and bad broadcast frames transmitted.

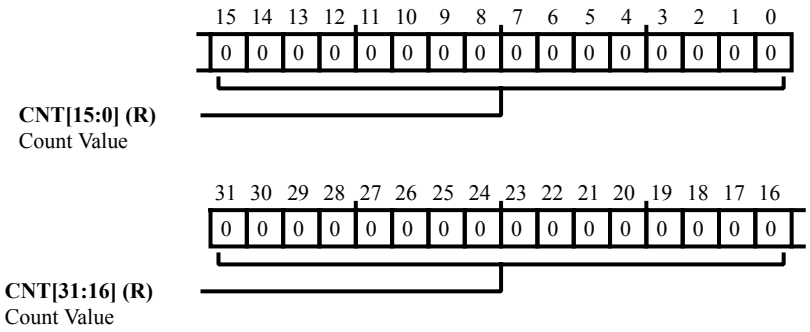

Figure 23-126: EMAC_TXBCASTFRM_GB Register Diagram

Table 23-156: EMAC_TXBCASTFRM_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Carrier Error Register

The EMAC_TXCARR_ERR register contains a count of the number of frames aborted due to carrier sense error (no carrier or loss of carrier).

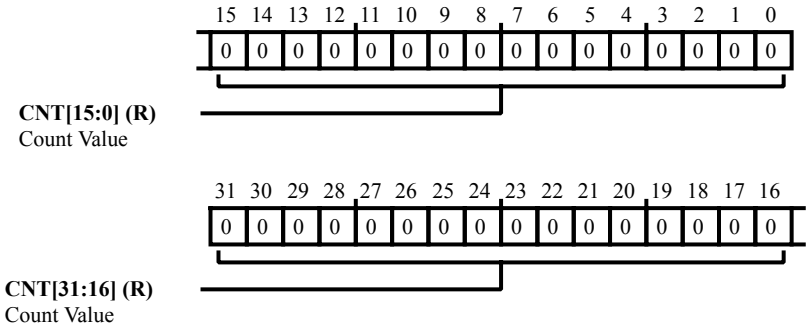

Figure 23-127: EMAC_TXCARR_ERR Register Diagram

Table 23-157: EMAC_TXCARR_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Deferred Register

The EMAC_TXDEFERRED register contains a count of the number of successfully transmitted frames after a deferral in Half-duplex mode.

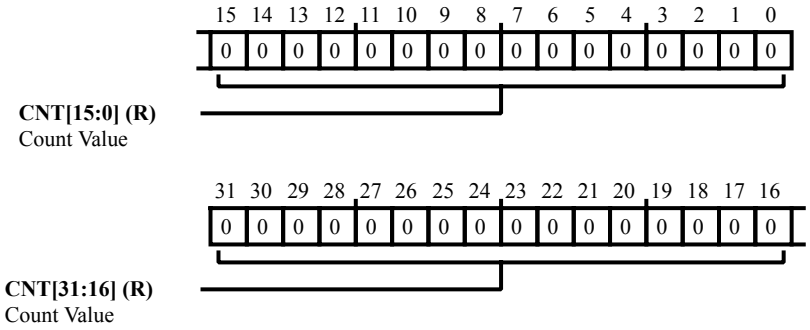

Figure 23-128: EMAC_TXDEFERRED Register Diagram

Table 23-158: EMAC_TXDEFERRED Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Excess Collision Register

The EMAC_TXEXCESSCOL register contains a count of the number of frames aborted due to excessive (16) collision errors.

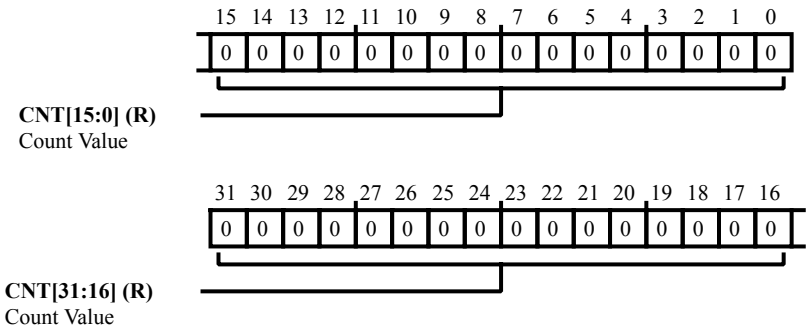

Figure 23-129: EMAC_TXEXCESSCOL Register Diagram

Table 23-159: EMAC_TXEXCESSCOL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Excess Deferral Register

The EMAC_TXEXCESSDEF register contains a count of the number of frames aborted due to excessive deferral error (deferred for more than two max-sized frame times).

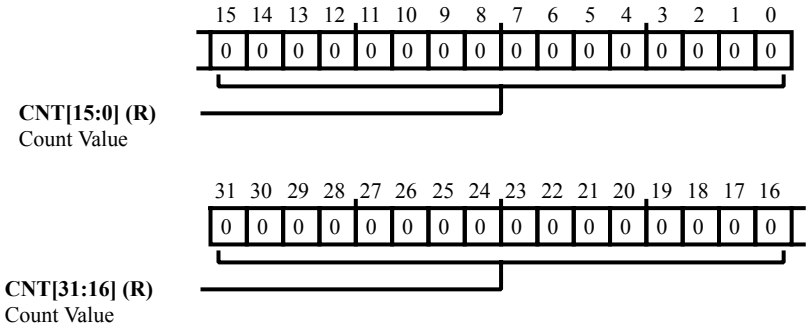

Figure 23-130: EMAC_TXEXCESSDEF Register Diagram

Table 23-160: EMAC_TXEXCESSDEF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Frame Count (Good) Register

The EMAC_TXFRMCNT_G register contains a count of the number of good frames transmitted.

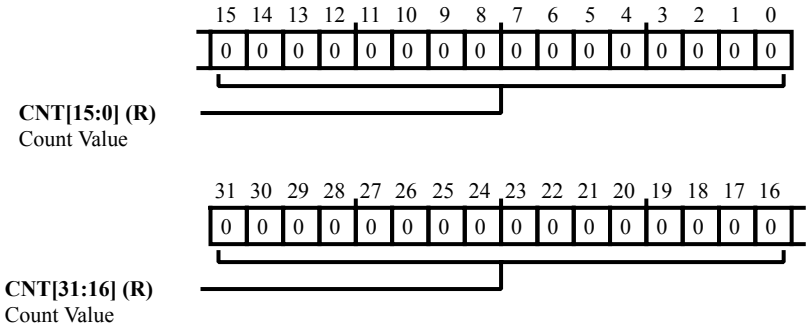

Figure 23-131: EMAC_TXFRMCNT_G Register Diagram

Table 23-161: EMAC_TXFRMCNT_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Frame Count (Good/Bad) Register

The EMAC_TXFRMCNT_GB register contains the count of the number of good and bad frames transmitted, exclusive of retried frames.

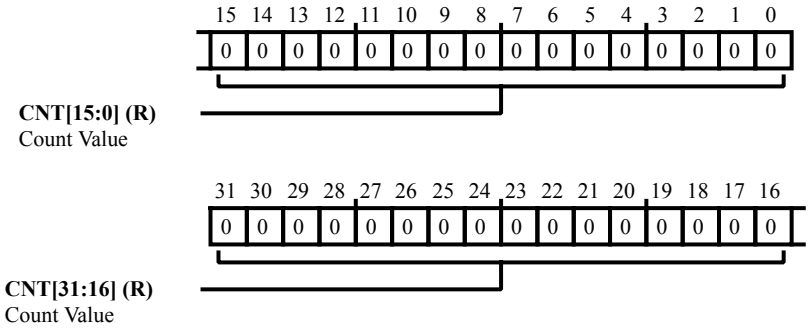

Figure 23-132: EMAC_TXFRMCNT_GB Register Diagram

Table 23-162: EMAC_TXFRMCNT_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Late Collision Register

The EMAC_TXLATECOL register contains a count of the number of frames aborted due to late collision error.

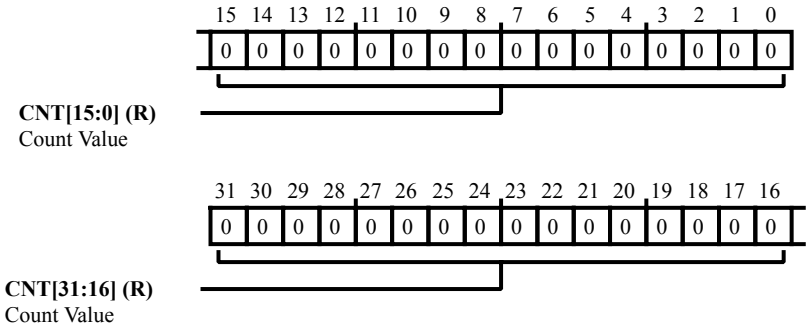

Figure 23-133: EMAC_TXLATECOL Register Diagram

Table 23-163: EMAC_TXLATECOL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Multicast Frames (Good) Register

The EMAC_TXMCASTFRM_G register contains the count of the number of good multicast frames transmitted.

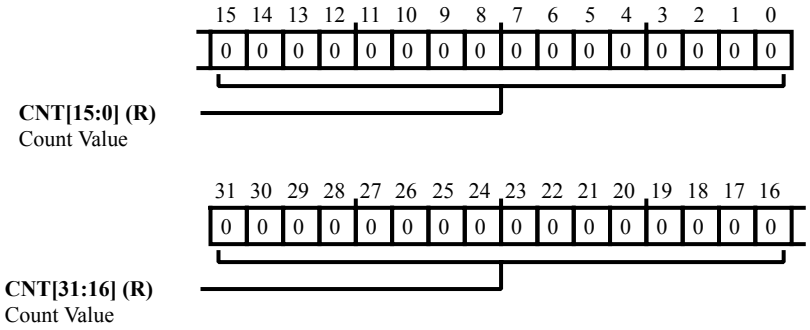

Figure 23-134: EMAC_TXMCASTFRM_G Register Diagram

Table 23-164: EMAC_TXMCASTFRM_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Multicast Frames (Good/Bad) Register

The EMAC_TXMCASTFRM_GB register contains the count of the number of good and bad multicast frames transmitted.

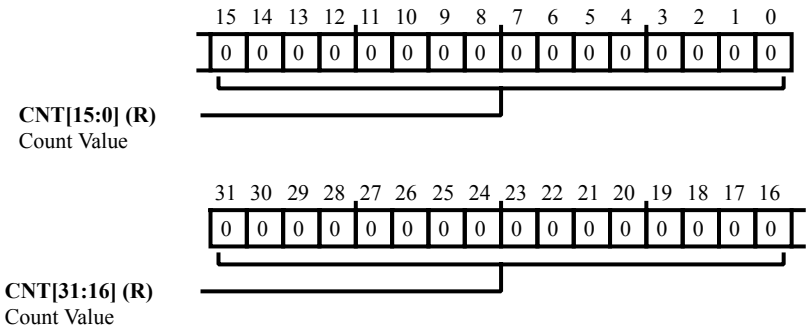

Figure 23-135: EMAC_TXMCASTFRM_GB Register Diagram

Table 23-165: EMAC_TXMCASTFRM_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Multiple Collision (Good) Register

The EMAC_TXMULTCOL_G register contains a count of the number of successfully transmitted frames after more than a single collision in Half-duplex mode.

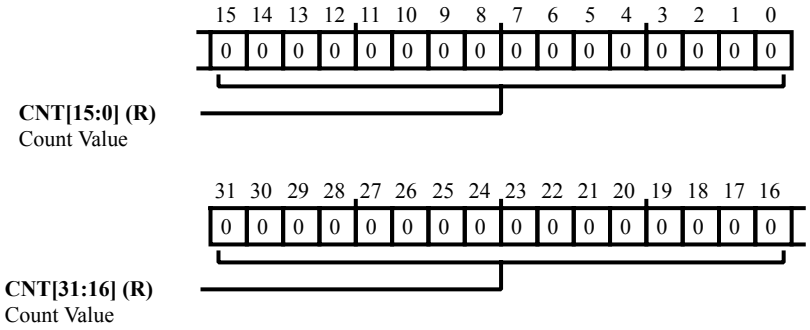

Figure 23-136: EMAC_TXMULTCOL_G Register Diagram

Table 23-166: EMAC_TXMULTCOL_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Octet Count (Good) Register

The EMAC_TXOCTCNT_G register contains a count of the number of bytes transmitted, exclusive of preamble, in good frames only.

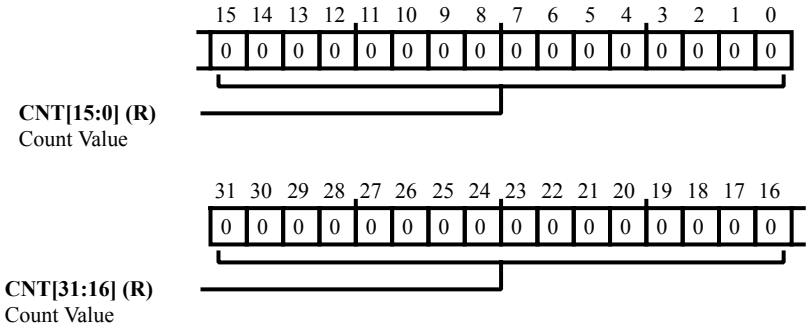

Figure 23-137: EMAC_TXOCTCNT_G Register Diagram

Table 23-167: EMAC_TXOCTCNT_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx OCT Count (Good/Bad) Register

The EMAC_TXOCTCNT_GB register contains the count of the number of bytes transmitted, exclusive of the preamble and retried bytes, in good and bad frames.

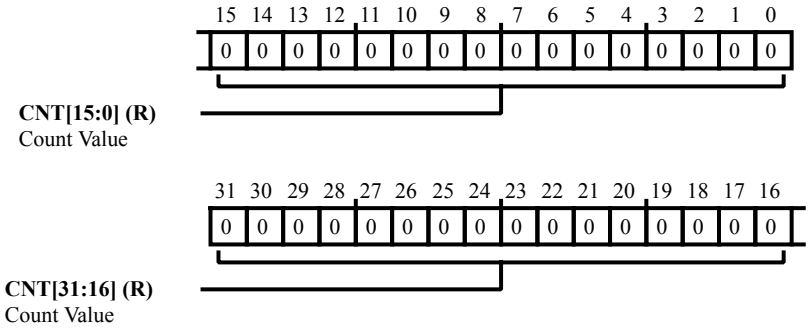

Figure 23-138: EMAC_TXOCTCNT_GB Register Diagram

Table 23-168: EMAC_TXOCTCNT_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Pause Frame Register

The EMAC_TXPAUSEFRM register contains a count of the number of good PAUSE frames transmitted.

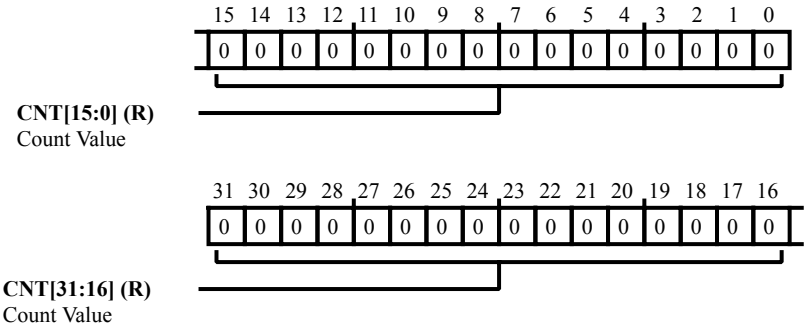

Figure 23-139: EMAC_TXPAUSEFRM Register Diagram

Table 23-169: EMAC_TXPAUSEFRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Single Collision (Good) Register

The EMAC_TXSNGCOL_G register contains a count of the number of successfully transmitted frames after a single collision in Half-duplex mode.

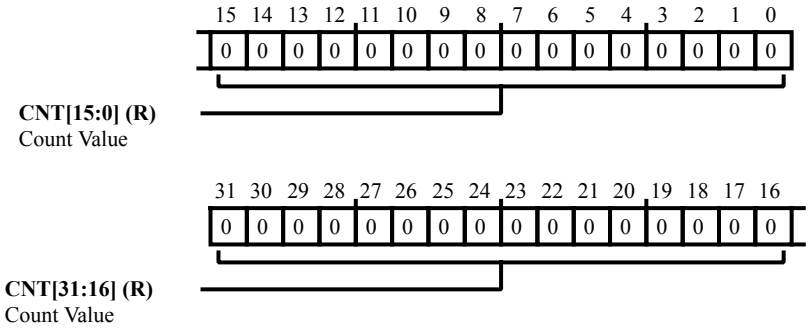

Figure 23-140: EMAC_TXSNGCOL_G Register Diagram

Table 23-170: EMAC_TXSNGCOL_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Unicast Frames (Good/Bad) Register

The EMAC_TXUCASTFRM_GB register contains the count of the number of good and bad unicast frames transmitted.

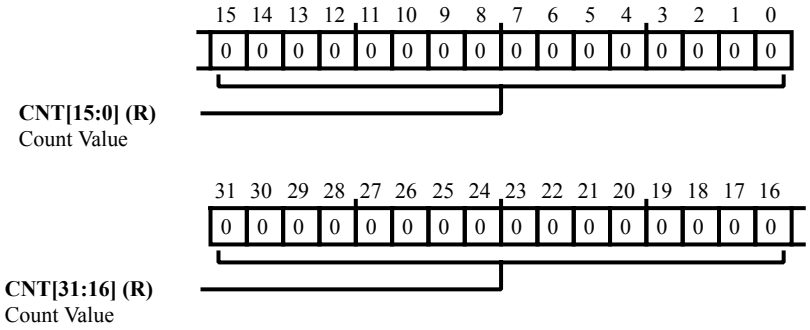

Figure 23-141: EMAC_TXUCASTFRM_GB Register Diagram

Table 23-171: EMAC_TXUCASTFRM_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Underflow Error Register

The EMAC_TXUNDR_ERR register contains a count of the number of frames aborted due to frame underflow error.

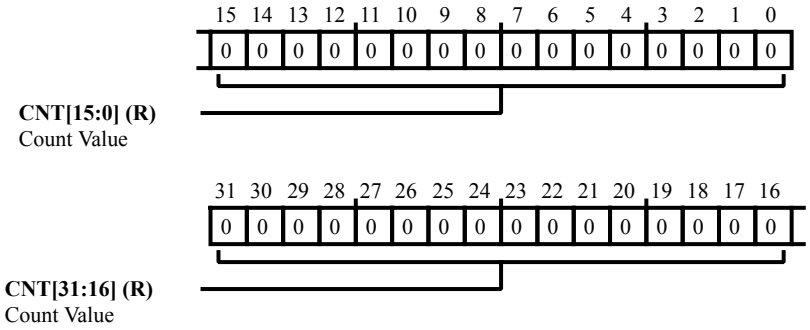

Figure 23-142: EMAC_TXUNDR_ERR Register Diagram

Table 23-172: EMAC_TXUNDR_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx VLAN Frames (Good) Register

The EMAC_TXVLANFRM_G register contains a count of the number of good VLAN frames transmitted, exclusive of retried frames.

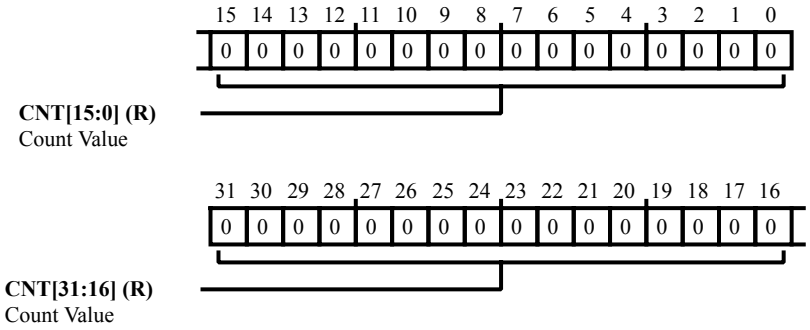

Figure 23-143: EMAC_TXVLANFRM_G Register Diagram

Table 23-173: EMAC_TXVLANFRM_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

VLAN Tag Register

The EMAC_VLANTAG register contains the VLAN tag.

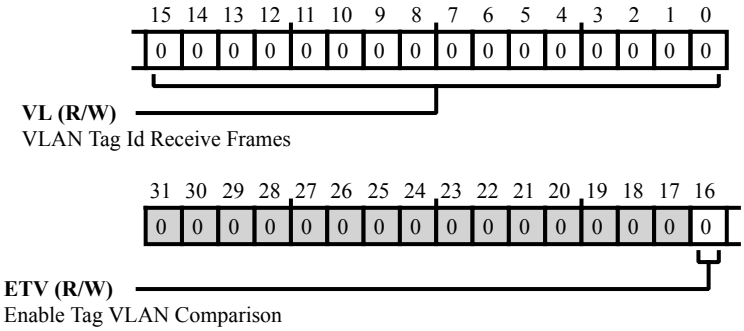

Figure 23-144: EMAC_VLANTAG Register Diagram

Table 23-174: EMAC_VLANTAG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	ETV	Enable Tag VLAN Comparison. The EMAC_VLANTAG.ETV bit, when set, directs the EMAC to use a 12-bit VLAN identifier, rather than the complete 16-bit VLAN tag, for comparison and filtering. Bits[11:0] of the VLAN tag are compared with the corresponding field in the received VLAN-tagged frame. When this bit is reset, all 16 bits of the received VLAN frame's fifteenth and sixteenth bytes are used for comparison.
15:0 (R/W)	VL	VLAN Tag Id Receive Frames. The EMAC_VLANTAG.VL bits contain the 802.1Q VLAN tag to identify VLAN frames, and is compared to the fifteenth and sixteenth bytes of the frames being received for VLAN frames. Bits[15:13] are the User Priority, Bit[12] is the Canonical Format Indicator (CFI) and bits[11:0] are the VLAN tag's VLAN Identifier (VID) field. When the ETV bit is set, only the VID (Bits[11:0]) is used for comparison. If VL (VL[11:0] if ETV is set) is all zeros, the MAC does not check the fifteenth and sixteenth bytes for VLAN tag comparison, and declares all frames with a Type field value of 0x8100 to be VLAN frames.

24 Serial Peripheral Interface (SPI)

The serial peripheral interface is an industry-standard synchronous serial link that supports communication with multiple SPI-compatible devices. The baseline SPI peripheral is a synchronous, four-wire interface consisting of two data pins, one device select pin, and a gated clock pin. The two data pins allow full-duplex operation to other SPI-compatible devices. Two extra (optional) data pins are provided for SPI2 to support quad SPI operation. Enhanced modes of operation such as flow control, fast mode, and dual-I/O mode (DIOM) are also supported. In addition, a direct memory access (DMA) mode allows for transferring several words with minimal CPU interaction.

With a range of configurable options, the SPI ports provide a glueless hardware interface with other SPI-compatible devices in master mode, slave mode, and multimaster environments. The SPI peripheral includes programmable baud rates, clock phase, and clock polarity. The peripheral can operate in a multimaster environment by interfacing with several other devices, acting as either a master device or a slave device. In a multimaster environment, the SPI peripheral uses open-drain outputs to avoid data bus contention. The flow control features enable slow slave devices to interface with fast master devices by providing an SPI ready pin which flexibly controls the transfers.

SPI Features

The SPI module supports the following features:

- Full-duplex, synchronous serial interface
- Supports 8, 16-bit and 32-bit word sizes
- Programmable baud rate, clock phase, and polarity
- Programmable inter-frame latency
- Flow control
- Support for Fast and DIOM modes
- Quad and memory-mapped modes are supported by SPI2 only
- Independent receive and transmit DMA channels
- Burst transfer mode for non-DMA write accesses

SPI Functional Description

This section provides information on the function of the SPI module.

Shift register functionality

The SPI is essentially a shift register that serially transmits and receives data bits to or from other SPI devices. During an SPI transfer, data is simultaneously transmitted (shifted out serially) and received (shifted in serially). A serial clock line synchronizes shifting and sampling of the information on the two serial data lines.

Master slave functionality

During a data transfer, one SPI system acts as the link master which controls the data flow. The other system acts as the slave, which has data shifted into and out of it by the master. Different devices can take turn being masters, and one master can simultaneously shift data into multiple slaves (broadcast mode). However, only one slave can drive its output to write data back to the master at any given time. This rule must be enforced in the broadcast mode. Several slaves can be selected to receive data from the master in this mode. But only one slave can be enabled to send data back to the master.

Enhanced operating modes

SPI supports enhanced modes of operation like fast mode, DIOM, and Quad-SPI, and optional flow control. In fast mode, received data is sampled on the transmit edge instead of the standard receive edge, thus enabling a full-cycle path for the received data. In DIOM, both MOSI and MISO are configured as input or output pins, and 2 bits are shifted in or out on each receive or transmit edge. In Quad-SPI mode, SPI_D3:0 are configured as input or output pins and 4 bits are shifted in or out on each receive or transmit edge. A slower slave can use flow control to stall a faster master device.

Single and multi-master use

The SPI can be used in a single master as well as multi-master environment. The `SPI_MOSI`, `SPI_MISO`, and the `SPI_CLK` signals are all tied together in both configurations. SPI transmission and reception can be enabled simultaneously or individually, depending on `SPI_RXCTL` and `SPI_TXCTL` settings. In broadcast mode, several slaves can be enabled to receive, but only one slave must be in transmit mode and driving the `SPI_MISO` line.

ADSP-CM40x SPI Register List

The Serial Peripheral Interface SPI provides a full-duplex, synchronous serial interface, which supports both master/slave modes and multi-master environments. The SPI's baud rate and clock phase/polarities are programmable, and it has integrated DMA channels for both transmit and receive data streams. A set of registers governs SPI operations. For more information on SPI functionality, see the SPI register descriptions.

Table 24-1: ADSP-CM40x SPI Register List

Name	Description
SPI_CLK	Clock Rate Register
SPI_CTL	Control Register
SPI_DLY	Delay Register
SPI_ILAT	Masked Interrupt Condition Register
SPI_ILAT_CLR	Masked Interrupt Clear Register
SPI_IMSK	Interrupt Mask Register
SPI_IMSK_CLR	Interrupt Mask Clear Register
SPI_IMSK_SET	Interrupt Mask Set Register
SPI_MMRDH	Memory Mapped Read Header
SPI_MMTOP	SPI Memory Top Address
SPI_RFIFO	Receive FIFO Data Register
SPI_RWC	Received Word Count Register
SPI_RWCR	Received Word Count Reload Register
SPI_RXCTL	Receive Control Register
SPI_SLVSEL	Slave Select Register
SPI_STAT	Status Register
SPI_TFIFO	Transmit FIFO Data Register
SPI_TWC	Transmitted Word Count Register
SPI_TWCR	Transmitted Word Count Reload Register
SPI_TXCTL	Transmit Control Register

ADSP-CM40x SPI Interrupt List

Table 24-2: ADSP-CM40x SPI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
25	SPI0_TXDMA_ERR	SPI0 DMA Channel 2 Error	Level	
26	SPI0_RXDMA_ERR	SPI0 DMA Channel 3 Error	Level	
31	SPI1_TXDMA_ERR	SPI1 DMA Channel 8 Error	Level	
32	SPI1_RXDMA_ERR	SPI1 DMA Channel 9 Error	Level	
41	SPI0_ERR	SPI0 Error	Level	
42	SPI1_ERR	SPI1 Error	Level	
43	SPI2_ERR	SPI2 Error	Level	

Table 24-2: ADSP-CM40x SPI Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
71	SPI0_STAT	SPI0 Status	Level	
72	SPI0_TXDMA	SPI0 TX DMA Channel Transfer Complete	Level	2
73	SPI0_RXDMA	SPI0 RX DMA Channel Transfer Complete	Level	3
81	SPI1_STAT	SPI1 Status	Level	
82	SPI1_TXDMA	SPI1 TX DMA Channel Transfer Complete	Level	8
83	SPI1_RXDMA	SPI1 RX DMA Channel Transfer Complete	Level	9
84	SPI2_TXDMA	SPI2 TX Channel (non-DMA) Transfer Complete	Level	
85	SPI2_RXDMA	SPI2 RX Channel (non-DMA) Transfer Complete	Level	
98	SPI2_STAT	SPI2 Status	Level	

SPI Block Diagram

The ***SPI Controller Block Diagram*** illustrates the block diagram of the SPI module. The module is comprised of three primary parts:

- SPI core contains the receive and transmit FIFOs and their associated shift registers
- Control blocks contain the synchronizer and logic to control the data flow through the data pipelines
- Register block

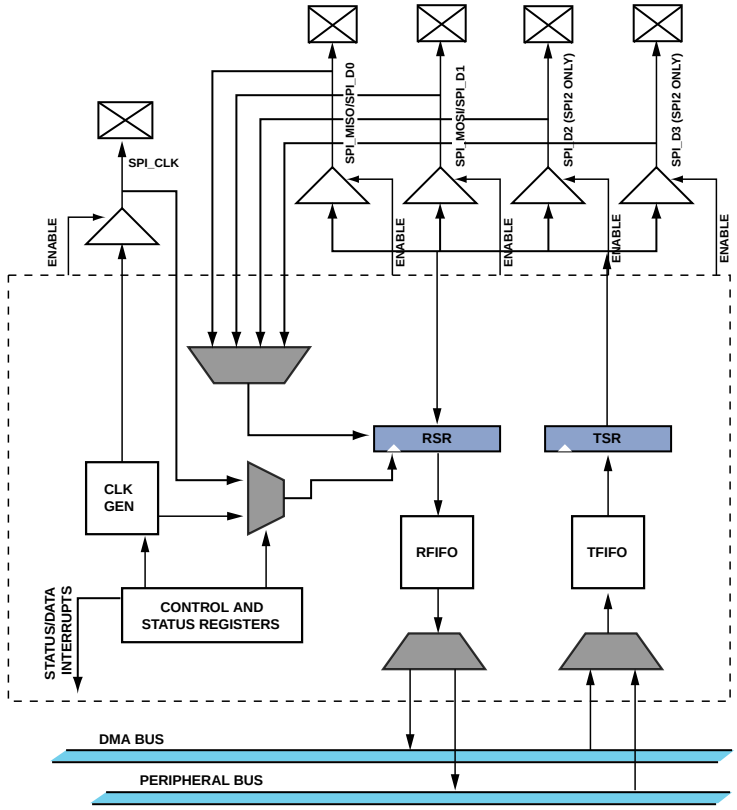

Figure 24-1: SPI Controller Block Diagram, Quad Mode

Transfer Protocol

The SPI module implements two channels that are independent of each other. The SPI module uses the SPI_RXCTL and SPI_TXCTL dedicated control registers to control these channels. Except in dual and quad modes, SPI can enable and use both channels simultaneously.

The SPI protocol supports four different combinations of serial clock phase and polarity. These combinations are selected through the SPI_CTL.CPOL and SPI_CTL.CPHA bits.

The ***SPI Transfer Protocol*** figures demonstrate the two basic transfer formats as defined by the CPHA bit. Two waveforms are shown for SPI_CLK; one for SPI_CTL.CPOL=0 and the other for SPI_CTL.CPOL=1. The diagrams can be interpreted as master or slave timing diagrams since the SPI_CLK, SPI_MISO, and SPI_MOSI pins are directly connected between the master and the slave. The SPI_MISO signal is the output from the slave (slave transmission), and the SPI_MOSI signal is the output from the master (master transmission). The master generates the SPI_CLK signal. The SPI_SS signal is the slave device select input to the slave from the master. The diagrams represent an 8-bit transfer (SPI_CTL.SIZE=0) with the MSB first (SPI_CTL.LSBF=0). Any combination of the SPI_CTL.SIZE and SPI_CTL.LSBF bits is permissible. For example, a 16-bit transfer with the LSB first is another possible configuration.

The clock polarity and the clock phase could be identical for the master device and the slave device involved in the communication link. The transfer format from the master can be changed between transfers to adjust for various requirements of a slave device.

The SPI module uses the SPI_CTL.ASSEL bit to determine when the SPI hardware or software control the SPI_SS line. When SPI_CTL.ASSEL=1, the slave select line must be set to the polarity set in the SPI_CTL.SELST field between each serial transfer. The actual behavior of SPI_SS also depends on the parameters programmed into the SPI_DLY register. The SPI hardware logic automatically controls this functionality. When SPI_CTL.ASSEL=0, SPI_SS can either remain active between successive transfers or be inactive. The software must control this activity through manipulation of the SPI_SLVSEL register.

The ***SPI Transfer Protocol*** pair of figures illustrates the case when SPI_CTL.ASSEL = 1 and the $\overline{\text{SPI\_SS}}$ line is inactive between frames. If ASSEL = 0, the SPI_SS line can remain active between frames; however, the first bit is only driven when an active transition of SPI_CLK occurs.

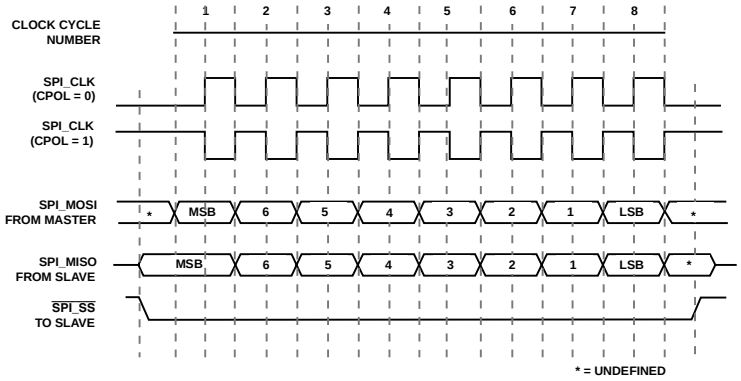

Figure 24-2: SPI Transfer Protocol for CPHA=0

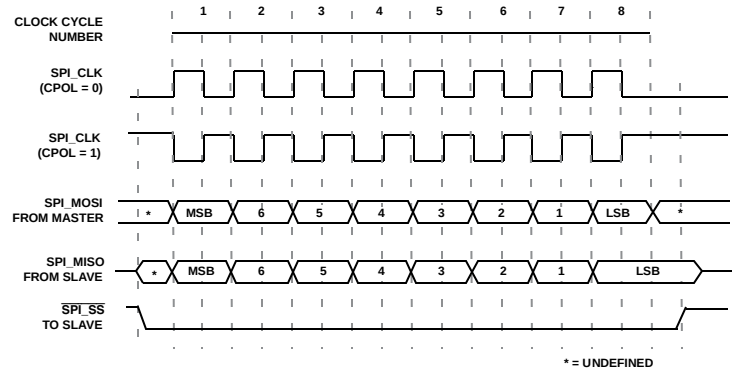

Figure 24-3: SPI Transfer Protocol for CPHA=1

Clock Considerations

The SPI_CLK signal is a gated clock that is only active during data transfers, for the time of the transferred word. In normal mode, the number of active edges is equal to the number of bits to be transmitted or received. In dual-I/O mode, it is half of the number of bits to be transmitted or received, and in quad-SPI mode it is one-fourth of the number. The clock rate can be as high as the SCLK rate, and both even and odd dividers from SCLK are supported. For master devices, the SPI uses the SPI_CLK register value to determine the clock rate, whereas this value is ignored for slave devices.

When the SPI controller is a master, SPI_CLK is an output signal. Conversely, when the SPI controller is a slave, SPI_CLK is an input signal. Slave devices ignore the SPI clock when the slave select input is driven inactive. The SPI uses the SPI_CLK signal to shift out and shift in the data driven onto the SPI_MISO and SPI_MOSI lines. The data is always shifted out on one edge of the clock (the active edge) and sampled on the opposite edge of the clock (the sampling edge). Clock polarity and clock phase relative to data are programmable through the SPI_CTL register and define the transfer format.

Controlling Delay Between Frames

The *SPI Timing with Lead and Lag Programming (Independent of SPI_CTL.CPHA Setting)* figure illustrates SPI timing using the SPI_DLY.LEADX and SPI_DLY.LAGX programming. The SPI uses the SPI_DLY.LAGX bits to control the timing between the slave select (SPI_SS) signal assertion and the first SPI_CLK edge. The SPI uses the SPI_DLY.LEADX bits to control the timing between the last SPI_CLK edge and deassertion of the SPI_SS signal. The lead and lag timing can be extended by a 1 SPI_CLK duration to ease timing restrictions on the slave device.

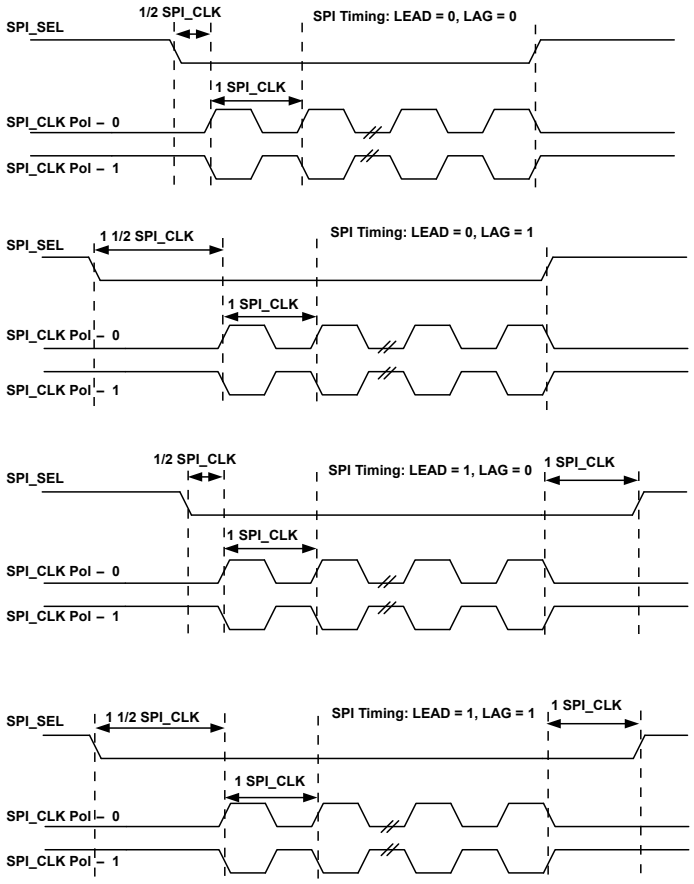

Figure 24-4: SPI Timing with Lead and Lag Programming (Independent of SPI_CTL.CPHA Setting)

The *SPI Timing with SPI_DLY.STOP Programming (Independent of SPI_CTL.CPHA Setting)* figure illustrates SPI timing with STOP programming. The SPI module uses this timing to insert multiples of SPI_CLK period delays between transfers. The SPI_SS line is deasserted for the duration specified in the SPI_DLY.STOP bit field, assuming the SPI_CTL.SELST bit is configured for de-assertion between transfers.

If the SPI_DLY.STOP bit =0, the master operates in a *continuous mode*. This mode causes an immediate start of the second word after the last bit is transferred from the first word. During this mode of operation, the slave select line is continuously asserted.

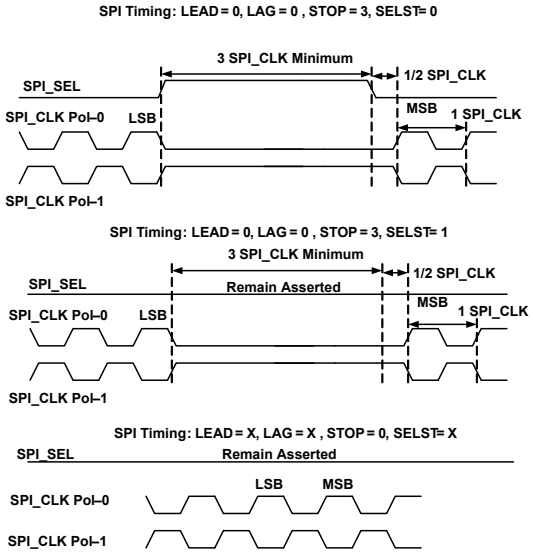

Figure 24-5: SPI Timing with SPI_DLY.STOP Programming (Independent of SPI_CTL.CPHA Setting)

When the SPI_DLY.STOP bit =0 and initial conditions for a transfer are not met, the interface pauses before the next transfer. During this pause, the SPI uses the SPI_CTL.SELST bit to determine the state of the slave select pin. The SPI uses the SPI_DLY.LEADX and SPI_DLY.LAGX bits to determine the timing between SPI_CLK edges and the slave select line.

Flow Control

In master mode, the slave device must drive the SPI_RDY pin. The pin acts as an input signal. The slave can deassert the SPI_RDY pin to stop the master from initiating any new transfer. If SPI_RDY is deasserted in the middle of a transfer, the current transfer continues, and the next transfer will not start unless the slave asserts the SPI_RDY signal. Whenever the slave deasserts SPI_RDY and stalls the master, the SPI controller goes into a waiting state, and the SPI_STAT.FCS bit is set. When the slave asserts SPI_RDY, the SPI controller resumes operation, and the SPI_STAT.FCS bit is cleared.

In slave mode, the SPI_RDY pin acts as an output signal. Flow control can be configured on either the TX channel or the RX channel. The SPI uses the SPI_CTL.FCCH bit to control this configuration. If flow control is configured on the TX channel, as the SPI_TFIFO status nears the empty condition, the SPI_RDY pin is deasserted. If flow control is configured on the RX channel, as the SPI_RFIFO status nears the full condition, the SPI_RDY pin is deasserted. The SPI uses the SPI_CTL.FCWM bits to control the FIFO status at which SPI_RDY de-assertion takes place. Flow control in slave mode is purely based on the FIFO status and does not depend on the word counters. End.

The ***SPI Flow Control Timing in Master Mode*** figure illustrates this timing.

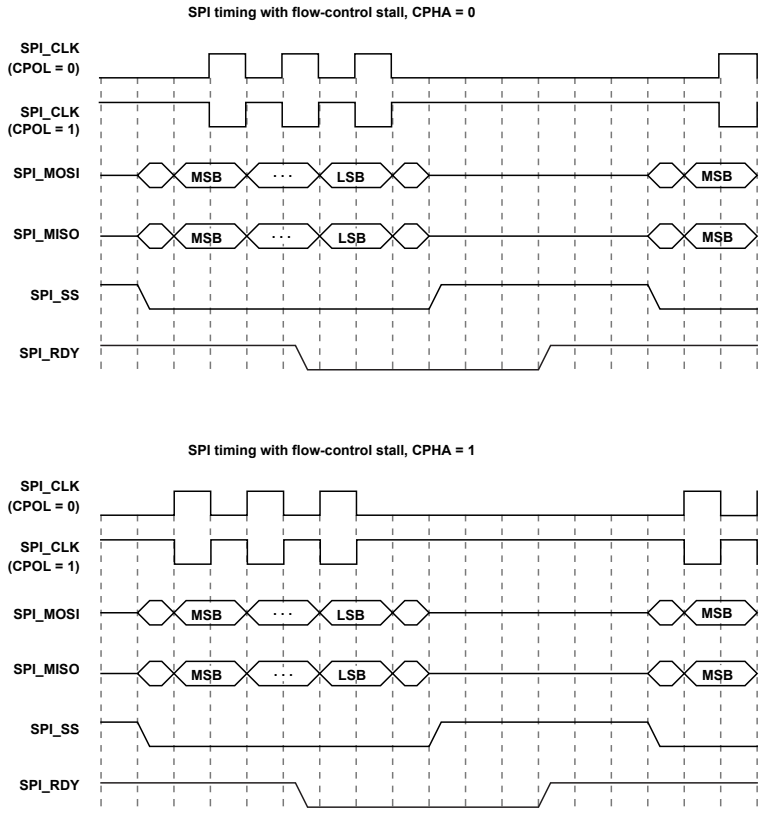

Figure 24-6: SPI Flow Control Timing in Master Mode.

Slave Select Operation

If the SPI is in slave mode, $\overline{\texttt{SPI\_SS}}$ acts as the slave select input. When SPI is enabled as a master, $\overline{\texttt{SPI\_SS}}$ can serve as an error detection input for the SPI in a multi-master environment. The SPI_CTL.PSSE bit enables this feature. When SPI_CTL.PSSE=1, the $\overline{\texttt{SPI\_SS}}$ input is the master mode error input. Otherwise, $\overline{\texttt{SPI\_SS}}$ is ignored.

The $\overline{\texttt{SPI\_SS}}$ signal is an active-low signal. The master asserts the signal during the transfer. The signal can be deasserted or remain asserted between transfers. When $\overline{\texttt{SPI\_SS}}$ is deasserted, SPI_CLK and inputs are ignored, and outputs are three-stated.

The slave select bits (SPI_SLVSEL.SSE1 – SPI_SLVSEL.SSEL7) are used in a multiple-slave SPI environment. For example, if there are eight SPI devices in the system including a processor master, the master processor can support the SPI mode transactions across the other seven devices. This configuration requires only one master processor in this multi-slave environment.

For example, assume that the SPI of the processor is the master. The SPI_SLVSEL.SSE1 – SPI_SLVSEL.SSEL7 bits on the processor can be connected to the slave select pin of each slave device. In this configuration, the slave select bits can be used in three ways. In cases 1 and 2, the processor is the master and the seven microcontrollers or peripherals with SPI interfaces are slaves. The processor can do one of the following:

1. Transmit to all seven SPI devices at the same time in a broadcast mode. Here, all slave select bits are set.

2. Receive and transmit from one SPI device by enabling only one slave SPI device at a time.
3. If all the slaves are also processors, then the requester can receive data from only one processor at a time. (The functionality is enabled by clearing the SPI_CTL.EMISO bit in the six other slave processors). The requestor can transmit broadcast data to all seven at the same time. This EMISO feature can be available in some other microcontrollers. Therefore, it is possible to use the EMISO feature with any other SPI device that includes this functionality.

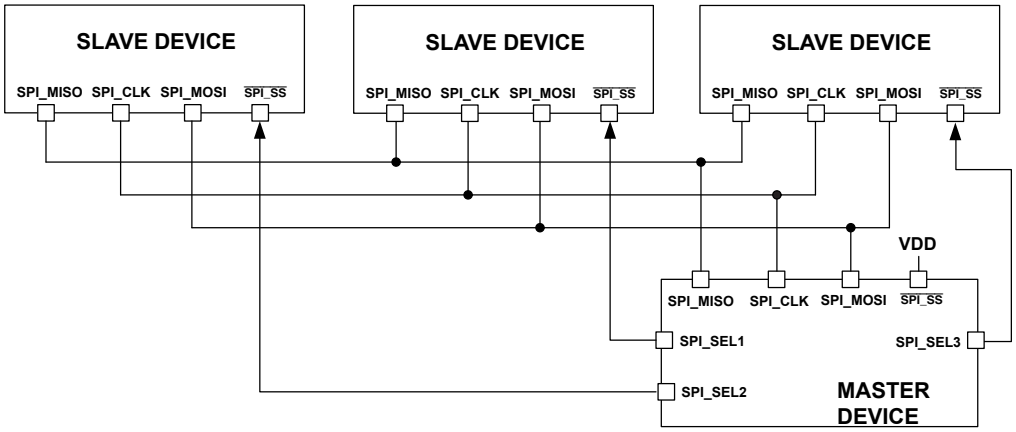

Figure 24-7: Single-Master, Multiple-Slave Configuration

Beginning and Ending a Non-DMA SPI Transfer

The start and finish of a non-DMA SPI transfer depend on the following settings.

1. Whether the device is configured as a master or a slave.
2. The state of the SPI_CTL.ASSEL bit, which selects between hardware and software control over SPI_SLVSEL.

When SPI_CTL.CPHA=0, the enabled slave select outputs are driven active. However, the SPI_CLK signal remains inactive for the first half of the first cycle of SPI_CLK. For a slave with SPI_CTL.CPHA=0, the transfer starts as soon as the SPI_SS input goes low.

When SPI_CTL.CPHA=1, a transfer starts with the first active edge of SPI_CLK for both slave and master devices. For a master device, a transfer is complete after it sends the last data and simultaneously receives the last data bit. A transfer for a slave device ends after the last sampling edge of SPI_CLK. If SPI_CTL.ASSEL=0, the hardware maintains responsibility for toggling SPI_SS between frames. If SPI_CTL.ASSEL=1, software controls the SPI_SS line and can keep it active between frames.

The SPI_STAT.RFE bit defines when the receive buffer can be read, indicating that SPI_RFIFO is not empty. The SPI_STAT.TFF bit defines when the transmit buffer can be written, indicating that the SPI_TFIFO is not full. The end of a single word transfer occurs when the SPI_STAT.RFE bit is cleared. The status indicates that a new word has been received and written into the receive FIFO. The SPI_STAT.RFE bit remains cleared as long as the receive FIFO has valid data.

To maintain software compatibility with other SPI devices, the SPI_STAT.SPIF bit is also available for polling. This bit can have a slightly different behavior from other commercially available devices.

In master mode with the SPI_CTL.ASSEL bit cleared, software manually asserts the required slave select signal before starting the transaction. After all data transfers, software typically releases the slave select line.

When the receive or transmit word counters are enabled in the SPI_TXCTL or SPI_RXCTL registers, the SPI generates a finish interrupt at the end of the transfer. It signals the end of all transfers related to that transaction.

Transmit Operation in Non-DMA Mode

Transmit operation in non-DMA mode is enabled through the SPI_TXCTL.TEN bit. Transmit operation can be enabled independently from the receive operation, and the transmit channel can become the initiating channel based on the SPI_TXCTL.TTI bit setting.

Transmit underrun is not possible in this mode, as no new transfer initiates unless the transmit FIFO is empty (in the case that SPI_TXCTL.TTI =1). A receive overflow is detected when data from a new frame transfer replaces older data in a full receive FIFO. This event can occur if SPI_TXCTL.TTI =1 and the receive channel is enabled in a non-initiating capacity.

A SPI transmit interrupt is signaled once the transmit channel has been enabled and the transmit FIFO is not full. The SPI uses the SPI_TXCTL.TDR bit setting to control the frequency of the interrupt.

Receive Operation in Non-DMA Mode

Receive operation in non-DMA mode is enabled through the SPI_RXCTL.REN bit. Receive operation can be enabled independently from transmit operation, and the receive channel can become the initiating channel based on the SPI_RXCTL.RTI bit setting.

Receive overflow is not possible in this mode, as no new transfer initiates when the receive FIFO is full (in the case of SPI_RXCTL.RTI =1). A transmit underrun can occur (SPI_TXCTL.TDU bit) when no valid data is in the SPI_TFIFO register when a transfer is initiated. This event can occur if SPI_RXCTL.RTI =1 and the transmit channel is enabled in a non-initiating capacity.

A SPI receive interrupt is signaled once the receive channel has been enabled and there is data waiting to be read. The SPI uses the SPI_RXCTL.RDR bit setting to control the frequency of the interrupt.

DMA and Interrupt Multiplexing

Please refer to the *Direct Memory Access (DMA)* chapter for information about DMA multiplexing. Several interrupts and DMA channels in the SPI can be multiplexed.

Dual I/O Mode

In dual I/O mode, the SPI_MISO and SPI_MOSI pins are configured to operate in the same direction which doubles bandwidth. The SPI uses the SPI_CTL.SOSI bit to determine the order of bits on the pins. When set, the processor sends the first bit on the SPI_MOSI pin and the second bit on the SPI_MISO pin. If the SPI_CTL.SOSI bit is cleared, the order is reversed. Since dual I/O mode uses both pins to transmit or receive data, only one channel

can be enabled, either transmit or receive. Flow control through the SPI_RDY pin is supported. Interrupt generation is unaffected by dual I/O mode. However, the gap between successive interrupts is reduced, since the individual transfer latency is halved.

Change to quad SPI mode when the SPI is in a quiescent state.

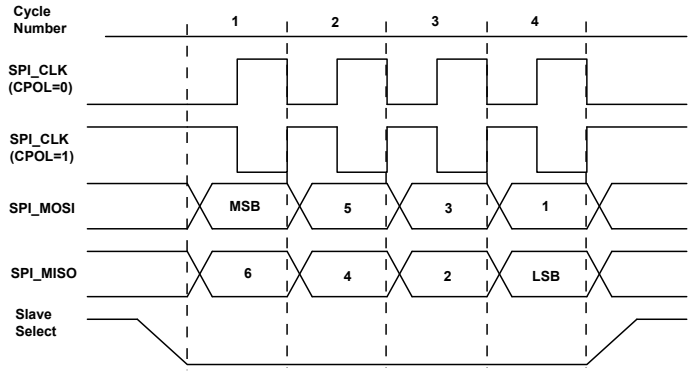

Figure 24-8: Dual I/O Mode Transfer Protocol for CPHA=0, SOSI=1, 8-Bit Transfer, LSBF=0.

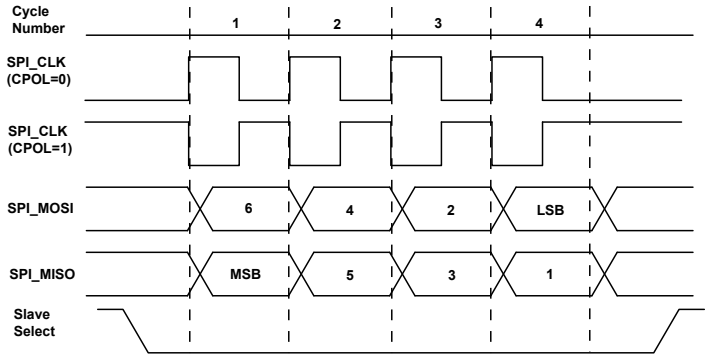

Figure 24-9: Dual I/O Mode Transfer Protocol for CPHA=1, SOSI=0, 8-Bit Transfer, LSBF=0.

Quad I/O Mode (SPI2 only)

In quad SPI mode, the SPI_MISO and SPI_MOSI pins, in tandem with the SPI_D2 and SPI_D3 pins, are configured to operate in the same direction. The SPI uses the SPI_CTL.SOSI bit to determine the order of bits on the pins. When set, the processor sends:

- The first bit on the SPI_MOSI pin
- The second bit on the SPI_MISO pin
- The third bit on the SPI_D2 pin
- The fourth bit on the SPI_D3 pin

If the SPI_CTL.SOSI bit is cleared, the order is reversed. Since quad SPI mode uses all four pins to transmit or receive data, only one channel can be enabled, either transmit or receive. Flow control through the SPI_RDY pin is supported. Interrupt generation is unaffected by quad SPI mode.

Change to quad SPI mode when the SPI is in a quiescent state.

While using dual or quad I/O mode for communicating with SPI flash devices, program the SPI_CTL.CPHA and the SPI_CTL.CPOL bits =1. This programming avoids bus contention during read operations, because the SPI flash device starts driving out the bits immediately after dummy cycles in read header.

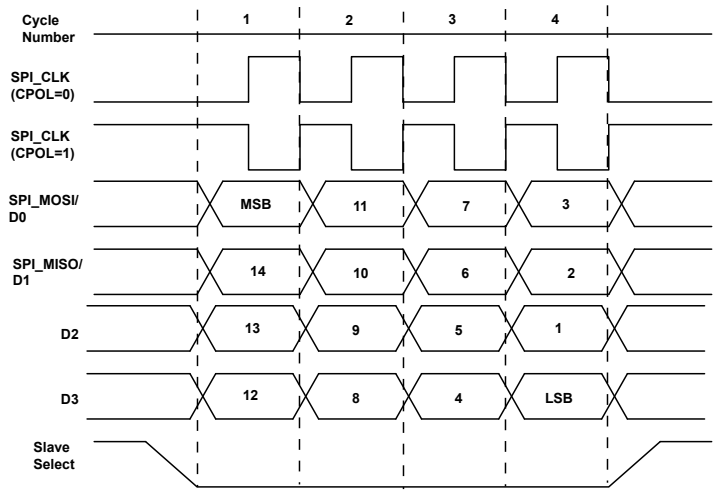

Figure 24-10: Quad Mode Timing for CPHA=0, SOSI=1, 16-Bit Transfer, LSBF=0.

NOTE: The SPI does support quad SPI 8-bit transfer in slave continuous mode of operation with an SCLK:SPI_CLK ratio less than 1:2. A minimum of 2 SCLK cycles is required between transfers in 8-bit quad SPI slave mode with SCLK:SPI_CLK ratio less than 1:2.

Fast Mode

Fast mode is similar to normal mode of operation when transmitting. When receiving, data is sampled at the next transmit edge allowing a full cycle of timing in the receive direction. This mode is valid in master mode operation only. When the SPI operates in fast mode, the slave drives the data for one full cycle.

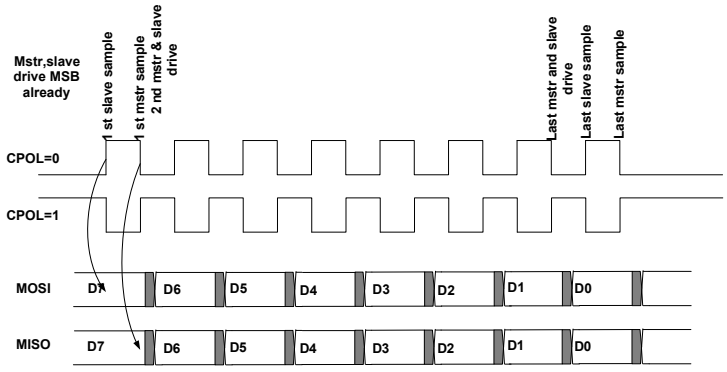

Figure 24-11: SPI Transfer Protocol in Fast Mode for SPI_CTL.CPHA = 0

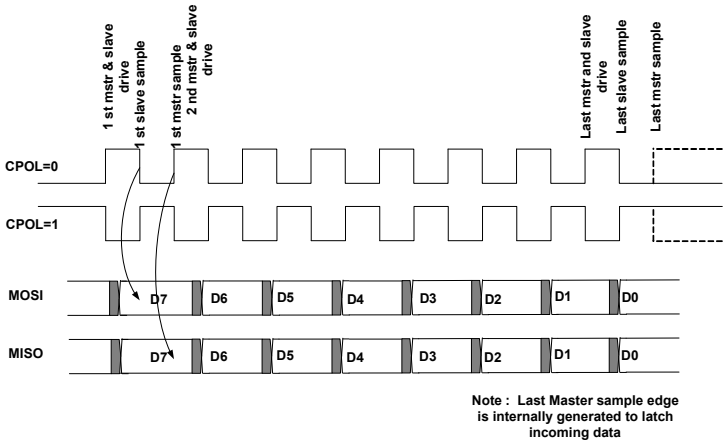

Figure 24-12: SPI Transfer Protocol in Fast Mode for SPI_CTL.CPHA = 1

Memory-Mapped Mode (SPI2 only)

The SPI supports direct memory-mapped read accesses of a SPI memory device. This memory-mapped access mode allows direct execution of instructions from an SPI memory without the need of a low-level software driver. Hardware handles all overhead tasks such as transmission of the read header, pin turnaround timing and receive data sizing. The memory-mapped access mode is enabled by setting the `SPI_CTL.MMSE` (memory mapped SPI enable) bit. The SPI provides more programming flexibility in the memory-mapped read header register (`SPI_MMRDH`) to support and allow compatibility with a wide range of SPI memory devices.

This memory-mapped mode differs from non-memory-mapped mode in the following ways:

- To access the SPI memory devices in non-memory-mapped mode, send the command, address and dummy bytes by directly accessing the transmit data buffer (TFIFO) of the SPI module. Access the buffer either in core mode or by using the peripheral DMA. After setting up the SPI memory device for read/write operations, the data can be accessed in core mode or in DMA mode. (In core mode, access data directly using the TFIFO or RFIFO registers of SPI). This type of access can also require polling the status bits of memory device, which adds to core overhead. In such cases, executing directly from flash is not possible.
- The processor SPI uses memory-mapped mode to respond to some of these concerns. The SPI provides dedicated hardware for SPI memory read accesses. In memory-mapped mode, communication to a SPI memory device is automated. The memory the SPI device contains is accessible directly through reads of processor address space. The read accesses can be code or data-accesses in core mode or using MDMA. This access allows code to execute directly from SPI memory devices (true eXecute-In-Place operations); and the contents can be cached to experience good performance. It is not necessary to access the SPI data buffer registers or poll for any status bits. However, the hardware does not support read accesses by the DMA of other peripherals in the SPI memory region. It also does not support any type of write operations.

The ***Types of Operations*** table summarizes the types of operations possible in SPI non-memory mapped mode and in memory mapped mode.

Table 24-3: Types of Operations

SPI Operations	SPI Non-Memory Mapped Mode	SPI Memory-Mapped Mode
Core data write	Y	N
Core data read	Y	Y
Code fetch: Execute-In-Place (XIP)	N	Y
Read/Write accesses using SPI Peripheral DMA	Y	N
Read/Write accesses by DMA of other peripherals	N	N
MDMA read	N	Y
MDMA write	N	N

CAUTION: Some variants of ADSP-CM40x processor have integrated SPI flash memory. This internal flash memory connects to a dedicated SPI module, SPI2. It does not provide any peripheral DMA. So, program the internal SPI memory in non-memory mapped mode using core mode of SPI. The SPI module does not support the SPI memory write operations. SPI memory mapped hardware cannot access the SPI TFIFO and RFIFO Data registers. Therefore, program or configure the SPI in non-memory mapped mode.

Memory-Mapped Description of Operation

The SPI memory-mapped mode is enabled by setting `SPI_CTL.MMSE` bit. When enabled, the SPI (if ready) accepts the read requests through a dedicated on-chip slave interface. The memory subsystem master drives this dedicated interface through the SCB fabric.

In a typical scenario, the memory subsystem master issues read-requests to the fabric, and the fabric routes these requests to slave port of SPI peripheral. The master describes the read access using a number of parameters such as starting address, transfer size, and burst type. The SPI responds to this read access request when it is ready for a new transfer. It loads the opcode, a specified number of address bytes and an optional mode byte into the transmit FIFO (TFIFO). The SPI memory state machine begins when both the transmit and receive channels of SPI are enabled:

- Transmit transfer initiation bit set (`SPI_TXCTL.TTI` =1), and
- Receive initiation bit clear (`SPI_RXCTL.RTI` =0)

The SPI memory-read sequence starts with the assertion of $\overline{\texttt{SPI\_SEL1}}$. (If the SPI memory state machine is in reset state, it looks for a command). The SPI hardware then sends the 8-bit read command specified (which can be optionally skipped), followed by SPI memory read address. Then, a dummy period is inserted, in which optionally a mode byte is sent and the pins are held or three-stated during the dummy clocking period.

NOTE: This read header is transmitted over the SPI standard protocol pins (`SPI_CLK`, `SPI_MOSI`, `SPI_MISO`, $\overline{\texttt{SPI\_SEL1}}$) or over the extended SPI protocol pins (`SPI_CLK`, `SPI_MOSI`, `SPI_MISO`, `SPI_D2`, `SPI_D3`, $\overline{\texttt{SPI\_SEL1}}$), based on `SPI_MMRDH.CMDPINS`, `SPI_MMRDH.ADRPINS`, and `SPI_CTL.MIOM` bit settings. (SPI memory devices usually support communication in MSB bit first mode. In dual mode,

SPI typically uses SPI_MISO as IO1 and SPI_MOSI as IO0; whereas in quad mode, SPI typically uses SPI_D3 pin as IO3, SPI_D2 as IO2, SPI_MISO as IO1 and SPI_MOSI as IO0).

When all I/O data pins are three-stated, the SPI continues clocking the SPI memory device (which drives out the data bits at the addressed location) until all bytes are received. The SPI hardware reads the data as configured by the SPI_CTL.MIOM bit setting. On reception of the last byte, the SPI typically deasserts the SPI_SEL1 to prepare for the next requested read header.

Programs must ensure that the opcode sent is consistent with multiple I/O programming. Programs must also ensure that the parameters specified in the memory-mapped read header register are with timings of flash read access.

The ***SPI Memory-Mapped Register Operations Flow*** diagram shows how the fields of the SPI_MMRDH register determine the read header while initiating the transfers in memory-mapped mode. (It excludes the merge bit settings).

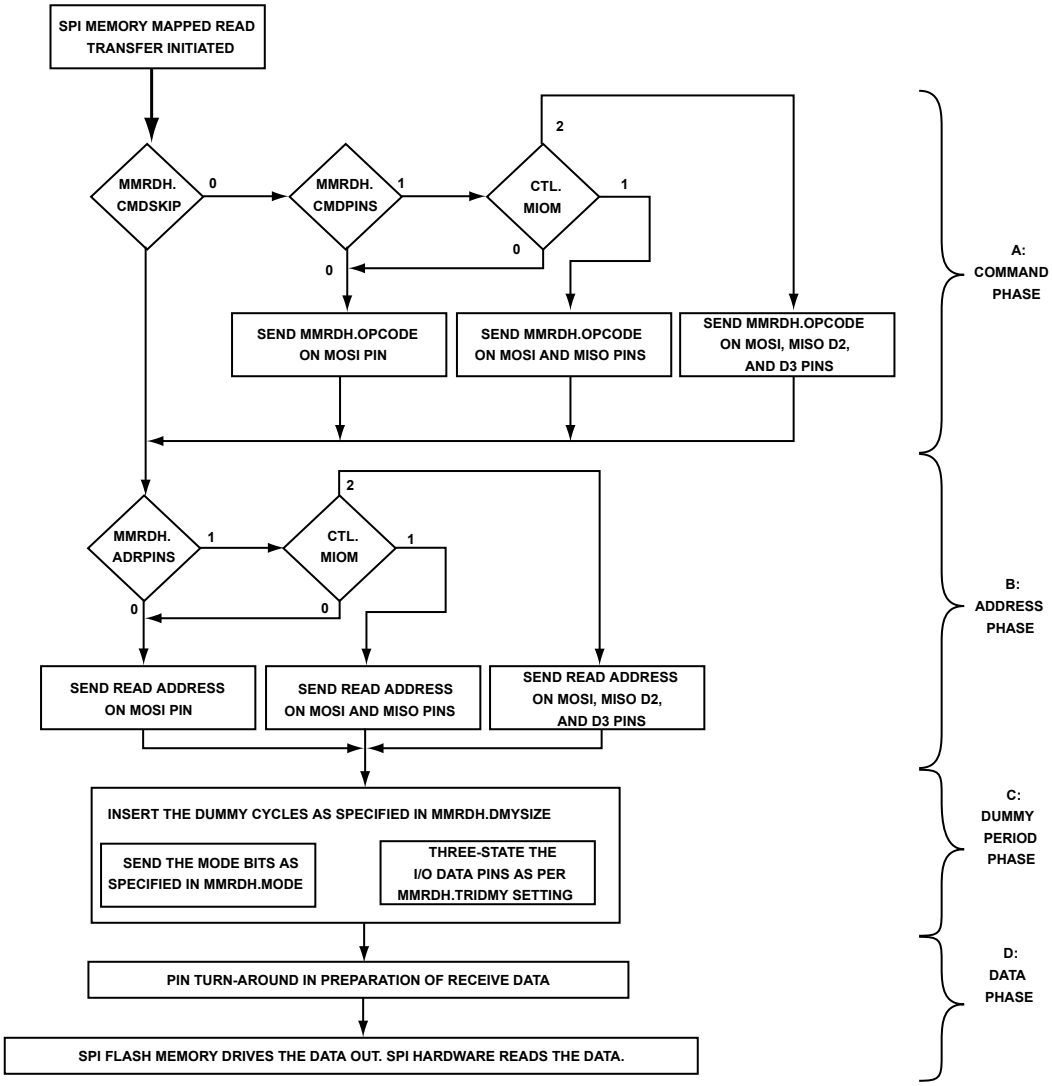

Figure 24-13: SPI Memory-Mapped Register Operations Flow

Memory-Mapped Architectural Concepts

In memory-mapped mode, the SPI accepts read-requests through a dedicated on-chip slave interface. The SPI (if ready) accepts these requests and begins the process of assembling the read-header based on access attributes described in both the SPI_MMRDH register and the internal bus request. After the read header transmission is complete, a pin-turnaround period is timed and the receiver is enabled. The SPI continues clocking the SPI memory device until all bytes are received.

The SPI memory-mapped hardware accommodates various memory devices with different read-timing. The capabilities include extra mode bits, flexible dummy period timing, and three-state control. The SPI uses the memory-mapped read header register (SPI_MMRDH) to configure these capabilities. The ***Memory-Mapped Read Header (MMRDH) Register*** table describes the bits. The ***Memory-Mapped Protocol*** figure shows the protocol for SPI controller in memory-mapped mode.

Table 24-4: Memory-Mapped Read Header (MMRDH) Register

Configuration bits	Description
OPCODE [8]	Read Command Opcode
CMDPINS [1]	No of pins used for sending Command read
CMDSKIP [1]	Command skip enable (for XIP operation)
ADRSIZE [3]	No of Address bytes for Read address
ADRPINS [1]	No of pins used for sending Address
DMYSIZE [3]	No of Dummy bytes
MODE [8]	Mode field (control byte to be driven during dummy period)
TRIDMY [2]	Three-state timing during dummy period
WRAP [1]	Enable Wrapping burst
MERGE [1]	Enable Merging of two successive transfers

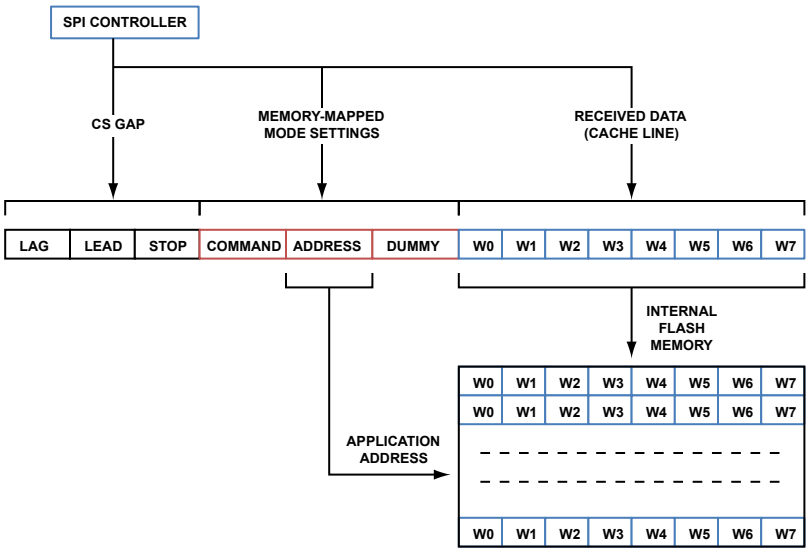

Figure 24-14: Memory-Mapped Protocol

OPCODE (Read command opcode):

In response to a read request of SPI memory, the value placed in this field is the first byte transmitted as part of the read header. The transmission immediately follows the reassertion of the `SPI_SEL[n]` signal. The SPI memory interprets this 8-bit value as a read command. Any 8-bit read opcode whose timing is compliant with the processor SPI and features provided by memory-mapped hardware is allowed. The most common are:

- Standard Read (0x03)
- Fast Read (0x0B)
- Fast Read Dual Output (0x3B)
- Fast Read Dual I/O (0x6B)
- Fast Read Quad Output (0xBB)
- Fast Read Quad I/O (0xEB)
- Word Read Quad I/O (0xE7)
- Octal Word Read Quad I/O (0xE3)

The SPI peripheral sends the OPCODE without interpretation.

CMDPINS (Number of pins used for sending read command opcode)

This bit specifies the number of pins the SPI uses for command transmission. This bit must be set consistent with expectations established by the read command. SPI hardware does not interpret the OPCODE, but rather relies on this bit to specify behavior.

If `SPI_MMRDH.CMDPINS` bit =0, the command is sent on single a `SPI_MOSI` pin of the SPI. The SPI uses 8 SPI clocks to transmit the command.

If SPI_MMRDH.CMDPINS bit =1, the multiple I/O configuration bit field SPI_CTL.MIOM determines the number of pins the SPI uses. If SPI_CTL.MIOM=1 (dual mode), the SPI sends the command on SPI_MOSI and MISO pins and it uses 4 SPI clocks. If SPI_CTL.MIOM=2 (quad mode), the SPI sends the command on the SPI_MOSI, SPI_MISO, SPI_D2, and SPI_D3 pins. The send command completes within 2 SPI clocks.

CMDSKIP (Command opcode skip enable)

This bit determines whether the SPI sends the command as part of the read header (SPI_MMRDH.CMDSKIP=0) or skips it (SPI_MMRDH.CMDSKIP=1). Command skip mode is useful for supporting XIP (Execute-In-Place) operation where only the address is sent and the same read command is assumed. The SPI flash device must be primed with an initial read command before the (SPI_MMRDH.CMDSKIP bit is set.

ADRSIZE (Number of address bytes for read address)

This 3-bit field determines how many address bytes the SPI transmits as part of the read header. The particular SPI memory device determines the number of address bytes. The SPI sends the read address immediately following the transmission of the opcode. The number of bytes can be 1, 2, 3 or 4 as the processor can generate 32-bit absolute addresses. The address sent to a connected SPI memory device is an echo of the read address received by the SCB slave port of the SPI peripheral. Typically SPI flash memory devices require a 3-byte address.

ADRPINS (Number of pins used for sending address)

Similar to the SPI_MMRDH.CMDPINS bit for sending opcode, this bit specifies the number of pins the SPI uses for address transmission.

If this bit =0, the SPI sends the address on a single SPI_MOSI pin. If this bit =1, the SPI sends the address on multiple pins as determined by the multiple I/O configuration field of SPI control register (SPI_CTL.MIOM).

DMYSIZE (Number of dummy bytes)

When operating at a high clock frequency in multi-IO modes, most flash devices require some dummy clocks after address bits. These dummy clock cycles allow the internal circuits of the device extra time for setting up the initial address. These bits specify the number of bytes separating address transmission and read data return.

The number of dummy cycles required varies per manufacturer, the read command used, and in some cases, the SPI access time. The SPI hardware allows dummy cycles to be programmed in bytes. The number of dummy clock cycles depends on the number of pins used to transmit the address (SPI_MMRDH.ADRPINS) as shown in the ***Pins Used to Transmit the Address (ADRPINS)*** table.

Table 24-5: Pins Used to Transmit the Address (ADRPINS)

	Dummy clock cycles		
DMYSIZE<2:0>	**(ADRPINS=0, MIOM=x)** **Dummy bytes elapse over 1-pin**	**(ADRPINS=1, MIOM=1)** **Dummy bytes elapse over 2-pins**	**(ADRPINS=1, MIOM=2)** **Dummy bytes elapse over 4-pins**
000	0	0	0
001	8	4	2

Table 24-5: Pins Used to Transmit the Address (ADRPINS) (Continued)

	Dummy clock cycles		
DMYSIZE<2:0>	(ADRPINS=0, MIOM=x) Dummy bytes elapse over 1-pin	(ADRPINS=1, MIOM=1) Dummy bytes elapse over 2-pins	(ADRPINS=1, MIOM=2) Dummy bytes elapse over 4-pins
010	16	8	4
011	24	12	6
100	32	16	8
101	40	20	10
110	48	24	12
111	56	28	14

This dummy clocking period allows the mode bits to be sent, the pins to be three-stated and the pins to be turned around in preparation for the receive data.

MODE (Mode bits)

The value placed in this field is the last byte transmitted as part of the read header. It specifies the leading byte to transmit during the interval of time specified by the `SPI_MMRDH.DMYSIZE` bits after the completion of the address phase. Mode bits are sent using the same number of pins which are used to transmit the address (configured using the `SPI_MMRDH.ADRPINS` bit).

This first byte, or a portion of it, is interpreted as mode bits when certain opcodes are used with some SPI memory devices. Each vendor (and different devices from a specific vendor) interprets this 8-bit value uniquely. Some vendors use this field to enable or disable opcode skipping (XIP) or to describe a wrapping burst type access. Typically, this field is programmed with a default value of either 0x00 or 0xFF which has no impact on the instruction.

Once the SPI sends these mode bits, the SPI output pins are held in their final state until the conclusion of all dummy byte periods. This state is held unless three-stating of outputs is specified with the `SPI_MMRDH.TRIDMY` bits.

TRIDMY (Three-state dummy timing)

In most SPI flash read operations, after transmitting the read header, the I/O data pins are three-stated before the flash operation starts to drive out the data bits.

The `SPI_MMRDH.TRIDMY` bit field specifies whether and when output pins are three-stated during the interval of time specified by the `SPI_MMRDH.DMYSIZE`, as shown in the ***TRIDMY field and Three-State Period*** table. During the pin turnaround time, programs can three-state the output pins immediately

- after the address bytes are sent, or
- after the mode bytes, or
- for the brief period before the SPI flash device starts sending the first byte of data

Table 24-6: TRIDMY field and Three-State Period

TRIDMY<1:0>	Three-State Period Start
00	Beginning of Dummy Period (MODE bits are not sent)
01	After sending First Nibble of MODE byte
10	After sending MODE Byte
11	End of Dummy Period

WRAP (Wrapping Enable):

When this bit is set, the SPI fetches the critical word first against the line base fetch first. For more information, refer to the Wrap Around Accesses section.

MERGE (Merging Enable):

When this bit is set, SPI hardware combines the two successive transfers. This functionality increases the throughput rate when accessing many sequential memory locations. For more information, refer to the Merged Read Accesses section.

Memory-Mapped Read Accesses

The SPI hardware supports the most commonly used read operations.

- Two standard SPI reads (read and read fast), which use the unidirectional `SPI_MOSI` and `SPI_MISO` pins in addition to `SPI_SEL[n]` and `SPI_CLK`.
- Four extended SPI multiple I/O reads: dual output, quad output, dual I/O, and quad I/O reads.

The ***SPI Read Operations*** table and ***SPI Flash Fast Read Sequence*** figures summarize the types of read operations. Program each read operation in a way that is compatible with the description given in the SPI flash data sheet.

Table 24-7: SPI Read Operations

Operation	Read Command (Opcode)	CMDPIN	ADRPIN	DMYSIZE	Three-state	Multiple I/O Mode	Data Pins
Read	0x03	1	1	zero	No	No	1
Fast Read	0x0B	1	1	Non-Zero	Yes	No	1
Dual Output Read	0x3B	1	1	Non-Zero	Yes	Yes(IO0-1)	2
Quad Output Read	0x6B	1	1	Non-Zero	Yes	Yes(IO0-3)	4
Dual I/O Read	0xBB	1, 2	2	Non-Zero	yes	Yes (IO0-1)	2
Quad I/O Read	0xEB	1, 4	4	Non-Zero	yes	Yes (IO0-3)	4

Some memory devices also support word quad I/O read (0xE7) and octal quad I/O read (0xE3) operations that require fewer dummy cycles compared to normal quad I/O read operations.

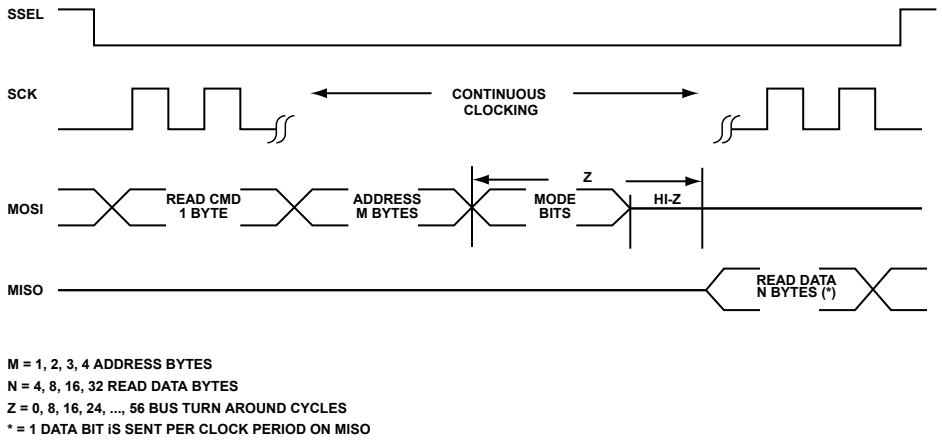

Figure 24-15: SPI Flash Fast Read Sequence

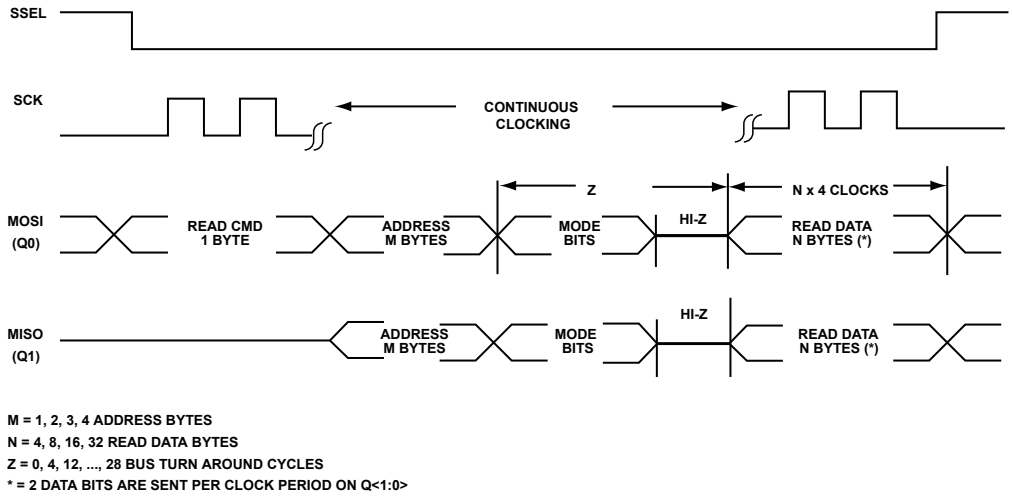

Figure 24-16: SPI Flash Fast Read (Dual Output) Sequence

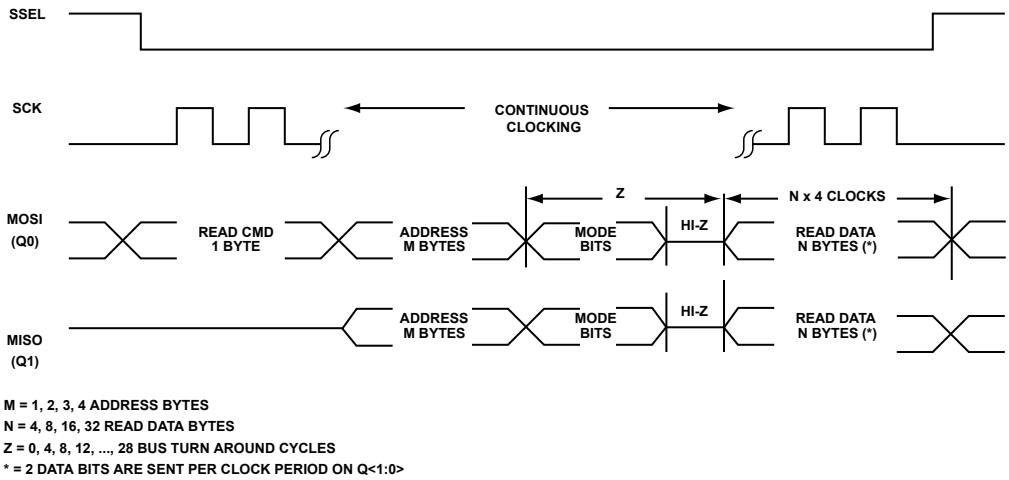

Figure 24-17: SPI Flash Fast Read (Dual I/O) Sequence

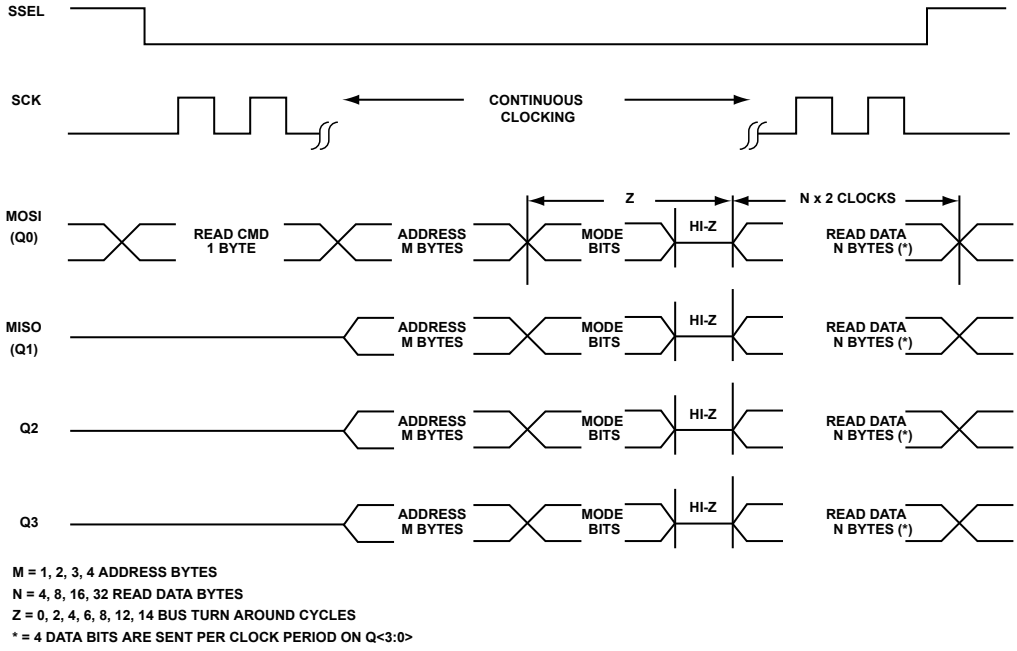

Figure 24-18: SPI Flash Quad Output Read Sequence

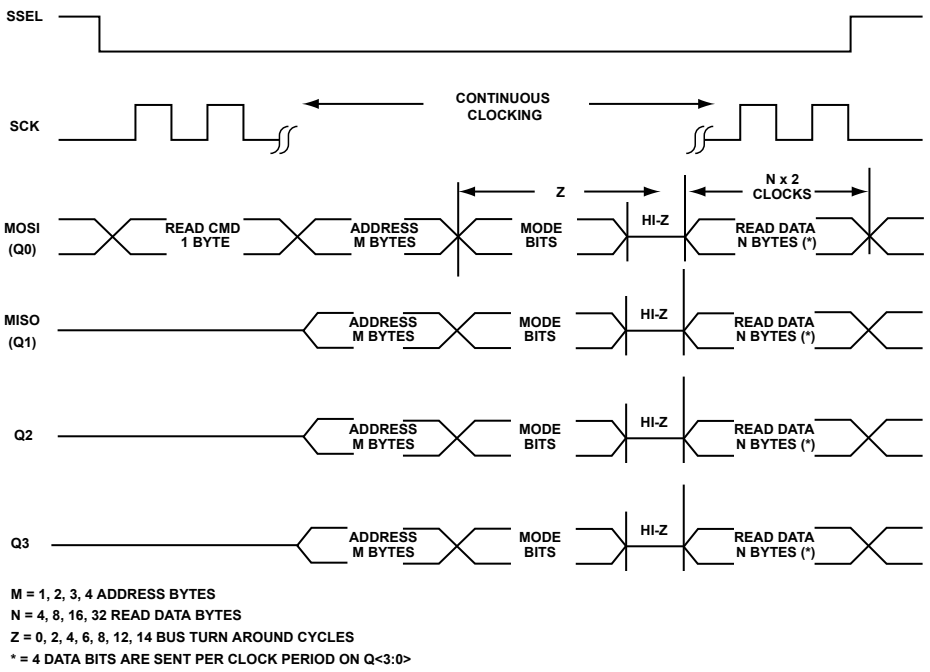

Figure 24-19: SPI Flash Quad I/O Read Sequence

Programs can enable the cache for the SPI memory-mapped operations (CMODE and PEMODE bits =0). The number of read data bytes, (N in the figures) is based on the following:

- For an instruction fetch by core (when in XIP mode); the number of instruction bytes to fetch depends the cache line size of the cache.
- For a data fetch by the core (data read), the number of data bytes to fetch depends on the cache line size of the cache.

Although the minimum size of a memory-mapped data read transfer is 4 bytes, applications can fetch a single byte or a 2-byte data. (For example, it can fetch an unsigned char or short access in C code). In this case, only the required bytes are provided to the core and the other bytes are cached.

For more information about cache settings, refer to the SPI (Internal) Flash chapter.

The on-chip memory subsystem master provides a starting address for the burst and the SPI hardware issues this address as part of the read header. The address provided is N-byte aligned. For example, to read the 30th byte from SPI memory, then the typical address to provide is:

- 28 (0x0000_001C) for a 32-bit cache line
- 24 (0x0000_0018) for a 64-bit cache line
- 16 (0x0000_0010) for a 128-bit cache line
- 0 (0x0000_0000) for a 256-bit cache line

The read data is returned to the memory subsystem in the order provided by the SPI memory. There can be considerable delay for the expected data provided to the master.

To minimize this delay, the wrap feature can be used where the memory subsystem provides the address of the critical word.

- For MDMA reads, the number of read data bytes (N) is always equal to 4 bytes. The MDMA read does not depend on the cache setting. Programs must use the MSIZE field of the MDMA control register as 1, 2 or 4 only. The address provided by the memory subsystem master to the SPI hardware is always 4 byte-aligned.

Memory-Mapped High-Performance Features

In addition to automating the SPI memory read accesses, the memory-mapped hardware also provides some features to improve SPI memory fetches and increase the system performance. The following sections describe these features.

Merged Read Accesses

It is common for the memory subsystem to fetch two or more cache lines from consecutive addresses (the address sequencing is linear without any jumps). To take advantage of this situation, the SPI memory-mapped hardware provides a feature called merging. Enable merging by setting the `SPI_MMRDH.MERGE` bit.

When enabled, the hardware compares the address of an incoming read request to the address of a request the SPI memory is actively servicing. It can decide to merge two accesses when the address for the second access is incremental. For example, if the first address of a 32-byte cache line fetch is 0x0000_0000 and the second fetch is to address 0x0000_0020, then these two accesses can be merged. Merging increases efficiency and overall fetch bandwidth by eliminating the read header for those accesses which only require continuation of the SPI clock.

Wrap Around Accesses

Many SPI flash memory devices support wrapping which is used to enhance critical word fetching of cache lines. In this mode, the SPI device automatically wraps the read address to the base of a cache line once the end of the cache line is reached.

Wrap around accesses are enabled by setting the `SPI_MMRDH.WRAP` bit.

Some flash devices require programs to send a ***Set Wrap*** command to place the device in wrap mode. Other flash devices provide a configuration register which must be programmed to set the flash in wrap mode. Since the SPI memory-mapped hardware does not support any write operations to flash, perform this step in non-memory-mapped mode (`SPI_CTL.MMSE`=0) by accessing the SPI registers.

Data access is limited to 8-byte, 16-byte, or 32-byte sections of flash page in wrap mode. The ARM core uses the Wrap 4 access (64-bit data) for L1 cache. The ARM core uses Wrap 4 and Wrap 8 accesses (64-bit data) for L2 cache. The cores use the Wrap 8 accesses for un-aligned accesses. During the read request to the SPI memory-mapped hardware, the memory subsystem master of the processor provides the address of a critical word instead of the line base. The read-data starts at the address specified in the instruction. Once it reaches the end boundary of the 8, 16, or 32-byte section, the output automatically wraps around to the beginning boundary to the line base address. The data fetch continues. It is not necessary to deassert the SPI `SPI_SEL[n]` signal or resend the read header to wrap to the cache line base when servicing misaligned cache fill requests.

The ***Byte Sequence in Wrap Modes*** table shows byte sequences in various wrap modes.

Table 24-8: Byte Sequence in Wrap Modes

Starting Address	8-Byte Wrap (cache_line = 8 byte)
0	0-1-2- . . . -6-7
1	1-2- 3-. . . -7-0
7	7-0-1- . . . -5-6
15	15-8-9- . . . -13-14
31	31-24-25-. . . .-29-30

The burst with wrap feature allows applications to fetch a critical address quickly. Applications then fill the cache afterwards within a fixed length (8/16/32-byte) of data without issuing multiple read commands. Certain applications can benefit from this feature to improve cache fill efficiency and overall performance of system code execution.

NOTE: Do not use the merge and wrap feature together. Using wrap bursts can unintentionally disable merging (merging cannot occur for unaligned wrapping bursts). A wrap burst can start fetching data words in the middle of the cache line and cannot be merged with the next access.

Execute-In-Place (XIP, SPI2 only)

Execute-In-Place, most commonly known as XIP, allows software code to execute directly from a SPI flash device rather than downloading the code and executing it out of RAM. XIP is a general term and can be applied to fetching data as well.

There is a difference between XIP mode and standard mode. In XIP mode, after the SPI memory device is selected (CS# =LOW), the memory device does not decode the first input byte as command code. Instead, it decodes the byte as the first part of a 3-byte address. In other words, the read header directly starts with address bytes. In standard mode, the memory decodes the first input byte it receives as a command code.

The XIP mode dramatically reduces random access time for applications that require fast code execution without shadowing the memory content on a RAM. The SPI memory-mapped hardware provides a control bit, `SPI_MMRDH.CMDSKIP` to skip the command from read header.

Some SPI memory devices require configuration of their control register to enable the XIP mode of operation, using the non-memory mapped mode of the SPI for the processor. Typically, during the dummy cycle period, the mode bits are used to confirm the XIP operation and the `SPI_MMRDH.MODE` field must be set appropriately. A dummy memory-mapped access can be needed before setting the `SPI_MMRDH.CMDSKIP` bit.

For more details about how to configure SPI memories into XIP mode, refer to the device data sheet.

Memory-Mapped Mode Error Status Bits

The SPI memory-mapped hardware provides bits in the `SPI_STAT` register to report errors. It provides these bits for notification only and their state has no effect on SPI operations. The status register bits are sticky. A W1C (write-1-to-clear) operation clears the bits.

- Memory-Mapped Write Error (SPI_STAT.MMWE). This bit is set (=1) if an attempt is made to write address space reserved for memory-mapped SPI memory. The SPI memory-mapped hardware does not support automated write access to SPI memory space.
- Memory-Mapped Read Error (SPI_STAT.MMRE). This bit is set (=1) if an attempt is made to read address space reserved for memory-mapped SPI memory while memory mapping is disabled (SPI_CTL.MMSE =0).
- Memory-Mapped Access Error (SPI_STAT.MMAE). This bit is set (=1) if an attempt is made to access either the TX or RX FIFO while memory-mapped access of SPI memory is enabled. In this case, attempts to communicate with the SPI device using legacy methods are blocked and receive fabric error responses. Legacy methods include any direct access made to the TX and RX FIFOs, whether by DMA or processor MMR.
- Memory-Mapped Write Error Mask (SPI_CTL.MMWEM) bit specifies whether an error response is returned to the fabric on write attempts to address space that is reserved for memory-mapped SPI memory reads. Regardless of whether a write error response is masked using this bit, the memory-mapped write error (SPI_STAT.MMWE) sticky notification bit is still set.

NOTE: Unlike other bits in the SPI_STAT register, these memory-mapped mode error bits do not have associated bits in the SPI interrupt mask (SPI_IMSK) and SPI interrupt condition (SPI_ILAT) registers.

The memory-mapped top register (SPI_MMTOP) is used to specify the top limit of the SPI memory address. The memory-mapped accesses to SPI memory addresses equal to or above this range are considered illegal. The accesses are blocked and a bus error response is generated.

This register is useful to block the invalid SPI memory address accesses. Some SPI memory vendors do not clearly specify (guarantee) that overrange address bits are ignored (address spaces can be wrapped).

Memory-Mapped Programming Guidelines

Set the SPI_CTL.MMSE bit to enable SPI memory-mapped mode. When enabled, the SPI interface is forced to be consistent with SPI memory requirements regardless the settings of certain control bits. The following tables specify typical settings for configuring the SPI in memory-mapped mode:

Table 24-9: SPI Control (SPI_CTL) Register

Bits	Typical values to set	Description	Comments
SPI_CTL.MSTR	1	Master mode enable	
SPI_CTL.PSSE	0	Protected slave select enable	
SPI_CTL.ODM	0	Open-Drain mode enable	
SPI_CTL.CPHA-SPI_CTL.CPOL	0–0 or 1–1	SPI mode of communication	Flash dependent, usually SPI flash supports mode-0 (CPHA=CPOL=0) and mode-3 (CPHA=CPOL=1)

Table 24-9: SPI Control (SPI_CTL) Register (Continued)

Bits	Typical values to set	Description	Comments
SPI_CTL.ASSEL	1	Hardware Slave Select Pin Control	
SPI_CTL.SELST	1	Slave select asserted between transfers	
SPI_CTL.EMISO	1	MISO pin enable	
SPI_CTL.SIZE	2	32-bit Transfer size	
SPI_CTL.LSBF	0	MSB bit first mode	Flash dependent, usually SPI flash communicates in MSB bit first mode
SPI_CTL.FCEN SPI_CTL.FCCH SPI_CTL.FCPL SPI_CTL.FCWM	0	Hardware flow control related bits	
SPI_CTL.FMODE	1	FAST mode enable	Typically set to 1 for full cycle timing, 0 only works at low speed
SPI_CTL.SOSI	0	Treat SPI_MOSI pin as IO0 pin.	

Table 24-10: SPI Receive Control Register

Bits	Typical values to set	Description
SPI_RXCTL.REN	1	Receive channel enable
SPI_RXCTL.RTI	0	Receive Transfer initiation disable
SPI_RXCTL.RWCEN	0	Receive Word Counter disable
SPI_RXCTL.RDR	0	Receive Data Request disable
SPI_RXCTL.RDO	0	Discard incoming data if RFIFO is full
SPI_RXCTL.RRWM	0	Receive FIFO Regular watermark
SPI_RXCTL.RUWM	0	Receive FIFO Urgent watermark disable

Table 24-11: SPI Transmit Control Register

Bits	Typical values to set	Description
SPI_TXCTL.TEN	1	Transmit channel enable
SPI_TXCTL.TTI	1	Transmit Transfer initiation disable
SPI_TXCTL.TWCEN	0	Transmit Word Counter disable
SPI_TXCTL.TDR	0	Transmit Data Request disable
SPI_TXCTL.TDU	0	Send last word when TFIFO is empty

Table 24-11: SPI Transmit Control Register (Continued)

Bits	Typical values to set	Description
SPI_TXCTL.TRWM	0	Transmit FIFO Regular watermark
SPI_TXCTL.TUWM	0	Transmit FIFO Urgent watermark disable

Table 24-12: SPI DLY Control Register

Bits	Typical values to set	Description	Comments see Flash data sheet for CS (for example. SSEL) timing specs
SPI_DLY.LAGX	1	Extended LAG timing	
SPI_DLY.LEADX	1	Extended LEAD timing	
SPI_DLY.STOP	3	STOP bit between the transfers	Can be set to 1 at lower SPI clock frequencies.

The multiple I/O mode (SPI_CTL.MIOM) bits are partially ignored:

- The command (opcode) is transmitted using either just one or the number of pins specified by the SPI_CTL.MIOM bits, depending on SPI_MMRDH.CMDPINS bit setting.
- The address is then transmitted using either just one or the number of pins specified by the SPI_CTL.MIOM bits, depending on SPI_MMRDH.ADRPINS bit setting.
- The data is always read with the number of pins specified by the SPI_CTL.MIOM bits.

NOTE: Set the SPI module enable bits SPI_CTL.EN last after configuring all registers

Use the following the programming guidelines for the memory-mapped mode.

- The SPI memory-mapped hardware does not check the flash status before initiating the access. It assumes that SPI memory is always able to respond to a read access. Take care before enabling memory-mapped mode (for example setting the SPI_CTL.MMSE bit) so that SPI flash is ready for a read access. When using non-memory-mapped mode, a write-complete status can be examined prior to enabling SPI in memory-mapped mode (See the write in progress bit in the SPI flash memory status register). Also, immediately after initial power-up, SPI memory devices can be inaccessible for a vendor-specified period.
- When SPI is enabled in memory-mapped mode, attempts to communicate with the SPI device using legacy methods are blocked. Legacy methods include any direct access made to the transmit or receive FIFOs, whether by initiated by DMA or processor MMR access.
- To use some of the features offered by SPI memory devices, programs could need to first configure the SPI memory device by setting its control word or sending some commands. Since SPI memory-mapped hardware does not allow any type of SPI write operations, configure the SPI in non-memory-mapped mode prior to enabling memory-mapped mode.

- The memory-mapped hardware does not interpret the opcode. It does not check the validity of the timing that is specified in the SPI_MMRDH register for a particular opcode. Programs must set the fields of the SPI_MMRDH register to be consistent with the read-type selected.
- When the core requests the data or code fetch, the memory-mapped transfer depends on cache settings. The cache configuration register in the SPI memory device must be appropriately configured before enabling memory-mapped mode. Some of the high performance modes like MERGE, WRAP, and transfer size depend on cache parameters.
- Read accesses made by MDMA do not use cache. Do not depend on cache settings. The start address specified in the MDMA_START_ADDR register is the absolute address of SPI flash address. Read accesses made by core (code or data fetch) go through the cache and depend on the cache settings. Set the MEMX and MEMY registers of cache to a proper value before enabling memory-mapped mode.
- SPI memory-mapped MDMA reads do not support wrapping. For MDMA reads, limit the DMA_CONFIG.MSIZE field to 1 byte, 2 bytes or 4 bytes.
- There is not always tool support to change the SPI memory-mapped hardware setting or cache settings on-the-fly. Changing these settings can optimize the performance of code that accesses SPI memory in memory-mapped mode. It is expected that the user programs the SPI memory, SPI peripheral, and cache to one specific set of control settings for the whole application. The user can do some profiling or benchmarking of their actual application to find the setting that works best.

Programming Example for Configuring SPI Memory Mapped Mode (Winbond W25Q32)

This example shows how to set XIP or wrap for Quad I/O mode of Winbond W25Q32 SPI flash (assuming SPI memory device is connected externally to SPI0 module):

• SPI Initialization

```
/* Initialise the Pin Mux logic of processor to enable SPI pins */
               Init_SPI_PinMux();

/* Configure the SPI Baud and SPI clock setting....*/
/* SPI_CLK = SCLK / (BAUD + 1)*/
     *pREG_SPI0_CLK = ((0x00 << BITP_SPI_CLK_BAUD) & BITM_SPI_CLK_BAUD );
     *pREG_SPI0_DLY = (((3 << BITP_SPI_DLY_STOP  ) & BITM_SPI_DLY_STOP )|
                      ((1 << BITP_SPI_DLY_LEADX ) & BITM_SPI_DLY_LEADX)|
                      ((1 << BITP_SPI_DLY_LAGX  ) & BITM_SPI_DLY_LAGX ));
/* Enable the Memory Mapped mode of SPI with following settings:
   Master mode, 32-bit transfer size, HW Slave select, MSB bit first,
   Quad Multi-IO mode, FAST enable. CPHA-CPOL settings depends on FLASH device. */
               *pREG_SPI0_CTL =(ENUM_SPI_CTL_MM_EN   |
                       ENUM_SPI_CTL_MASTER  |
                       ENUM_SPI_CTL_SIZE32  |
                       ENUM_SPI_CTL_HW_SSEL |
                       ENUM_SPI_CTL_ASSRT_SSEL |
```

```
                        ENUM_SPI_CTL_MSB_FIRST  |
                        ENUM_SPI_CTL_FAST_EN  |
                        ENUM_SPI_CTL_MIO_QUAD |
                        ENUM_SPI_CTL_STMISO   |
                        ((CPHA << BITP_SPI_CTL_CPHA) & BITM_SPI_CTL_CPHA)|
                        ((CPOL << BITP_SPI_CTL_CPOL) & BITM_SPI_CTL_CPOL));

      /* Enable the Transmit part of SPI with TTI = 1; other bits should be kept
to default values*/
           *pREG_SPI0_TXCTL = (BITM_SPI_TXCTL_TTI | BITM_SPI_TXCTL_TEN);

      /* Enable the Receive part of SPI with RTI = 0; other bits should be kept
to default values*/
           *pREG_SPI0_RXCTL = (BITM_SPI_RXCTL_REN );

      /* Configure the Memory Mapped Read Header as per read mode selected*/
           *pREG_SPI0_MMRDH =(
                    ((0xE3<< BITP_SPI_MMRDH_OPCODE)& BITM_SPI_MMRDH_OPCODE)|
                    ((3 << BITP_SPI_MMRDH_ADRSIZE) & BITM_SPI_MMRDH_ADRSIZE)|
                    ((1 << BITP_SPI_MMRDH_ADRPINS) & BITM_SPI_MMRDH_ADRPINS)|
                    ((1 << BITP_SPI_MMRDH_DMYSIZE) & BITM_SPI_MMRDH_DMYSIZE)|
                    ((0 << BITP_SPI_MMRDH_MODE)   & BITM_SPI_MMRDH_MODE) |
                    ((1 << BITP_SPI_MMRDH_TRIDMY) & BITM_SPI_MMRDH_TRIDMY) |
                    ((0 << BITP_SPI_MMRDH_MERGE)  & BITM_SPI_MMRDH_MERGE)|
                    ((0 << BITP_SPI_MMRDH_WRAP)   & BITM_SPI_MMRDH_WRAP) |
                    ((0 << BITP_SPI_MMRDH_CMDSKIP) & BITM_SPI_MMRDH_CMDSKIP)|
                    ((0 << BITP_SPI_MMRDH_CMDPINS) & BITM_SPI_MMRDH_CMDPINS));

/* Top addr of Flash device*/
           *pREG_SPI0_MMTOP = 0x50000000 + (FLASH_BLOCK_SIZE * FLASH_BLOCK_COUNT);

/* Slave Select Control reg*/
            *pREG_SPI0_SLVSEL= ENUM_SPI_SLVSEL_SSEL1_HI |
ENUM_SPI_SLVSEL_SSEL1_EN;

/* Enable SPI*/
            *(pREG_SPI0_CTL + SPI_module_Offset*SPI_ID) |= BITM_SPI_CTL_EN;
```

- **Setting XIP mode**

```
  /* configure the STATUS register-2 of flash to set QE bit (Quad mode enable).
This programming should be done in non-memory-mapped mode */

      /* Enable the SPI module in memory-mapped mode*/

      /* Set the Mode bit in SPI Memory mapped Read Header Reg such that M(5,4) =
1,0*/

           *pREG_SPI0_MMRDH |= ((0x20 << BITP_SPI_MMRDH_MODE) &
```

```
BITM_SPI_MMRDH_MODE);

      /* Make one Dummy access to SPI flash */
           unsigned int* pSPI_MEM;
           pSPI_MEM = (unsigned int*) 0x19000000;
           int temp = *pSPI_MEM;

       /* After the Dummy access, set the Command Skip bit of SPI Memory Mapped
Read Header register */
            *pREG_SPI0_MMRDH |= BITM_SPI_MMRDH_CMDSKIP;

      /* All the SPI flash read accesses after this, is in XIP mode*/
```

Setting Wrap mode

- Configure the Flash Device for wrap using the Set the Burst with WRAP command for the required cache line. (SPI is in non-memory-mapped mode)
- Configure the cache to use the required cache line with the `M4P_CACHE_CFG.DLBF` bit =0
- Enable the SPI module in memory-mapped mode as shown in listing 4 with the `SPI_MMRDH.WRAP` bit =1.

  ```
  *pREG_SPI0_MMRDH |= ((1 << BITP_SPI_MMRDH_WRAP ) & BITM_SPI_MMRDH_WRAP);
  ```

- XIP mode can also be configured to achieve the best throughput.

SPI Interrupt Signals

The SPI controller supports three types of interrupt signals, corresponding to data, status, and error conditions.

Data Interrupts

The SPI peripheral supports two data interrupt channels – receive and transmit. These interrupt signals are multiplexed into the DMA request lines. Since the peripheral interfaces with separate read and write interfaces with DMA, the read and write data interrupts are independent. When the DMA channels are not used, the interrupts are routed directly to the system event controller. The interrupts occupy the same vector locations as the corresponding DMA channels.

Each of the data interrupts can be individually controlled. Program the `SPI_RXCTL.RDR` and `SPI_TXCTL.TDR` bit fields for receive and transmit, respectively. When receive is enabled, the RX interrupt is issued whenever there is data available in the receive datapath for reading. (The event occurs according to the `SPI_RXCTL.RDR` bit setting). When transmit is enabled, the TX interrupt is issued whenever the transmit datapath can be written. (The event occurs according to the `SPI_TXCTL.TDR` setting). DMA data interrupts are compatible with second-generation DMA to incorporate urgent data requests and transfer finish interrupts apart from usual data request interrupts. Transmit interrupts operate independently from the word counter-value in the `SPI_TWC` register.

Status Interrupts

The SPI controller supports several status interrupts to indicate different conditions of the receiver and transmitter. All status interrupts can be masked. Status interrupts are signaled directly through a single SPI status IRQ line. The line cannot be combined with the SPI error IRQ line for some processors. The ***SPI Status Interrupts*** table describes the status interrupts that are available for the SPI controller.

Table 24-13: SPI Status Interrupts

SPI_STAT Bit	Description
SPI_STAT.RUWM	Receive FIFO urgent watermark interrupt. Issued when the level of the RFIFO breaches the watermark set in the SPI_RXCTL.RUWM field. It is cleared when the level of the RFIFO reaches the watermark set in the SPI_RXCTL.RRWM field. If the RX channel is configured in DMA mode, RUWM is multiplexed with the data request.
SPI_STAT.TUWM	Receive FIFO urgent watermark interrupt. Issued when the level of the TFIFO breaches the watermark set using the SPI_TXCTL.TUWM bit. It is cleared when the level of the TFIFO reaches the watermark set in the SPI_TXCTL.TRWM field. If the TX channel is configured in DMA mode, TUWM is multiplexed with the data request.
SPI_STAT.TS	Transmit start interrupt. Issued when the start of a transmit burst is detected by loading of the SPI_TWC register with the contents of the SPI_TWCR register.
SPI_STAT.RS	Receive start interrupt. Issued when the start of a receive burst is detected by loading of SPI_RWC with the contents of SPI_RWCR.
SPI_STAT.TF	Transmit finish interrupt. Issued when a transmit burst completes (SPI_TWC decrements to zero).
SPI_STAT.RF	Receive finish interrupt. Issued when a receive burst completes (SPI_RWC decrements to zero).

Error Conditions

The SPI controller supports interrupts upon several different error conditions. All interrupts are maskable. The individual interrupt indications combine into a single SPI error IRQ signal, which can be multiplexed on some processors with the aggregated SPI status IRQ signal. The ***SPI Error Interrupts*** table details the possible error indications.

Error conditions and interrupts arise depending on which of the channels (transmit or receive) are enabled. If a channel is disabled, all errors relating to it are ignored. When both channels are enabled, errors and interrupts from both channels are enabled.

Table 24-14: SPI Error Interrupts

Bit	Description
SPI_STAT.MF	Mode fault. Signaled when another device also tries to be a master in a multi-master system and drives the SPI_SS input low. This error is signaled in master mode operation.
SPI_STAT.TUR	Transmission error. Signaled when an underflow condition occurs on the transmit channel. This event occurs when a new transfer starts but SPI_TFIFO is empty. This error does not occur in master transmit initiating mode since SPI_TFIFO Not Empty is one of the conditions for transfer initiation.

Table 24-14: SPI Error Interrupts (Continued)

Bit	Description
SPI_STAT.ROR	Reception error. Signaled when an overflow condition occurs on the receive channel. This event occurs when a new data word is received, but the SPI_RFIFO is full. This error condition does not occur in master receive initiating mode since SPI_RFIFO not full is one of the conditions for transfer initiation.
SPI_STAT.TC	Transmit collision error. Signaled when loading data to the transmit shift register happens near the first transmitting edge of SPI_CLK. In slave mode of operation, the SPI controller is unaware of when the next transfer starts. Loading of data to the transmit shift register can happen just after the transmitting edge. This event results in setup time not met for the first bit transmitted. The transmitted data is corrupt. In SPI_CTL.CPHA 1 mode, the first SPI_CLK edge is taken as first transmitting edge. If SPI_CTL.CPHA =0, then the last SPI_CLK edge of the last transmission (SPI_CTL.SELST =1) or slave select deassertion (SPI_CTL.SELST =0) is taken as the first transmitting edge. This error is signaled only in slave mode of operation. In master mode of operation, loading of data happens before the first transmitting edge of SPI_CLK.

SPI Programming Concepts

The following sections provide general programming guidelines and procedures.

Programming Guidelines

It is acceptable to program SPI_RXCTL and SPI_TXCTL registers after programming SPI_CTL. However, program the initiating mode register and its counter-register, if enabled, after the non-initiating mode register. For example, if transmit is the initiating mode and receive is the non-initiating mode, then program SPI_RXCTL and SPI_RWC before SPI_TXCTL and SPI_TWC. If enabling both transmit and receive in initiating mode, enable SPI_CTL after programming both SPI_RXCTL and SPI_TXCTL.

These programming guidelines prevent SPI from starting a transfer when SPI registers are not fully programmed. Other ways of programming are also allowed as long as initiating conditions prevent the start of communication until after programming of SPI registers is complete.

Take precautions to avoid data corruption when changing the SPI module configuration. Do not change the configuration during a data transfer. Additionally, change the clock polarity only when no slaves are selected. An exception to this rule exists. When an SPI communication link consists of a single master and slave, SPI_CTL.ASSEL = 0. The slave select input of the slave is permanently tied low. In this case, the slave is always selected. Avoid data corruption by enabling the slave only after both the master and slave devices are configured.

The module supports 8, 16-bit and 32-bit word sizes. To ensure correct operation, configure both the master and slave with the same word size.

Master Operation in Non-DMA Modes

This section describes the operation of the SPI as a master in non-DMA mode.

1. Write to the SPI_SLVSEL register, setting one or more of the SPI select enable bits. This operation ensures that the desired slaves are properly deselected while the master is configured.

2. The SPI_RXCTL.RTI and SPI_TXCTL.TTI bits determine the SPI initiating mode. The initiating mode defines the primary transfer channel, and also the initiating condition for the transfer.
3. Write to the SPI_CLK, SPI_CTL, SPI_RXCTL, and SPI_TXCTL registers. This operation enables the device as a master and configures the SPI system. It specifies the transfer modes and channels, appropriate word length, transfer format, baud rate, and other control information.

   ADDITIONAL INFORMATION: If SPI_RXCTL.RTI is enabled and SPI_TXCTL.TTI is not, write to the SPI_RXCTL register after writing into SPI_CTL, SPI_TXCTL, and SPI_TFIFO registers to prevent a transmit underrun for the first transfer.
4. If SPI_CTL.ASSEL=0, activate the desired slaves by clearing one or more of the SPI_SLVSEL flag bits. Otherwise, the SPI hardware takes care of slave activation.
5. The SPI controller then generates the programmed clock pulses on SPI_CLK and simultaneously shifts data out of SPI_MOSI while shifting data in from SPI_MISO. Before a shift, the shift register is loaded with the contents of the SPI_TFIFO register. At the end of the transfer, the contents of the shift register are loaded into SPI_RFIFO.
6. Whenever the initiating conditions are satisfied, the SPI continues to send and receive words. If the transmit buffer remains empty or the receive buffer remains full, the device operates according to the states of the SPI_TXCTL.TDU and SPI_RXCTL.RDO bits.
7. It is possible to program a secondary channel in addition to the initiating channel. This feature allows usage of available channel resources for receives or transmits simultaneously with the initiating channel.

Slave Operation in Non-DMA Modes

When a device is enabled as a slave in a non-DMA mode, a transition of the $\overline{\text{SPI\_SS}}$ select signal to the active state (low) triggers the the start of a transfer. Or, the first active edge of SPI_CLK triggers the start, depending on the state of SPI_CTL.CPHA bit. The interface operates in the following manner.

1. The core writes to the SPI_CTL, SPI_RXCTL, and SPI_TXCTL registers. The operation defines the mode of the serial link to be the same as the mode setup in the SPI master.
2. To prepare for the data transfer, the core writes data to be transmitted into SPI_TFIFO.
3. Once the $\overline{\text{SPI\_SS}}$ falling edge is detected, the slave starts sending data on active SPI_CLK edges and sampling data on inactive SPI_CLK edges.
4. Reception or transmission continues until $\overline{\text{SPI\_SS}}$ is released or until the slave has received the proper number of clock cycles.
5. The slave device continues to receive or transmit with each new falling edge transition on $\overline{\text{SPI\_SS}}$ or active SPI_CLK edge. If the transmit buffer remains empty or the receive buffer remains full, the device operates according to the states of the SPI_TXCTL.TDU and SPI_RXCTL.RDO bits.

Configuring DMA Master Mode

The SPI interface supports a write DMA channel and a read DMA channel. It can use these functions individually or in a lock-step manner in duplex mode (SPI_TXCTL.TTI= SPI_RXCTL.RTI=1).

1. Write to the appropriate DMA registers to enable the SPI DMA channel and to configure the necessary work units, access direction, word count, and so on.
2. Write to the SPI_SLVSEL register, setting one or more of the SPI flag select bits.
3. Write to the SPI_CLK and SPI_CTL registers, enabling the device as a master and configuring the SPI system by specifying the appropriate word length, transfer format, baud rate, and so forth.
4. Write to SPI_RXCTL to configure SPI master receive mode, or write to SPI_TXCTL to configure SPI master transmit mode.
5. Finally, write to the SPI_RXCTL.REN bit to enable the receive channel, or write to SPI_TXCTL.TEN to enable the transmit channel.
6. If the SPI_RXCTL.RTI bit is enabled, a receive transfer is initiated upon enabling SPI_CTL.EN bit. If the receive word counter is enabled (SPI_RXCTL.RWCEN, then the SPI_RWC register must be non-zero for a transfer to initiate.

   ADDITIONAL INFORMATION: If enabling both receive and transmit DMA channels, but not enabling SPI_TXCTL.TTI, write to the SPI_RXCTL register after writing the SPI_CTL and SPI_TXCTL registers. In this way, a transmit underrun can be prevented for the first transfer. Subsequent transfers are initiated as the SPI reads data from the receive shift register and writes to the SPI receive FIFO. The SPI then requests a write from DMA to memory. Upon a DMA grant, the DMA engine reads a word from the SPI receive FIFO and writes to memory. New requests continue to be initiated as long as the receive FIFO does not fill up, when SPI_RWC does not become zero while SPI_RXCTL.RWCEN=1.
7. If SPI_TXCTL.TTI is enabled, the SPI controller requests DMA reads from memory as long as there is space for more data in the transmit pipe. Upon a DMA grant, the DMA engine reads a word from memory and writes to the transmit FIFO. As long as transmit data is available in the FIFO, and the SPI_TWC register is non-zero when SPI_TXCTL.TWCEN=1, the SPI continues to initiate transfers until disabled.
8. If both the SPI_TXCTL.TTI and SPI_RXCTL.RTI bits are enabled, the SPI controller requests a DMA read from memory. However, there must be space for more data in the transmit pipe and the number of words written into the SPI must be less than SPI_TWC if SPI_TXCTL.TWCEN=1. Upon a DMA grant, the DMA engine reads a word from memory and writes to the transmit FIFO.

   ADDITIONAL INFORMATION: As the SPI writes data from the transmit FIFO into the transmit shift register, it initiates a transfer on the SPI link. Data received from the transfer is moved from the SPI receive shift register to the receive FIFO. The SPI controller requests a write from DMA to memory. Upon a DMA grant, the DMA engine reads a word from the receive FIFO and writes to memory. Transfer continues to be initiated as long as both receives and transmits can accommodate new data.

9. If the receive pipe fills up due to unavailability of DMA grants, the transmit pipe stalls until the pipe is drained. If the transmit pipe fills up, the SPI stops requesting for DMA writes. If the value in SPI_RWC expires, further write-requests to DMA stop. However, data already written into the transmit FIFO is sent, and read requests to DMA continue until the receive data is read from the receive FIFO.
10. The SPI then generates the programmed clock pulses on SPI_CLK and simultaneously shifts data out of SPI_MOSI while shifting data in from SPI_MISO. For receive transfers, the value in the shift register is loaded into the SPI_RFIFO register at the end of the transfer. For transmit transfers, the value in the SPI_TFIFO register is loaded into the shift register at the start of the transfer.

Configuring DMA Slave Mode Operation

This mode occurs when the SPI is enabled as a slave and the DMA engine is configured to transmit or receive data. A transition of the SPI_SS signal to the active-low state triggers the start of a transfer. Or, the first active edge of SPI_CLK triggers the start of a transfer, depending on the state of the SPI_CTL.CPHA bit. The following steps illustrate the SPI receive or transmit DMA sequence in an SPI slave (in response to a master command). The SPI supports a receive DMA channel and a transmit DMA channel.

1. Write to the appropriate DMA registers to enable the SPI DMA channel and configure the necessary work units, access direction, word count, and so on.
2. Write to the SPI_CTL, SPI_RXCTL, and SPI_TXCTL registers to define the mode of the serial link to be the same as the mode configured in the SPI master.
3. If the receive channel is enabled (SPI_RXCTL.REN is asserted), the following actions occur:
    a. Once the slave select input is active, the slave starts receiving and transmitting data on active SPI_CLK edges.
    b. The value in the shift register is loaded into the SPI_RFIFO register at the end of the transfer.
    c. Once SPI_RFIFO has valid data, it requests a write from DMA to memory.
    d. Upon a DMA grant, the DMA engine reads a word from the receive FIFO and writes to memory.
    e. As long as there is data in the receive FIFO, the SPI slave continues to request a DMA write to memory. The DMA engine continues to read a word from the FIFO and writes to memory until the SPI_RWC counts to zero. The SPI slave continues receiving words on active SPI_CLK edges as long as the SPI_SS input is active.
    f. If the data collected in the receive pipe breaches the set level, and the DMA engine cannot keep up with the receive rate, the slave can deassert the SPI_RDY signal. This signaling throttles the master. The receive pipe level is set according to the SPI_CTL.FCWM field. The signal is deasserted as the DMA drains the receive FIFO. Alternatively, the SPI can use the SPI_RXCTL.RDO bit to decide when the incoming data is discarded or overwritten into the receive FIFO (when SPI_CTL.FCEN is inactive).
4. If the transmit channel is enabled (SPI_TXCTL.TEN is asserted), the following actions occur:
    a. The SPI requests a DMA read from memory.

b. Upon a DMA grant, the DMA engine reads a word from memory and writes to the transmit FIFO.

c. The SPI then reads DMA data from the transmit FIFO and writes to the transmit shift register, awaiting the start of the next transfer.

d. Once the slave select input is active, the slave starts receiving and transmitting data on active SPI_CLK edges.

e. As long as there is room in the transmit FIFO, the SPI slave continues to request a DMA read from memory. The DMA engine continues to read a word from memory and write to the transmit FIFO until the SPI_TWC register value counts down to 0. The SPI slave continues transmitting words on active SPI_CLK edges as long as the SPI_SS input is active.

f. If the number of outstanding data entries in the transmit pipe breaches the level set and the DMA cannot keep up with the transmit rate, the slave de-asserts the SPI_RDY signal. This signaling throttles the master. The transmit pipe level is set according to the SPI_CTL.FCWM field. The signal is deasserted as the DMA fills the transmit FIFO. Alternately the SPI_TXCTL.TDU bit decides the state of the transmit data (when SPI_CTL.FCEN is deasserted).

5. If both receive and transmit channels are enabled, the following actions occur after the actions for each channel. Transfers continue as long as both receives and transmit channels can accommodate new data.

a. If the receive pipe fills up due to unavailability of DMA grant, the SPI interface stalls the master by asserting the SPI_RDY pin. This signal is deasserted as the DMA drains the receive FIFO. Alternately, the SPI uses the SPI_RXCTL.RDO bit to decide when the incoming data is discarded or overwritten into the receive FIFO (when SPI_CTL.FCEN is deasserted).

b. If the transmit pipe fills up, the SPI stops requesting for DMA writes until the pipe clears.

c. If there is an underflow problem in the transmit pipe, the slave stalls the master by deasserting SPI_RDY while DMA fills the transmit FIFO. Alternately, the SPI uses the SPI_TXCTL.TDU bit to decide the state of the transmit data (when SPI_CTL.FCEN is deasserted).

ADSP-CM40x SPI Register Descriptions

Serial Peripheral Interface (SPI) contains the following registers.

Table 24-15: ADSP-CM40x SPI Register List

Name	Description
SPI_CLK	Clock Rate Register
SPI_CTL	Control Register
SPI_DLY	Delay Register
SPI_ILAT	Masked Interrupt Condition Register
SPI_ILAT_CLR	Masked Interrupt Clear Register

Table 24-15: ADSP-CM40x SPI Register List (Continued)

Name	Description
SPI_IMSK	Interrupt Mask Register
SPI_IMSK_CLR	Interrupt Mask Clear Register
SPI_IMSK_SET	Interrupt Mask Set Register
SPI_MMRDH	Memory Mapped Read Header
SPI_MMTOP	SPI Memory Top Address
SPI_RFIFO	Receive FIFO Data Register
SPI_RWC	Received Word Count Register
SPI_RWCR	Received Word Count Reload Register
SPI_RXCTL	Receive Control Register
SPI_SLVSEL	Slave Select Register
SPI_STAT	Status Register
SPI_TFIFO	Transmit FIFO Data Register
SPI_TWC	Transmitted Word Count Register
SPI_TWCR	Transmitted Word Count Reload Register
SPI_TXCTL	Transmit Control Register

Clock Rate Register

The SPI_CLK register selects the baud rate for SPI data transfers, relating this rate to the SPI serial clock (SCK) and the system clock (SYSCLK).

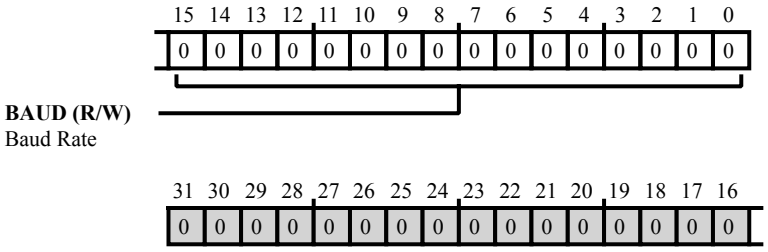

Figure 24-20: SPI_CLK Register Diagram

Table 24-16: SPI_CLK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	BAUD	Baud Rate. The SPI_CLK.BAUD bits set the SPI baud rate according to the formula: BAUD = (SYSCLK / SPI Clock) - 1

Control Register

The SPI_CTL register enables the SPI and configures settings for operating modes, communication protocols, and buffer operations.

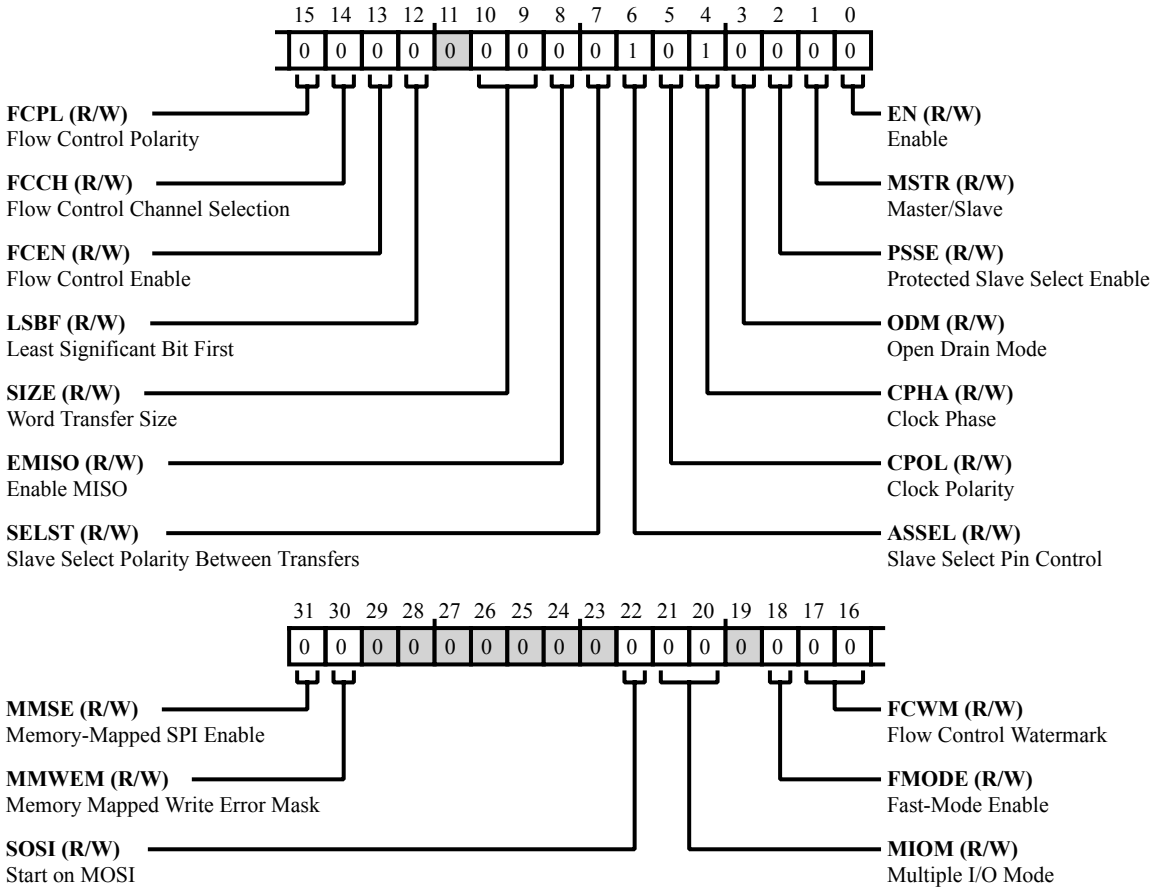

Figure 24-21: SPI_CTL Register Diagram

Table 24-17: SPI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	MMSE	Memory-Mapped SPI Enable . When the SPI_CTL.MMSE bit is asserted, communication to a SPI memory device is automated such that the memory it contains is accessible directly through the read of processor address space assigned to it (As far as the SPI peripheral is concerned, this includes all read accesses received by the SPI peripherals system crossbar slave port). Note that when memory-mapped access of SPI memory is enabled, attempts to communicate with the SPI device using legacy methods are blocked and receive fabric error responses are generated. Legacy methods include any direct access made to the TX and Rx FIFOs, whether initiated by DMA or processor MMR access. Blocked attempts also set the SPI_STAT.MMAE bit notification bit.	
		0	Hardware automated access of memory-mapped SPI memory disabled.
		1	Hardware-automated access of memory-mapped SPI memory enabled.
30 (R/W)	MMWEM	Memory Mapped Write Error Mask . The SPI_CTL.MMWEM bit specifies whether an error response is returned to the fabric upon write attempts to address space reserved for memory-mapped reads of SPI memory. Whether or not a write error response is masked by this bit, the associated sticky notification bit SPI_STAT.MMWE bit is still set.	
		0	Write error response returned upon write attempts to memory-mapped SPI memory
		1	Write error response masked (not returned) upon write attempts to memory-mapped SPI memory
22 (R/W)	SOSI	Start on MOSI . The SPI_CTL.SOSI bit is valid only when SPI_CTL.MIOM is enabled for either DIOM or QIOM, and this bit selects the starting pin and the bit placement on pins for these modes. In DIOM, by default (SPI_CTL.SOSI =0) SPI sends first bit on the SPI_MISO pin and second bit on the SPI_MOSI pin. In QIOM, by default, the SPI sends first bit on the SPI_D3 pin, second bit on the SPI_D2 pin, third bit on the SPI_MISO pin and fourth bit on the SPI_MOSI pin. This order can be reversed by setting the SPI_CTL.SOSI bit. When this bit is set, the SPI sends first bit on the SPI_MOSI pin. The first bit referred to here depends on the SPI_CTL.LSBF bit setting (MSB bit or LSB bit).	
		0	Start on MISO (DIOM) or start on SPI_D3
		1	Start on MOSI

Table 24-17: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/W)	MIOM	Multiple I/O Mode . The SPI_CTL.MIOM bits enable SPI operation in dual I/O mode (DIOM) or quad I/O mode (QIOM). These bits may only be changed when the SPI is disabled (SPI_CTL.EN =0).	
		0	No MIOM (disabled)
		1	DIOM operation
		2	QIOM operation
		3	Reserved
18 (R/W)	FMODE	Fast-Mode Enable. The SPI_CTL.FMODE bit enables fast mode operation for SPI receive transfers. SPI transmit operations in fast mode are the same as normal mode.	
		0	Disable
		1	Enable
17:16 (R/W)	FCWM	Flow Control Watermark. The SPI_CTL.FCWM bits select the watermark level of the transmit channel (SPI_TFIFO buffer) or receive channel (SPI_RFIFO buffer) that triggers flow control operation. These bits are applicable only when the SPI is a slave (SPI_CTL.MSTR = 0) and flow control is enabled (SPI_CTL.FCEN =1). When the watermark condition is met, the SPI slave de-asserts the SPI_RDY pin.	
		0	TFIFO empty or RFIFO full
		1	TFIFO 75% or more empty, or RFIFO 75% or more full
		2	TFIFO 50% or more empty, or RFIFO 50% or more full
		3	Reserved
15 (R/W)	FCPL	Flow Control Polarity. The SPI_CTL.FCPL bit selects flow control polarity for the SPI_RDY pin when flow control is enabled. When the SPI_RDY pin is active, the SPI is indicating it's ready for data transfer.	
		0	Active-low RDY
		1	Active-high RDY

Table 24-17: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	FCCH	Flow Control Channel Selection. The SPI_CTL.FCCH bit selects whether the SPI applies flow control to the transmit channel (SPI_TFIFO buffer) or receive channel (SPI_RFIFO buffer). This bit is applicable only when the SPI is a slave and flow control is enabled.	
		0	Flow control on RX buffer
		1	Flow control on TX buffer
13 (R/W)	FCEN	Flow Control Enable. The SPI_CTL.FCEN bit enables SPI flow control operation, which permits slow slave devices to interface with fast master devices. This bit controls operation of the SPI_RDY pin. Note that options for flow control operation are available using the SPI_CTL.FCCH, SPI_CTL.FCPL, and SPI_CTL.FCWM bits.	
		0	Disable
		1	Enable
12 (R/W)	LSBF	Least Significant Bit First. The SPI_CTL.LSBF bit selects whether the SPI transmits/receives data as LSB first (little endian) or MSB first (big endian). This bit may only be changed when the SPI is disabled.	
		0	MSB sent/received first (big endian)
		1	LSB sent/received first (little endian)
10:9 (R/W)	SIZE	Word Transfer Size. The SPI_CTL.SIZE bits select the SPI transfer word size as 8, 16 or 32 bits. To ensure correct operation, both the master and slave must be configured with the same word size. This bit may only be changed when the SPI is disabled (SPI_CTL.EN =0).	
		0	8-bit word
		1	16-bit word
		2	32-bit word
		3	Reserved
8 (R/W)	EMISO	Enable MISO. The SPI_CTL.EMISO bit enables master-in-slave-out (MISO) mode. This SPI mode is applicable only when the SPI is a slave.	
		0	Disable
		1	Enable

Table 24-17: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	SELST	Slave Select Polarity Between Transfers. The SPI_CTL.SELST bit selects the state (polarity) for the $\overline{\text{SPI\_SEL[n]}}$ pin in-between SPI transfers when the SPI is a master and hardware slave select assertion is enabled (SPI_CTL.ASSEL =1). In slave mode, this bit affects the detection of both transmit collision (SPI_STAT.TC and under-run (SPI_STAT.TUR) errors.	
		0	De-assert slave select (high)
		1	Assert slave select (low)
6 (R/W)	ASSEL	Slave Select Pin Control. The SPI_CTL.ASSEL bit selects whether the SPI hardware sets the $\overline{\text{SPI\_SEL[n]}}$ pin output value (ignoring the slave select SPI_SLVSEL.SSEL1 - $\overline{\text{SPI\_SLVSEL}}$.SSEL7 bits) or whether software control of the slave select bits set the $\overline{\text{SPI\_SEL[n]}}$ pin output value. This feature is applicable only when the SPI is a master. When hardware control is enabled, the $\overline{\text{SPI\_SEL[n]}}$ pin output is asserted during the transfers, and the pin polarity between transfers is selected by the SPI_CTL.SELST bit. When software control is enabled, the $\overline{\text{SPI\_SEL[n]}}$ pin output value is set through software control of the slave select bits, and as such, the pin may either remain asserted (low) or be deasserted between transfers.	
		0	Software Slave Select Control
		1	Hardware Slave Select Control
5 (R/W)	CPOL	Clock Polarity. The SPI_CTL.CPOL bit selects whether the SPI uses an active-low or active-high signal for the SPI clock (SPI_CLK). This bit works with the SPI_CTL.CPHA bit to select combinations of clock phase and polarity for the SPI_CLK pin. This bit may only be changed when the SPI is disabled.	
		0	Active-high SPI CLK
		1	Active-low SPI CLK
4 (R/W)	CPHA	Clock Phase. The SPI_CTL.CPHA bit selects whether the SPI starts toggling the signal for the SPI clock (SPI_CLK) from the start of the first data bit or from the middle of the first data bit. The SPI_CTL.CPHA bit works with the SPI_CTL.CPOL bit to select combinations of clock phase and polarity for the SPI_CLK pin. This bit may only be changed when the SPI is disabled.	
		0	SPI CLK toggles from middle
		1	SPI CLK toggles from start

Table 24-17: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	ODM	Open Drain Mode. The SPI_CTL.ODM bit configures the data output pins (SPI_MOSI and SPI_MISO) to behave as open drain outputs, which prevents contention and possible damage to pin drivers in multi-master or multi-slave SPI systems. When SPI_CTL.ODM is enabled and the SPI is a master, the SPI three-states the SPI_MOSI pin when the data driven out on MOSI is a logic-high. The SPI does not three-state the SPI_MOSI pin when the driven data is a logic-low. When SPI_CTL.ODM is enabled and the SPI is a slave, the SPI three-states the SPI_MISO pin when the data driven out on SPI_MISO is a logic-high. Note that an external pull-up resistor is required on both the SPI_MOSI and SPI_MISO pins when SPI_CTL.ODM is enabled.	
		0	Disable
		1	Enable
2 (R/W)	PSSE	Protected Slave Select Enable. The SPI_CTL.PSSE bit enables the $\overline{\text{SPI\_SS}}$ pin to provide error detection input in a multi-master environment when the SPI is in master mode. If some other device in the system asserts the $\overline{\text{SPI\_SS}}$ pin while SPI is enabled as master (and SPI_CTL.PSSE is enabled), this condition causes a mode fault error.	
		0	Disable
		1	Enable
1 (R/W)	MSTR	Master/Slave. The SPI_CTL.MSTR bit toggles the SPI between master mode and slave mode. This bit may only be changed when the SPI is disabled.	
		0	Slave
		1	Master
0 (R/W)	EN	Enable. The SPI_CTL.EN bit enables SPI operation.	
		0	Disable SPI module
		1	Enable

Delay Register

The SPI_DLY register selects a transfer delay and the lead/lag timing between slave select signals and SPI clock edge assertion/de-assertion.

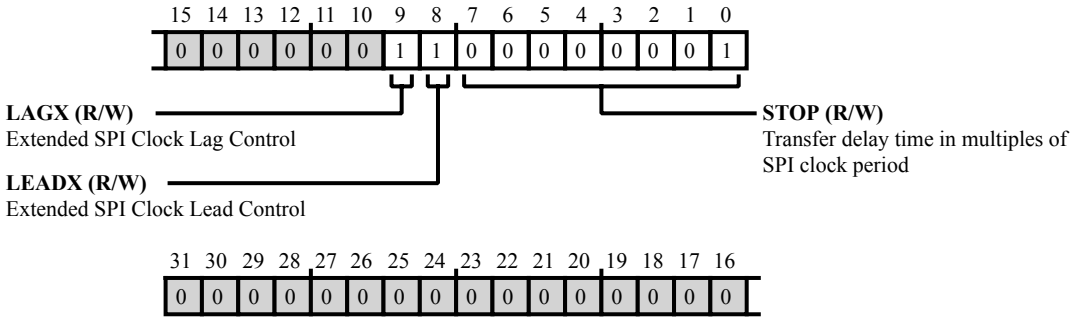

Figure 24-22: SPI_DLY Register Diagram

Table 24-18: SPI_DLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	LAGX	Extended SPI Clock Lag Control. The SPI_DLY.LAGX bit enables insertion of a 1-SPI_CLK cycle lag (extend lag) in the timing between the slave select (SPI_SEL[n]) assertion and first SPI Clock edge.	
		0	Disable
		1	Enable
8 (R/W)	LEADX	Extended SPI Clock Lead Control. The SPI_DLY.LEADX bit enables insertion of a 1-SPI_CLK cycle lead (extend lead) in the timing between the slave select (SPI_SEL[n]) de-assertion and last SPI Clock edge.	
		0	Disable
		1	Enable
7:0 (R/W)	STOP	Transfer delay time in multiples of SPI clock period. The SPI_DLY.STOP bits select a delay (number of stop bits in multiples of SPI Clock duration) at the end of each SPI transfer. The default delay is the minimum value required to comply with the SPI protocol (1-bit duration). The SPI_DLY.STOP bits may be programmed with smaller delay values, resulting in continuous operation (for example, stop bits =0).	

Masked Interrupt Condition Register

The SPI_ILAT register latches interrupts, queuing the interrupts for service. When a condition is indicated by a bit in the SPI_STAT register and the corresponding interrupt is unmasked in SPI_IMSK, the SPI latches the interrupt's bit in SPI_ILAT.

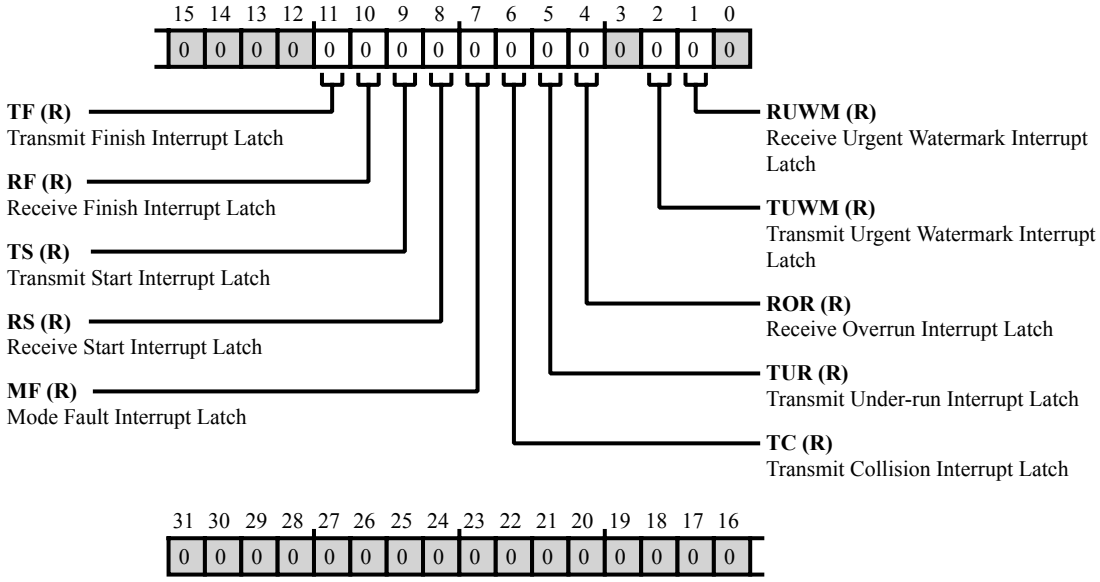

Figure 24-23: SPI_ILAT Register Diagram

Table 24-19: SPI_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/NW)	TF	Transmit Finish Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
10 (R/NW)	RF	Receive Finish Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
9 (R/NW)	TS	Transmit Start Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
8 (R/NW)	RS	Receive Start Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt

Table 24-19: SPI_ILAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	MF	Mode Fault Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
6 (R/NW)	TC	Transmit Collision Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
5 (R/NW)	TUR	Transmit Under-run Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
4 (R/NW)	ROR	Receive Overrun Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
2 (R/NW)	TUWM	Transmit Urgent Watermark Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
1 (R/NW)	RUWM	Receive Urgent Watermark Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt

Masked Interrupt Clear Register

The SPI_ILAT_CLR register permits clearing individual mask bits in the SPI_ILAT register without affecting other bits in the register. Use write-1-to-clear on a bit in SPI_ILAT_CLR to clear the corresponding bit in the SPI_ILAT register.

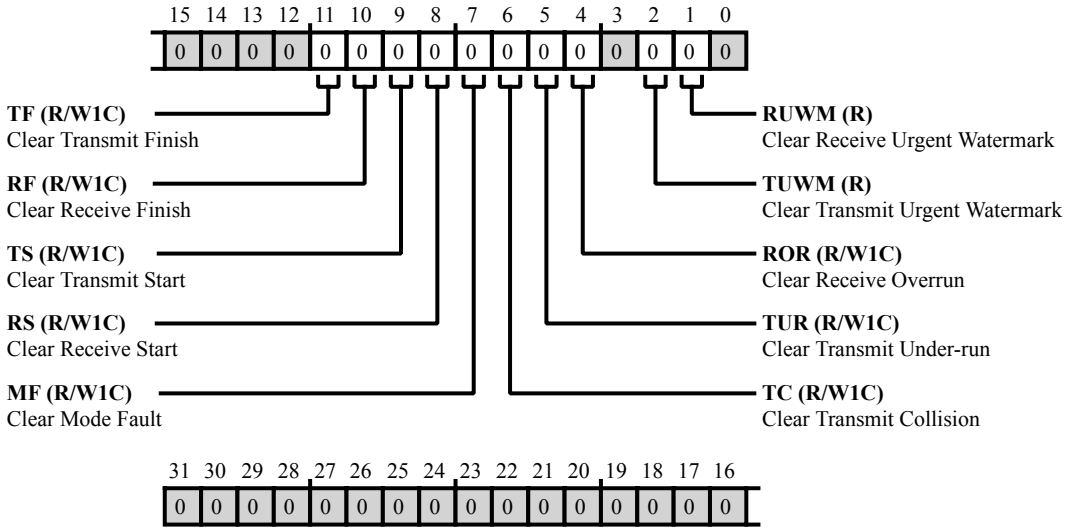

Figure 24-24: SPI_ILAT_CLR Register Diagram

Table 24-20: SPI_ILAT_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W1C)	TF	Clear Transmit Finish. The SPI_ILAT_CLR.TF bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit
10 (R/W1C)	RF	Clear Receive Finish. The SPI_ILAT_CLR.RF bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit
9 (R/W1C)	TS	Clear Transmit Start. The SPI_ILAT_CLR.TS bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit

Table 24-20: SPI_ILAT_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W1C)	RS	Clear Receive Start. The SPI_ILAT_CLR.RS bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit
7 (R/W1C)	MF	Clear Mode Fault. The SPI_ILAT_CLR.MF bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit
6 (R/W1C)	TC	Clear Transmit Collision. The SPI_ILAT_CLR.TC bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit
5 (R/W1C)	TUR	Clear Transmit Under-run. The SPI_ILAT_CLR.TUR bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit
4 (R/W1C)	ROR	Clear Receive Overrun. The SPI_ILAT_CLR.ROR bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit
2 (R/NW)	TUWM	Clear Transmit Urgent Watermark. The SPI_ILAT_CLR.TUWM bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit

Table 24-20: SPI_ILAT_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	RUWM	Clear Receive Urgent Watermark. The `SPI_ILAT_CLR.RUWM` bit clears the corresponding mask bit in the `SPI_ILAT` register.	
		0	No effect
		1	Clear mask bit

Interrupt Mask Register

The SPI_IMSK register unmasks (enables) or masks (disables) SPI interrupts. When a condition is indicated by a bit in the SPI_STAT register and the corresponding interrupt is unmasked in SPI_IMSK, the SPI latches the interrupt's bit in the SPI_ILAT register, queuing the interrupt for service.

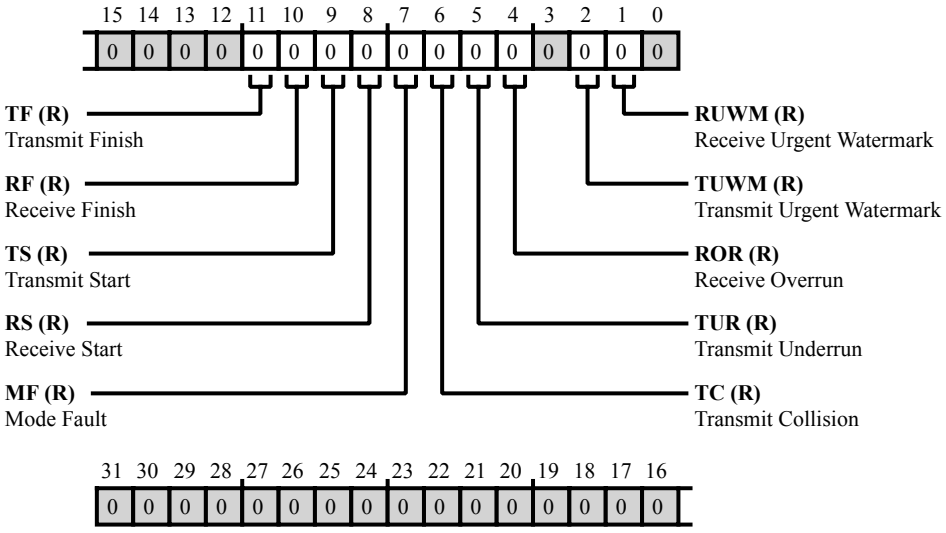

Figure 24-25: SPI_IMSK Register Diagram

Table 24-21: SPI_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/NW)	TF	Transmit Finish. The SPI_IMSK.TF bit unmasks (enables) or masks (disables) the TF interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
10 (R/NW)	RF	Receive Finish. The SPI_IMSK.RF bit unmasks (enables) or masks (disables) the RF interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
9 (R/NW)	TS	Transmit Start. The SPI_IMSK.TS bit unmasks (enables) or masks (disables) the TS interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt

Table 24-21: SPI_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	RS	Receive Start. The SPI_IMSK.RS bit unmasks (enables) or masks (disables) the RS interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
7 (R/NW)	MF	Mode Fault. The SPI_IMSK.MF bit unmasks (enables) or masks (disables) the MF interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
6 (R/NW)	TC	Transmit Collision. The SPI_IMSK.TC bit unmasks (enables) or masks (disables) the TC interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
5 (R/NW)	TUR	Transmit Underrun. The SPI_IMSK.TUR bit unmasks (enables) or masks (disables) the TUR interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
4 (R/NW)	ROR	Receive Overrun. The SPI_IMSK.ROR bit unmasks (enables) or masks (disables) the ROR interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
2 (R/NW)	TUWM	Transmit Urgent Watermark. The SPI_IMSK.TUWM bit unmasks (enables) or masks (disables) the TUWM interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
1 (R/NW)	RUWM	Receive Urgent Watermark. The SPI_IMSK.RUWM bit unmasks (enables) or masks (disables) the RUWM interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt

Interrupt Mask Clear Register

The SPI_IMSK_CLR register permits clearing individual mask bits in the SPI_IMSK register without affecting other bits in the register. Use write-1-to-clear on a bit in SPI_IMSK_CLR to clear the corresponding bit in the SPI_IMSK register.

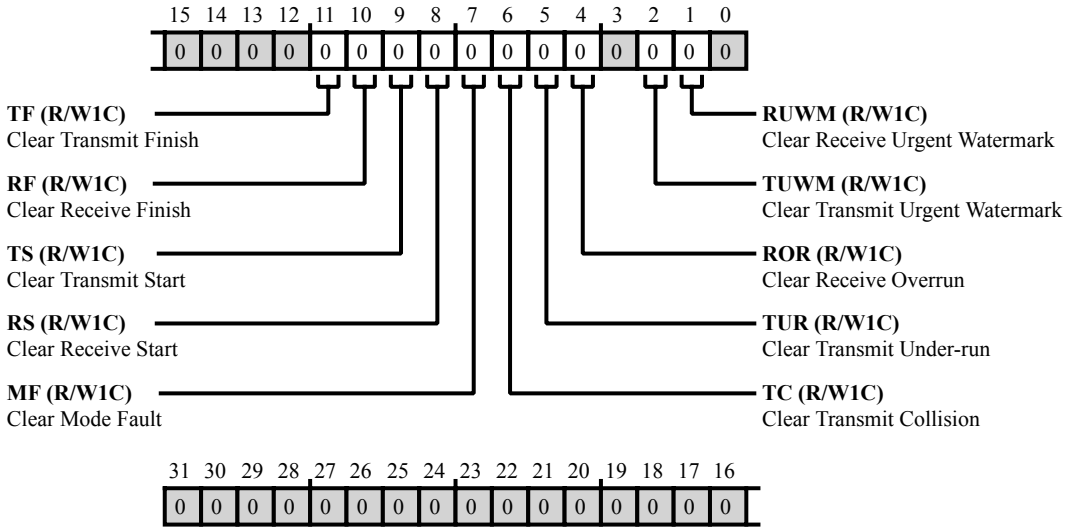

Figure 24-26: SPI_IMSK_CLR Register Diagram

Table 24-22: SPI_IMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W1C)	TF	Clear Transmit Finish. The SPI_IMSK_CLR.TF bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit
10 (R/W1C)	RF	Clear Receive Finish. The SPI_IMSK_CLR.RF bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit
9 (R/W1C)	TS	Clear Transmit Start. The SPI_IMSK_CLR.TS bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit

Table 24-22: SPI_IMSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W1C)	RS	Clear Receive Start. The SPI_IMSK_CLR.RS bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit
7 (R/W1C)	MF	Clear Mode Fault. The SPI_IMSK_CLR.MF bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit
6 (R/W1C)	TC	Clear Transmit Collision. The SPI_IMSK_CLR.TC bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit
5 (R/W1C)	TUR	Clear Transmit Under-run. The SPI_IMSK_CLR.TUR bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit
4 (R/W1C)	ROR	Clear Receive Overrun. The SPI_IMSK_CLR.ROR bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit
2 (R/W1C)	TUWM	Clear Transmit Urgent Watermark. The SPI_IMSK_CLR.TUWM bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit

Table 24-22: SPI_IMSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	RUWM	Clear Receive Urgent Watermark. The `SPI_IMSK_CLR.RUWM` bit clears the corresponding mask bit in the `SPI_IMSK` register.	
		0	No effect
		1	Clear mask bit

Interrupt Mask Set Register

The SPI_IMSK_SET register permits setting individual mask bits in the SPI_IMSK register without affecting other bits in the register. Use write-1-to-set on a bit in SPI_IMSK_SET to set the corresponding bit in the SPI_IMSK register.

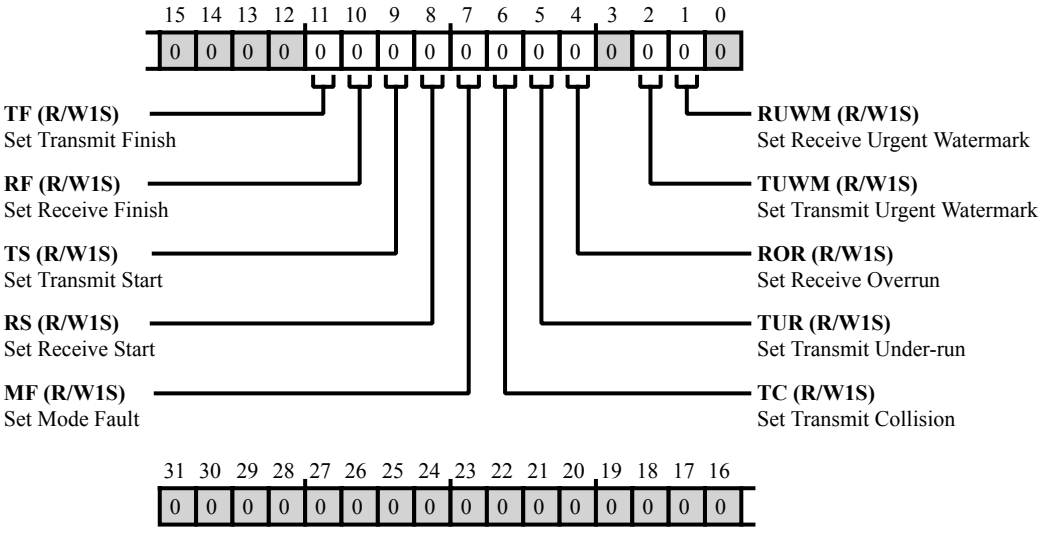

Figure 24-27: SPI_IMSK_SET Register Diagram

Table 24-23: SPI_IMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W1S)	TF	Set Transmit Finish. The SPI_IMSK_SET.TF bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit
10 (R/W1S)	RF	Set Receive Finish. The SPI_IMSK_SET.RF bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit
9 (R/W1S)	TS	Set Transmit Start. The SPI_IMSK_SET.TS bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit

Table 24-23: SPI_IMSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W1S)	RS	Set Receive Start. The SPI_IMSK_SET.RS bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit
7 (R/W1S)	MF	Set Mode Fault. The SPI_IMSK_SET.MF bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit
6 (R/W1S)	TC	Set Transmit Collision. The SPI_IMSK_SET.TC bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit
5 (R/W1S)	TUR	Set Transmit Under-run. The SPI_IMSK_SET.TUR bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit
4 (R/W1S)	ROR	Set Receive Overrun. The SPI_IMSK_SET.ROR bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit
2 (R/W1S)	TUWM	Set Transmit Urgent Watermark. The SPI_IMSK_SET.TUWM bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit

Table 24-23: SPI_IMSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1S)	RUWM	Set Receive Urgent Watermark. The `SPI_IMSK_SET.RUWM` bit sets the corresponding mask bit in the `SPI_IMSK` register.	
		0	No effect
		1	Set mask bit

Memory Mapped Read Header

The SPI_MMRDH register enables the use of memory-mapped mode. This mode allows direct memory-mapped read accesses of a SPI memory device and is primarily used to directly execute instructions from a SPI FLASH memory without using a low-level software driver. All overhead tasks such as transmission of the read header, pin turnaround timing and receive data sizing are handled in hardware.

The memory-mapped access mode is enabled by setting the SPI_CTL.MMSE bit. The features within the SPI_MMRDH register include a command skip mode, variable length byte addressing, and independent multi-pin support for command transmission, address transmission and data reception. In addition the command op-code and mode bytes are fully programmable.

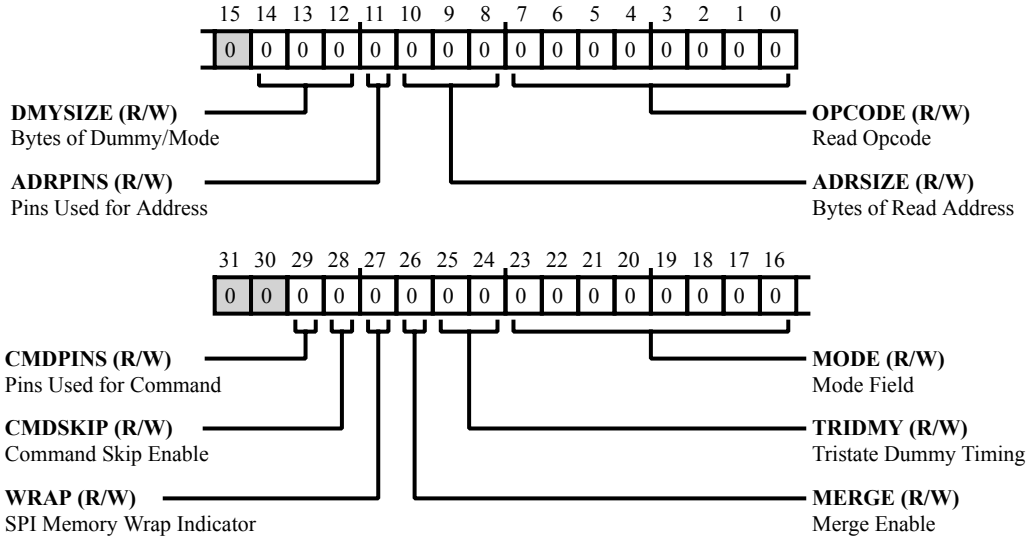

Figure 24-28: SPI_MMRDH Register Diagram

Table 24-24: SPI_MMRDH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/W)	CMDPINS	Pins Used for Command. The SPI_MMRDH.CMDPINS bit specifies the number of pins to be used for command transmission. This bit must be set consistent with expectations established by read op-code. Hardware does not interpret the OPCODE, but rather relies on this bit to specify behavior. When cleared, overrides SPI_CTL.MIOM bits). When set, uses bits specified by SPI_CTL.MIOM bit setting.	
		0	Use only one pin: MOSI (overrides SPI_CTL.MIOM bits in register SPI_CTL)
		1	Use pins specified by SPI_CTL.MIOM bits in register SPI_CTL

Table 24-24: SPI_MMRDH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
28 (R/W)	CMDSKIP	Command Skip Enable. The SPI_MMRDH.CMDSKIP bit enables command skip mode where the address is sent first and the OPCODE field is not sent (SPI_MMRDH.CMDSKIP bit =1). This mode is useful for supporting XIP (Execute-In-Place) operation where only the address is sent and the same read command is assumed. The SPI flash device must be primed with an initial read command, before the SPI_MMRDH.CMDSKIP bit is set.	
		0	OPCODE field is sent first followed by address
		1	OPCODE field is not sent; address is sent first
27 (R/W)	WRAP	SPI Memory Wrap Indicator. The SPI_MMRDH.WRAP bit must be set by software if software places a connected SPI memory device into a 8-byte, 16-byte or 32-byte wrap mode based on the ILINE and DLINE field setting of the cache configuration register address wrap mode. Software achieves this by transmitting a vendor specified command to the SPI memory device while the SPI_CTL.MMSE bit =0. If the SPI_MMRDH.WRAP bit =1, the SPI does not need to deassert the SPI slave select signal and resend the read header in order to wrap to the cache line base when servicing misaligned cache fill requests. Although this improves cache fill efficiency, it requires that the SPI deassert the SPI slave select pin and resend the read header whenever a DMA burst requests crosses 32 byte alignments. Setting this bit improves cache throughput but decreases DMA throughput.	
		0	SPI Memory auto increments address purely sequentially
		1	SPI Memory auto increments address but wraps within 32 Byte lines
26 (R/W)	MERGE	Merge Enable. When the SPI_MMRDH.MERGE bit is set, SPI hardware combines the two successive transfers. This increases the throughput rate when accessing a large number of sequential memory locations. For more information refer to the Merged Read Accesses section.	

Table 24-24: SPI_MMRDH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
25:24 (R/W)	TRIDMY	Tristate Dummy Timing. The SPI_MMRDH.TRIDMY bits specify whether and when output pins are three-stated during the interval of time specified by the SPI_MMRDH.DMYSIZE bits. Output pins potentially three-stated include all pins which were used to transmit the address	
		0	Tristate outputs immediately
		1	Tristate outputs after 4 bits of dummy/mode are transmitted
		2	Tristate outputs after 8 bits of dummy/mode are transmitted
		3	Never Tristate outputs never (previously specified output state is held)
23:16 (R/W)	MODE	Mode Field. These bits specify up to a leading byte to be transmitted during the interval of time specified by the SPI_MMRDH.DMYSIZE bit field. This first byte, or a portion of it, is interpreted as mode bits when certain opcodes are used in conjunction with certain SPI memory device. Mode bits are sent using the same number of pins which were used to transmit the address. Once sent, output pins will be held in their final resultant state until the conclusion of all dummy byte periods, unless three-stating the outputs is specified first by the SPI_MMRDH.TRIDMY bits.	
14:12 (R/W)	DMYSIZE	Bytes of Dummy/Mode. The SPI_MMRDH.DMYSIZE bit field specifies the number of bytes separating address transmission and read data return. Dummy bytes elapse assuming dummy bits are transmitted using the same number of pins which were used to transmit address.	
		0	0 Bytes
		1	1 Bytes
		2	2 Bytes
		3	3 Bytes
		4	4 Bytes
		5	5 Bytes
		6	6 Bytes
		7	7 Bytes

Table 24-24: SPI_MMRDH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	ADRPINS	Pins Used for Address. The SPI_MMRDH.ADRPINS bit specifies the number of pins to be used for address transmission. This bit must be set consistent with expectations established by read opcode. Hardware does not interpret the OPCODE, but rather relies on this bit to specify behavior.	
		0	Use only one pin: MOSI (overrides SPI_CTL.MIOM bits)
		1	Use pins specified by SPI_CTL.MIOM bits
10:8 (R/W)	ADRSIZE	Bytes of Read Address. The SPI_MMRDH.ADRSIZE bit field defines the number of bytes used to specify the read address. The read address is sent immediately following the transmission of opcode. Unlike opcode bits, address bits may be sent using either one or multiple pins. The number of pins is selected using the SPI_MMRDH.ADRPINS bit. The address sent to a connected SPI memory device is an echo of the read address received by the SPI peripheral slave port. Least significant bytes of address are sent when entire address is not sent.	
		0	1 Byte
		1	1 Byte
		2	2 Bytes
		3	3 Bytes
		4	4 Bytes
7:0 (R/W)	OPCODE	Read Opcode. The SPI_MMRDH.OPCODE bit field specifies initial bits transmitted in response to a read request of SPI memory. Although any opcode may be sent, values 0x03, 0x0B, 0x3B, 0x6B, 0xBB, and 0xEB are likely to be the most commonly used. Opcode is sent by the SPI without interpretation; the states of these bits have no affect beyond specifying what is initially shirted across the SPI interface.	

SPI Memory Top Address

The SPI_MMTOP register specifies the top populated address of a connected SPI memory device.

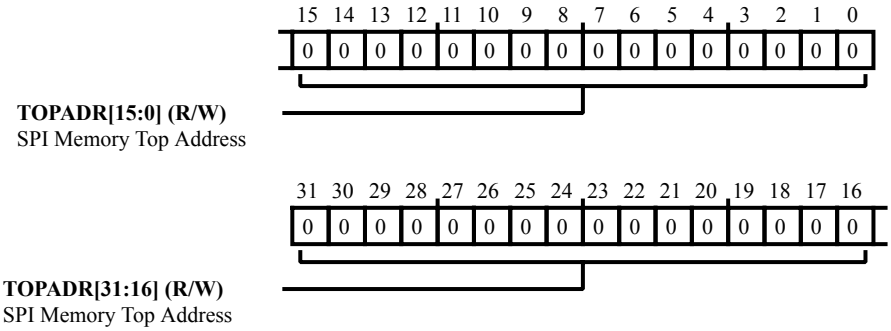

Figure 24-29: SPI_MMTOP Register Diagram

Table 24-25: SPI_MMTOP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TOPADR	SPI Memory Top Address. The SPI_MMTOP.TOPADR bit field specifies the top populated address of a connected SPI memory device. Attempts to access SPI memory are not blocked if this address is exceeded and an error is generated as part of the read response.

Receive FIFO Data Register

The SPI_RFIFO register has an interface to the receive shift register in the SPI and has an interface to the processor's data buses. The top level of the buffer is visible to programs as the 32-bit SPI_RFIFO register, but the size (number of word locations) of the receive FIFO is actually flexible with transfer word size. The size of the receive FIFO is 8 if word size is 8-bit, or the size is 4 if word size is 16-bit, or the size is 2 if word size is 32-bit.

Both masters and slaves may stop or stall receive transfers based on FIFO status. When the receive FIFO is full, the SPI master stops initiating new transfers on the SPI if SPI_RXCTL.RTI is enabled. A slave may stall the SPI interface when the content of the FIFO crosses the selected watermark. If data reception continues after SPI_RFIFO is full, the data in the receive FIFO is invalid, and the SPI indicates this condition with receive overrun (SPI_STAT.ROR). This condition is possible when SPI_RXCTL.RTI =0 and SPI_RXCTL.REN =1 for a master, or for a slave that does not exercise flow control.

Note that the receive FIFO is reset (cleared) when the SPI is disabled after being enabled.

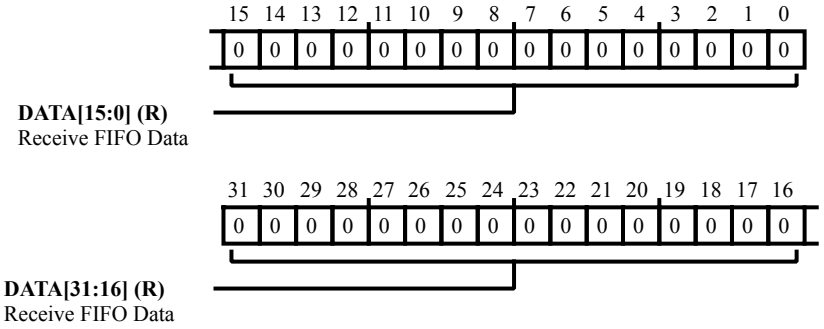

Figure 24-30: SPI_RFIFO Register Diagram

Table 24-26: SPI_RFIFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	DATA	Receive FIFO Data. The SPI_RFIFO.DATA bit field contains the FIFO receive data.

Received Word Count Register

The SPI_RWC register holds a count of the number of words remaining to be received by the SPI. To start the decrement of the word count in SPI_RWC, enable the receive word counter (SPI_RXCTL.RWCEN =1). The SPI uses the word count to control the duration of transfers and to signal the completion of a burst of transfers with the receive finish interrupt (SPI_ILAT.RF). In DMA mode, the SPI uses the SPI_RWC to ensure that the number of frames received during a DMA transfer is equal to the number of words programmed in the DMA channel controller. The values programmed into the SPI_RWC registers should match the word count in the DMA configuration. The SPI_RWC maintains the number of frames to be received in a transfer. The SPI_RWC should only be changed when the counter is disabled.

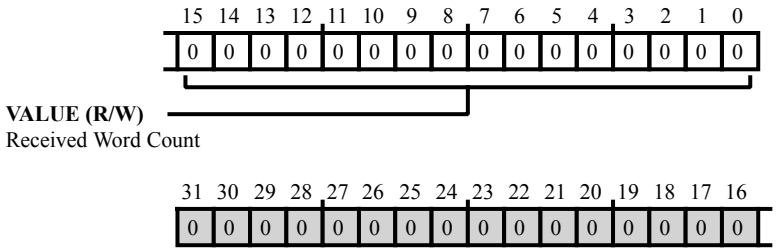

Figure 24-31: SPI_RWC Register Diagram

Table 24-27: SPI_RWC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Received Word Count. The SPI_RWC.VALUE bits hold the receive transfer word count.

Received Word Count Reload Register

The SPI_RWCR register holds the receive word count value that the SPI loads into the SPI_RWC register when the transfer count decrements to zero. To prevent the SPI from reloading the counter, use zero for the reload count value. The SPI_RWCR should only be changed when the counter is disabled.

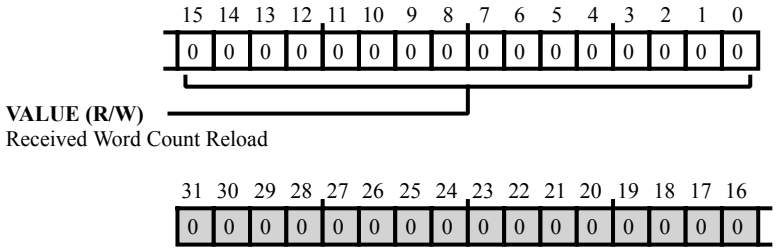

Figure 24-32: SPI_RWCR Register Diagram

Table 24-28: SPI_RWCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Received Word Count Reload. The SPI_RWCR.VALUE bits hold the receive transfer word count reload value.

Receive Control Register

The SPI_RXCTL register enables the SPI receive channel, initiates receive transfers, and configures SPI_RFIFO buffer watermark settings.

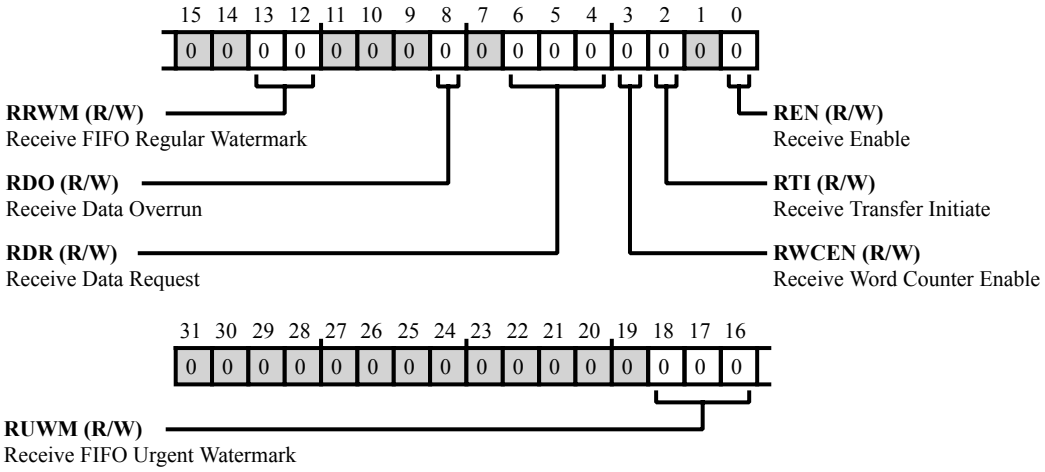

Figure 24-33: SPI_RXCTL Register Diagram

Table 24-29: SPI_RXCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	RUWM	Receive FIFO Urgent Watermark. The SPI_RXCTL.RUWM bits select the receive FIFO (SPI_RFIFO) watermark level for urgent data bus requests. The SPI also uses this watermark level for generation of the SPI_ILAT.RUWM interrupt. When an urgent SPI_RFIFO watermark is enabled with SPI_RXCTL.RUWM, the SPI_RXCTL.RRWM selection is used as the de-assertion condition for any SPI_ILAT.RUWM interrupts that are latched.	
		0	Disabled
		1	25% full RFIFO
		2	50% full RFIFO
		3	75% full RFIFO
		4	Full RFIFO
		5	Reserved
		6	Reserved
		7	Reserved

Table 24-29: SPI_RXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13:12 (R/W)	RRWM	Receive FIFO Regular Watermark. The SPI_RXCTL.RRWM bits select the receive FIFO (SPI_RFIFO) watermark level for regular data bus requests. When an urgent SPI_RFIFO watermark is enabled with SPI_RXCTL.RUWM, the SPI_RXCTL.RRWM selection is used as the de-assertion condition for any SPI_ILAT.RUWM interrupts that are latched.	
		0	Empty RFIFO
		1	RFIFO less than 25% full
		2	RFIFO less than 50% full
		3	RFIFO less than 75% full
8 (R/W)	RDO	Receive Data Overrun. The SPI_RXCTL.RDO bit selects handling for receive data requests when the receive buffer (SPI_RFIFO) is full. If enabled and SPI_RFIFO is full, the SPI overwrites old data in the buffer with incoming data. If disabled and SPI_RFIFO is full, the SPI keeps old data in the buffer and discards incoming data.	
		0	Discard incoming data if SPI_RFIFO is full
		1	Overwrite old data if SPI_RFIFO is full
6:4 (R/W)	RDR	Receive Data Request. The SPI_RXCTL.RDR bits select receive FIFO (SPI_RFIFO) watermark conditions that direct the SPI to generate a receive data request.	
		0	Disabled
		1	Not empty RFIFO
		2	25% full RFIFO
		3	50% full RFIFO
		4	75% full RFIFO
		5	Full RFIFO
		6	Reserved
		7	Reserved
3 (R/W)	RWCEN	Receive Word Counter Enable. The SPI_RXCTL.RWCEN bit enables the decrement of the SPI_RWC register when the count is not zero and SPI_RXCTL.RTI is enabled. Enabling SPI_RXCTL.RWCEN prevents receive overrun errors from occurring. The SPI_RXCTL.RWCEN bit is valid only when the SPI is a master.	
		0	Disable
		1	Enable

Table 24-29: SPI_RXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	RTI	Receive Transfer Initiate. The SPI_RXCTL.RTI bit enables initiation of receive transfers if the receive FIFO (SPI_RFIFO) is not full. The bit also enables this initiation if SPI_RWC is not zero when SPI_RXCTL.RWCEN is enabled. Enabling SPI_RXCTL.RTI prevents receive overrun errors from occurring. The SPI_RXCTL.RTI bit is valid only when the SPI is a master.	
		0	Disable
		1	Enable
0 (R/W)	REN	Receive Enable. The SPI_RXCTL.REN bit enables SPI receive channel operation.	
		0	Disable
		1	Enable

Slave Select Register

The SPI_SLVSEL register enables the SPI_SEL[n] pins for input and indicates the state (high or low) of these pins when enabled.

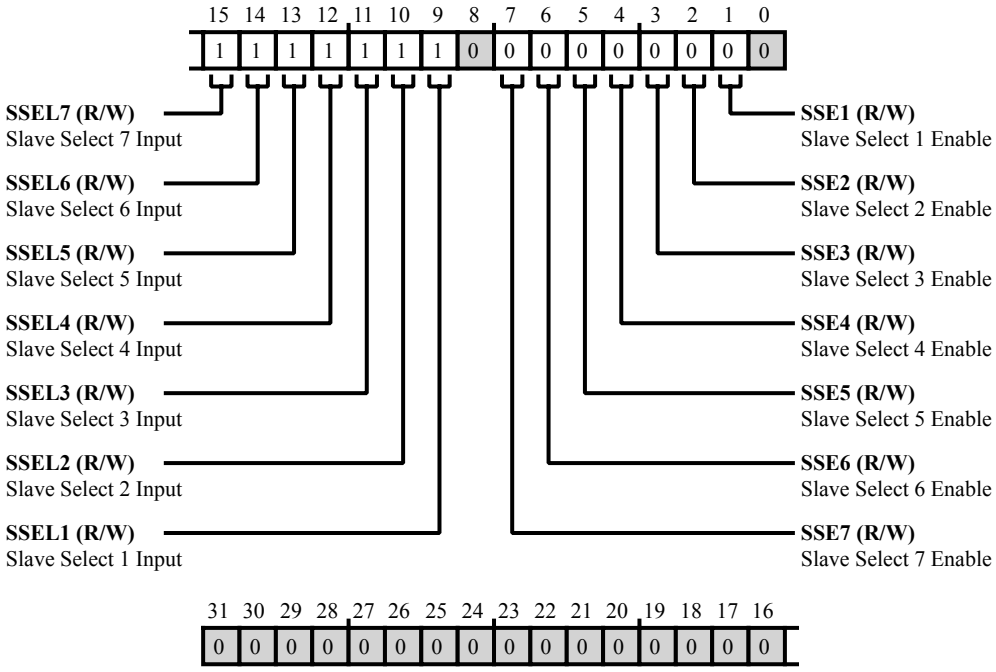

Figure 24-34: SPI_SLVSEL Register Diagram

Table 24-30: SPI_SLVSEL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	SSEL7	Slave Select 7 Input. The SPI_SLVSEL.SSEL7 bit state indicates the value on the related SPI_SEL[n] pin.	
		0	Low
		1	High
14 (R/W)	SSEL6	Slave Select 6 Input. The SPI_SLVSEL.SSEL6 bit state indicates the value on the related SPI_SEL[n] pin.	
		0	Low
		1	High

Table 24-30: SPI_SLVSEL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	SSEL5	Slave Select 5 Input. The SPI_SLVSEL.SSEL5 bit state indicates the value on the related $\overline{\texttt{SPI\_SEL[n]}}$ pin.	
		0	Low
		1	High
12 (R/W)	SSEL4	Slave Select 4 Input. The SPI_SLVSEL.SSEL4 bit state indicates the value on the related $\overline{\texttt{SPI\_SEL[n]}}$ pin.	
		0	Low
		1	High
11 (R/W)	SSEL3	Slave Select 3 Input. The SPI_SLVSEL.SSEL3 bit state indicates the value on the related $\overline{\texttt{SPI\_SEL[n]}}$ pin.	
		0	Low
		1	High
10 (R/W)	SSEL2	Slave Select 2 Input. The SPI_SLVSEL.SSEL2 bit state indicates the value on the related $\overline{\texttt{SPI\_SEL[n]}}$ pin.	
		0	Low
		1	High
9 (R/W)	SSEL1	Slave Select 1 Input. The SPI_SLVSEL.SSEL1 bit state indicates the value on the related $\overline{\texttt{SPI\_SEL[n]}}$ pin.	
		0	Low
		1	High
7 (R/W)	SSE7	Slave Select 7 Enable. The SPI_SLVSEL.SSE7 bit enables the related $\overline{\texttt{SPI\_SEL[n]}}$ pin for input. If disabled, the SPI three-states the related $\overline{\texttt{SPI\_SEL[n]}}$ pin. When the SPI is a slave, the master (not the SPI) asserts the input during the transfer. The input may be de-asserted or remain asserted between transfers. While the input is de-asserted, the SPI ignores SPI Clock, ignores inputs, and three-states outputs.	
		0	Disable
		1	Enable

Table 24-30: SPI_SLVSEL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	SSE6	Slave Select 6 Enable. The SPI_SLVSEL.SSE6 bit enables the related $\overline{\texttt{SPI\_SEL[n]}}$ pin for input. See the SPI_SLVSEL.SSE7 bit description for more information.	
		0	Disable
		1	Enable
5 (R/W)	SSE5	Slave Select 5 Enable. The SPI_SLVSEL.SSE5 bit enables the related $\overline{\texttt{SPI\_SEL[n]}}$ pin for input. See the SPI_SLVSEL.SSE7 bit description for more information.	
		0	Disable
		1	Enable
4 (R/W)	SSE4	Slave Select 4 Enable. The SPI_SLVSEL.SSE4 bit enables the related $\overline{\texttt{SPI\_SEL[n]}}$ pin for input. See the SPI_SLVSEL.SSE7 bit description for more information.	
		0	Disable
		1	Enable
3 (R/W)	SSE3	Slave Select 3 Enable. The SPI_SLVSEL.SSE3 bit enables the related $\overline{\texttt{SPI\_SEL[n]}}$ pin for input. See the SPI_SLVSEL.SSE7 bit description for more information.	
		0	Disable
		1	Enable
2 (R/W)	SSE2	Slave Select 2 Enable. The SPI_SLVSEL.SSE2 bit enables the related $\overline{\texttt{SPI\_SEL[n]}}$ pin for input. See the SPI_SLVSEL.SSE7 bit description for more information.	
		0	Disable
		1	Enable
1 (R/W)	SSE1	Slave Select 1 Enable. The SPI_SLVSEL.SSE1 bit enables the related $\overline{\texttt{SPI\_SEL[n]}}$ pin for input. See the SPI_SLVSEL.SSE7 bit description for more information.	
		0	Disable
		1	Enable

Status Register

The SPI_STAT register indicates SPI status including FIFO status, error conditions, and interrupt conditions. When an interrupt condition from this register is unmasked (enabled) by the corresponding bit in the SPI_IMSK register, the interrupt is latched into the corresponding bit in the SPI_ILAT register.

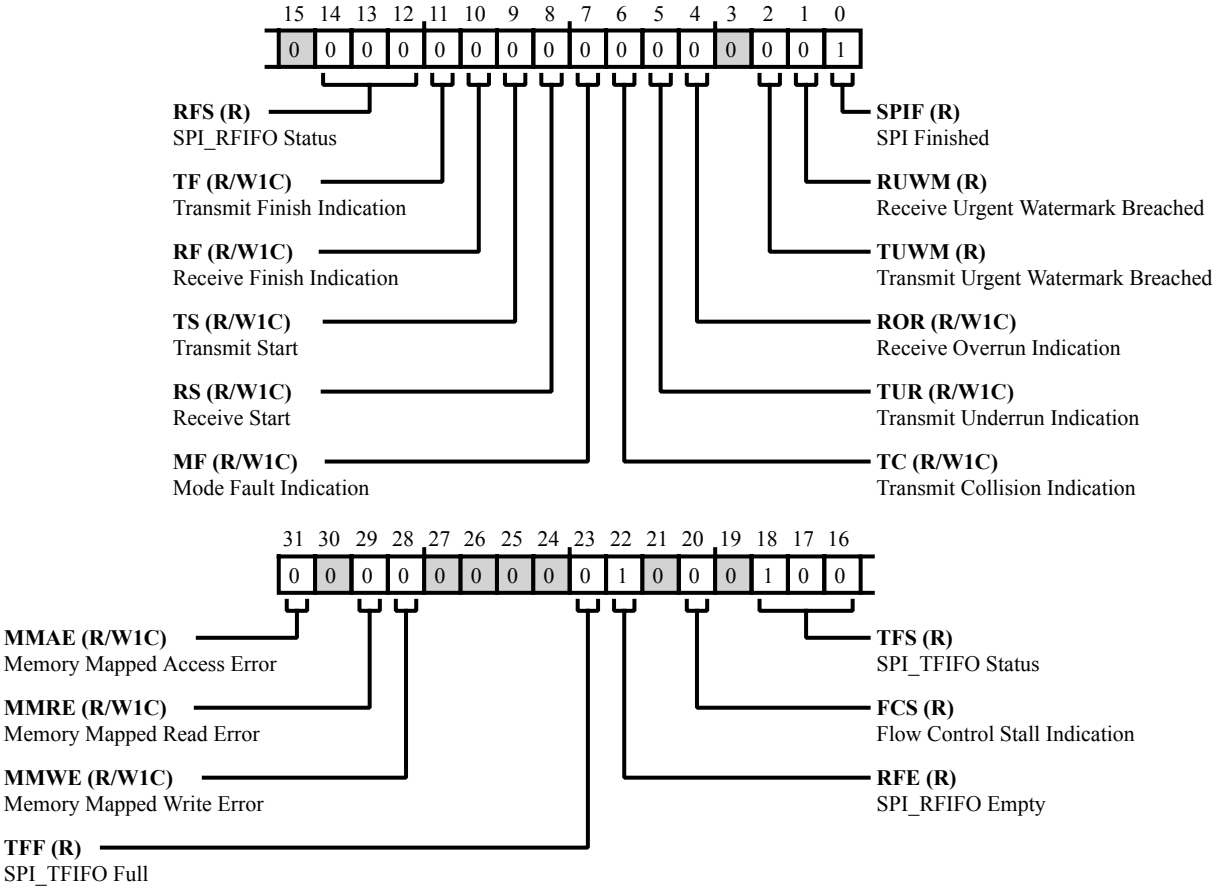

Figure 24-35: SPI_STAT Register Diagram

Table 24-31: SPI_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	MMAE	Memory Mapped Access Error. The SPI_STAT.MMAE bit =1 if an attempt is made to access either the Tx or Rx FIFO while memory-mapped access of SPI memory is enabled (see the SPI_CTL.MMSE) bit. The SPI_STAT.MMAE bit =0 when a 1 is written to it. The SPI_STAT.MMAE bit is provided for software notification only. Its state has no further effect.
29 (R/W1C)	MMRE	Memory Mapped Read Error.

Table 24-31: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		The SPI_STAT.MMRE bit =1 if an attempt is made to read address space reserved for memory-mapped SPI memory while memory mapping is disabled (see the SPI_CTL.MMSE bit. The SPI_STAT.MMRE bit =0 when a 1 is written to it. This bit is provided for software notification only. Its state has no further effect.	
28 (R/W1C)	MMWE	Memory Mapped Write Error. The SPI_STAT.MMWE bit =1 if an attempt is made to write address space reserved for memory-mapped SPI memory. The SPI_STAT.MMWE bit =0 when a 1 is written to it. This bit is provided for software notification only. Its state has no further effect.	
23 (R/NW)	TFF	SPI_TFIFO Full. The SPI_STAT.TFF bit indicates whether the SPI_TFIFO is full or not full.	
		0	Not full Tx FIFO
		1	Full Tx FIFO
22 (R/NW)	RFE	SPI_RFIFO Empty. The SPI_STAT.RFE bit indicates whether the SPI_RFIFO is empty or not empty.	
		0	RX FIFO not empty
		1	RX FIFO empty
20 (R/NW)	FCS	Flow Control Stall Indication. The SPI_STAT.FCS bit indicates whether a slave has de-asserted the SPI_RDY pin to stall the SPI master while the slave is unable to service the SPI masters request. This bit is valid only when the SPI is a master (SPI_CTL.MSTR =1 and flow control is enabled (SPI_CTL.FCEN =1).	
		0	No Stall (RDY pin asserted)
		1	Stall (RDY pin de-asserted)
18:16 (R/NW)	TFS	SPI_TFIFO Status. The SPI_STAT.TFS bits indicate the status of the SPI_TFIFO. The SPI uses this status when evaluating transmit watermark conditions.	
		0	Full TFIFO
		1	25% empty TFIFO
		2	50% empty TFIFO
		3	75% empty TFIFO
		4	Empty TFIFO
14:12 (R/NW)	RFS	SPI_RFIFO Status. The SPI_STAT.RFS bits indicate the status of the SPI_RFIFO. The SPI uses this status when evaluating receive watermark conditions.	

Table 24-31: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	Empty RFIFO
		1	25% full RFIFO
		2	50% full RFIFO
		3	75% full RFIFO
		4	Full RFIFO
11 (R/W1C)	TF	Transmit Finish Indication. The SPI_STAT.TF bit indicates that the SPI has detected the finish of a transmit burst transfer (the SPI_TWC count decrements to zero). This condition can only occur when SPI_TXCTL.TTI and SPI_TXCTL.TWCEN are enabled.	
		0	No status
		1	Transmit finish detected
10 (R/W1C)	RF	Receive Finish Indication. The SPI_STAT.RF bit indicates that the SPI has detected the finish of a receive burst transfer (the SPI_RWC count decrements to zero). This condition can only occur when SPI_RXCTL.RTI and SPI_RXCTL.RWCEN are enabled.	
		0	No status
		1	Receive finish detected
9 (R/W1C)	TS	Transmit Start. The SPI_STAT.TS bit indicates that the SPI has detected the start of a transmit burst transfer. A transmit bursts starts with the load of SPI_TWC from the SPI_TWCR. This condition can only occur when SPI_TXCTL.TTI and SPI_TXCTL.TWCEN are enabled.	
		0	No status
		1	Transmit start detected
8 (R/W1C)	RS	Receive Start. The SPI_STAT.RS bit indicates that the SPI has detected the start of a receive burst transfer. A receive bursts starts with the load of SPI_RWC from the SPI_RWCR. This condition can only occur when SPI_RXCTL.RTI and SPI_RXCTL.RWCEN are enabled.	
		0	No status
		1	Receive start detected

Table 24-31: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	MF	Mode Fault Indication. The SPI_STAT.MF bit, when SPI is a master and SPI_CTL.PSSE is enabled, indicates that multiple masters have asserted slave select inputs.	
		0	No status
		1	Mode fault occurred
6 (R/W1C)	TC	Transmit Collision Indication. The SPI_STAT.TC bit, when SPI is a slave, indicates that the load of data into the shift register has occurred too close to the first transmitting edge of the SPI Clock.	
		0	No status
		1	Transmit collision occurred
5 (R/W1C)	TUR	Transmit Underrun Indication. The SPI_STAT.TUR bit, when the transmit FIFO (SPI_TFIFO is empty, indicates that the last word in the transmit FIFO has been re-sent as transmit data. Alternately, it indicates that zero has been sent as transmit data.	
		0	No status
		1	Transmit underrun occurred
4 (R/W1C)	ROR	Receive Overrun Indication. The SPI_STAT.ROR bit, when the receive FIFO (SPI_RFIFO is full, indicates that a word in the receive FIFO has been overwritten with incoming receive data. Alternately, it indicates that incoming receive data has been discarded.	
		0	No status
		1	Receive overrun occurred
2 (R/NW)	TUWM	Transmit Urgent Watermark Breached. The SPI_STAT.TUWM bit indicates that the transmit urgent watermark (SPI_TXCTL.TUWM) has been reached. This condition is cleared when the transmit FIFO fills enough to reach the transmit regular watermark (SPI_TXCTL.TRWM).	
		0	TX Regular Watermark reached
		1	TX Urgent Watermark breached
1 (R/NW)	RUWM	Receive Urgent Watermark Breached. The SPI_STAT.RUWM bit indicates that the receive urgent watermark (SPI_RXCTL.RUWM) has been reached. This condition is cleared when the receive FIFO empties enough to reach the receive regular watermark (SPI_RXCTL.RRWM).	
		0	RX Regular Watermark reached
		1	RX Urgent Watermark breached

Table 24-31: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/NW)	SPIF	SPI Finished. The `SPI_STAT.SPIF` bit indicates that a single word transfer is complete.	
		0	No status
		1	Completed single-word transfer

Transmit FIFO Data Register

The SPI_TFIFO register has an interface to the transmit shift register in the SPI and has an interface to the processor's data buses. The top level of the buffer is visible to programs as the 32-bit SPI_TFIFO register, but the size (number of word locations) of the transmit FIFO is actually flexible with transfer word size. The size of the transmit FIFO is 8 if word size is 8-bit, or the size is 4 if word size is 16-bit, or the size is 2 if word size is 32-bit.

Both masters and slaves may stop or stall transmit transfers based on FIFO status. When the transmit FIFO is empty, the SPI master stops initiating new transfers on the SPI if SPI_TXCTL.TTI is enabled. A slave may stall the SPI interface when the content of the FIFO crosses the selected watermark. If data transmit requests continue after SPI_TFIFO is empty, the data sent from the transmit FIFO is invalid, and the SPI indicates this condition with transmit underrun (SPI_STAT.TUR). This condition is possible when SPI_TXCTL.TTI =0 and SPI_TXCTL.TEN =1 for a master, or for a slave that does not exercise flow control.

Note that the transmit FIFO is reset (cleared) when the SPI is disabled after being enabled.

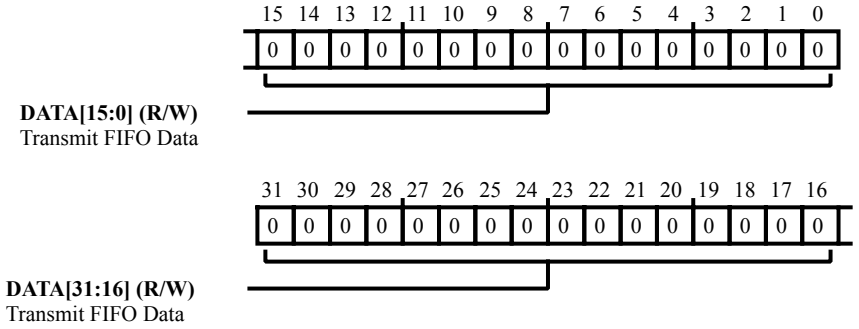

Figure 24-36: SPI_TFIFO Register Diagram

Table 24-32: SPI_TFIFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Transmit FIFO Data. The SPI_TFIFO.DATA bit field contains the FIFO transmit data.

Transmitted Word Count Register

The SPI_TWC register holds a count of the number of words remaining to be transmitted by the SPI. To start the decrement of the word count in SPI_TWC, enable the transmit word counter (SPI_TXCTL.TWCEN =1). The SPI uses the word count to control the duration of transfers and to signal the completion of a burst of transfers with the transmit finish interrupt. In DMA mode, the SPI uses the SPI_TWC to ensure that the number of frames transmitted during a DMA transfer is equal to the number of words programmed in the DMA channel controller. The values programmed into the SPI_TWC registers should match the word count in the DMA configuration. The SPI_TWC maintains the number of frames to be transmitted in a transfer. The SPI_TWC should only be changed when the counter is disabled.

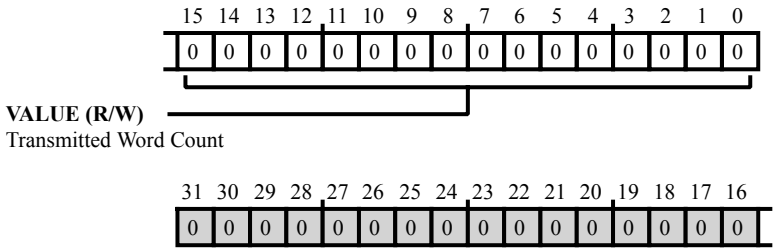

Figure 24-37: SPI_TWC Register Diagram

Table 24-33: SPI_TWC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Transmitted Word Count. The SPI_TWC.VALUE bits hold the transmit transfer word count.

Transmitted Word Count Reload Register

The SPI_TWCR register holds the transmit word count value that the SPI loads into the SPI_TWC register when the transfer count decrements to zero. To prevent the SPI from reloading the counter, use zero for the reload count value. The SPI_TWCR should only be changed when the counter is disabled.

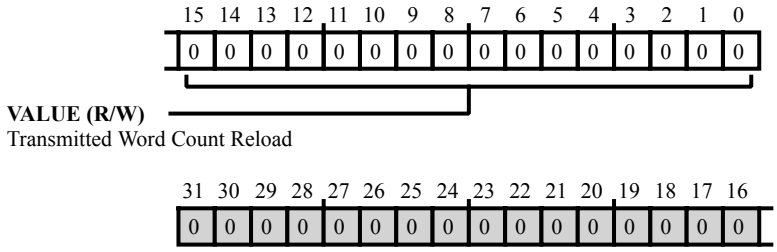

Figure 24-38: SPI_TWCR Register Diagram

Table 24-34: SPI_TWCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Transmitted Word Count Reload. The SPI_TWCR.VALUE bits hold the transmit transfer word count reload value.

Transmit Control Register

The SPI_TXCTL register enables the SPI transmit channel, initiates transmit transfers, and configures SPI_TFIFO buffer watermark settings.

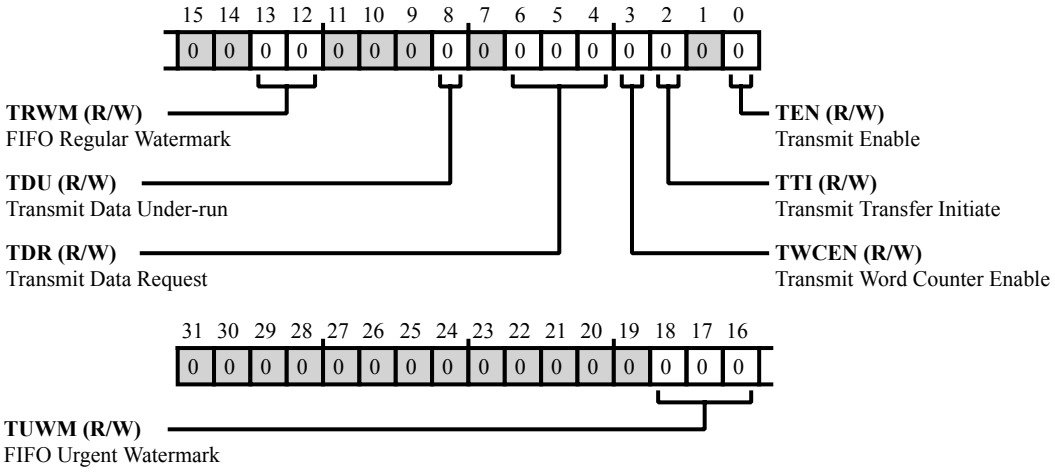

Figure 24-39: SPI_TXCTL Register Diagram

Table 24-35: SPI_TXCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	TUWM	FIFO Urgent Watermark. The SPI_TXCTL.TUWM bits select the transmit FIFO (SPI_TFIFO) watermark level for urgent data bus requests. The SPI also uses this watermark level for generation of the SPI_ILAT.TUWM interrupt. When an urgent SPI_TFIFO watermark is enabled with SPI_TXCTL.TUWM, the SPI_TXCTL.TRWM selection is used as the de-assertion condition for any SPI_ILAT.TUWM interrupts that are latched.	
		0	Disabled
		1	25% empty TFIFO
		2	50% empty TFIFO
		3	75% empty TFIFO
		4	Empty TFIFO

Table 24-35: SPI_TXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13:12 (R/W)	TRWM	FIFO Regular Watermark. The SPI_TXCTL.TRWM bits select the transmit FIFO (SPI_TFIFO) watermark level for regular data bus requests. When an urgent SPI_TFIFO watermark is enabled with SPI_TXCTL.TUWM, the SPI_TXCTL.TRWM selection is used as the de-assertion condition for any SPI_ILAT.TUWM interrupts that are latched.	
		0	Full TFIFO
		1	TFIFO less than 25% empty
		2	TFIFO less than 50% empty
		3	TFIFO less than 75% empty
8 (R/W)	TDU	Transmit Data Under-run. The SPI_TXCTL.TDU bit selects handling for transmit data requests when the transmit buffer (SPI_TFIFO) is empty. If enabled and SPI_TFIFO is empty, the SPI transmits zero as data. If disabled and SPI_TFIFO is empty, the SPI transmits the last word in the buffer as data.	
		0	Send last word when SPI_TFIFO is empty
		1	Send zeros when SPI_TFIFO is empty
6:4 (R/W)	TDR	Transmit Data Request. The SPI_TXCTL.TDR bits select transmit FIFO (SPI_TFIFO) watermark conditions that direct the SPI to generate a transmit status interrupt.	
		0	Disabled
		1	Not full TFIFO
		2	25% empty TFIFO
		3	50% empty TFIFO
		4	75% empty TFIFO
		5	Empty TFIFO
3 (R/W)	TWCEN	Transmit Word Counter Enable. The SPI_TXCTL.TWCEN bit enables the decrement of the transmit word count (SPI_TWC) register when the count is not zero and SPI_TXCTL.TTI is enabled. Enabling SPI_TXCTL.TWCEN prevents transmit under-run errors from occurring. The SPI_TXCTL.TWCEN bit is valid only when the SPI is a master.	
		0	Disable
		1	Enable

Table 24-35: SPI_TXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	TTI	Transmit Transfer Initiate. The SPI_TXCTL.TTI bit enables initiation of transmit transfers if the transmit FIFO (SPI_TFIFO) is not empty. The bit also enables this initiation if SPI_TWC is not zero when SPI_TXCTL.TWCEN is enabled. Enabling SPI_TXCTL.TTI prevents transmit underrun errors from occurring. The SPI_TXCTL.TTI bit is valid only when the SPI is a master.	
		0	Disable
		1	Enable
0 (R/W)	TEN	Transmit Enable. The SPI_TXCTL.TEN bit enables SPI transmit channel operation.	
		0	Disable
		1	Enable

25 Serial Port (SPORT)

The serial ports (SPORTs) support various protocols for serial data communication. In addition, the SPORTs provide a glueless hardware interface to many industry-standard data converters and codecs. The SPORTs have high data rates and dual half-duplex datapaths. The SPORT interface is a perfect choice for direct serial interconnection between two or more processors in a multiprocessor system. Many processors provide compatible serial interfaces, including DSPs from Analog Devices and other manufacturers.

The SPORT top module consists of two half SPORTs with identical functionality. Each SPORT half can be independently configured as either a transmitter or receiver and can be coupled with the other HSPORT within the same SPORT. Further, each SPORT half provides two synchronous half-duplex data lines to double the total supported data streams.

Each SPORT half has the same capabilities and is programmed in the same way.

Features

An individual SPORT module consists of two independently configurable SPORT halves with identical functionality. These SPORT halves offer the following features.

- Two bidirectional data lines—Primary (0) and secondary (1) per SPORT half, configurable as either transmitters or receivers. Therefore, each SPORT half can be configured for two transmitter or two receiver channels, permitting two unidirectional streams into or out of the same SPORT half. This bidirectional functionality provides greater flexibility for serial communications. Further, two SPORT halves can be combined to enable full-duplex, dual-stream communications.
- Six operation modes
    1. Standard DSP serial mode
    2. I ^{2}S mode
    3. Left-Justified mode
    4. Right-Justified Mode
    5. Multichannel Mode
    6. Packed mode

- Improved granularity for internal clock generation, allowing both even and odd SYSCLK to SPORT_CLK ratios. If both data lines of a SPORT half are active, it can have a maximum throughput of 2 x SPORT_CLK. The SPORTs can accept an input clock from an external source.
- Configurable rising or falling edge of the SPORT_CLK for driving or sampling data and frame sync.
- Gated clock mode support for both internal clock and external clock mode in DSP serial mode and stereo modes (left-justified and I ^{2}S mode).
- Operates with or without a frame synchronization signal for each data word.
- Operates with internally generated or externally generated frame signals
- Operates with active high or active low frame signals
- Operates with either of two configurable pulse widths and frame signal timing.
- Status flagging and optional interrupt generation for prematurely received external frame syncs.
- Signal for external frame sync configurable as level-sensitive or edge-sensitive signal.
- Serial data words 4–32 bits in length, either in most significant bit (MSB) first or in least significant bit (LSB) first format. Optional sign-extension on received data.
- Optional 16-bit to 32-bit word packing, when SPORT is configured as receiver. 32-bit to 16-bit word unpacking, when SPORT is configured as transmitter.
- When configured as transmitter, both primary and secondary datapaths can have optional compress engines enabled. Similarly, in receiver mode, both paths can have optional expand engines enabled. A-law and µ-law compression or decompression hardware companding according to G.711 specification on transmitted and received words in all operating modes.
- Status flagging and optional interrupt generation for transmit under-run or receive over-flow.
- Supports multichannel mode for TDM interfaces. Each SPORT half can transmit or receive data selectively from a time-division-multiplexed serial bit stream. The mode transfers data on 128 contiguous channels from a stream of up to 1024 total channels. This mode can be useful for H.100/H.110 and other telephony interfaces as a network communication scheme for multiple processors.
- Performs interrupt-driven, single word transfers to and from on-chip or off-chip memory under processor control.
- Dedicated DMA channel for each SPORT half. This DMA is common for both data lines and is configurable for multiple work units. Units include auto-buffer based (for a repeated, identical range of transfers) or descriptor-based (individual or repeated ranges of transfers with differing DMA parameters).
- The DMA of the SPORT can be programmed to accept the incoming trigger when configured as trigger slave. The DMA can generate outgoing trigger as well.
- When using DMA in transmit mode, the SPORT has a transfer finish interrupt (TFI). The interrupt ensures that the last word of the transfer shifts out of the transmit shift register.

- Provide SPMUX, a local multiplexing block integrated between the SPORT and the PinMux logic. The SPORT module can route and share the clocks and frame sync between the SPORT halves. The internal routing helps to reduce the total number of processor pins required for the interface. This functionality is especially efficient when a SPORT performs full-duplex data transfers.

Signal Descriptions

Each SPORT half module has five dedicated pins, as described in the ***SPORT Pin Descriptions*** table. The actual pin name varies with different SPORT halves. The individual SPORT half does not share any of its pins across the pair. However, if necessary, interconnect the clock and frame sync signals between the SPORT half pair, as explained in SPORT pin MUX section.

All the SPORT signals are available on the GPIO pins and are multiplexed with other peripheral signals. By default, these pins are in GPIO mode. To enable the pins for SPORT functionality, set the appropriate bits in the PORT_FER and PORT_MUX registers. Configure the PORT_MUX register before the PORT_FER register.

Table 25-1: SPORT Pin Descriptions

Internal Node	Direction	Description
SPORTx_CLK	I/O	Transmit or receive serial clock. Data and frame sync are driven or sampled based on this clock. This signal can be either internally or externally generated.
SPORTx_FS	I/O	Transmit or receive frame sync. The frame sync pulse initiates shifting of serial data. This signal is either generated internally or externally.
SPORTx_D0	I/O	Transmit or receive primary data channel. Bidirectional data pin. This signal can be configured as an output to transmit serial data, or as an input to receive serial data.
SPORTx_D1	I/O	Transmit or receive secondary data channel. Bidirectional data pin. This signal can be configured as an output to transmit serial data, or as an input to receive serial data.
SPORTx_TDV	O	Multichannel transmit data valid. This signal is only active if SPORT is configured in multichannel transmit mode. The signal is asserted during enabled slots based on the channel selection registers (SPORT_CS0_A through SPORT_CS3_B).

The signals are transmit signals when the serial port is configured in transmit mode (SPORT_CTL_A.SPTRAN = 1). The signals are receive signals when the serial port is configured in receive mode (SPORT_CTL_A.SPTRAN = 0). The following sections describe these SPORT signals.

Serial Clock

The serial port clock (SPORT_ACLK) signal is a receive serial clock when the transfer direction is configured as receiver. The signal is a transmit serial clock when the transfer direction is configured as transmitter.

The serial clock (SPORT_ACLK) is one of the control signals of the serial port depending on which the data bits are shifted-in or shifted-out serially. The frame sync signal is driven (in internal frame sync mode) or sampled (in external frame sync mode) based on the serial clock signal. The serial clock can be internally generated from the system clock (SYSCLK) of the processor or externally provided, based on the SPORT_CTL_A.ICLK bit setting. If a SPORT is configured in internal clock mode (SPORT_CTL_A.ICLK= 1), then the SPORT_DIV_A.CLKDIV field specifies the

divider to generate a serial port clock signal from its fundamental clock, SYSCLK. This divisor is a 16-bit value, allowing a wide range of serial clock rates. Use the following equation to calculate the serial clock frequency:

```
SPORT_ACLK = [SYSCLK ÷ (SPORT_DIV_A.CLKDIV + 1)]
```

Use the following equation to determine the value of SPORT_DIV_A.CLKDIV, given the SYSCLK frequency, and the desired frequency of the serial port clock:

```
SPORT_DIV_A.CLKDIV= [(SYSCLK ÷ SPORT_ACLK) - 1]
```

This equation results in improved granularity for internal clock generation, allowing both odd and even SYSCLK: SPORT_ACLKratios.

It also supports 1:1 SPORT_ACLK to SYSCLK ratio, when CLKDIV field is programmed to zero. The resulting frequency of the serial port clock is equal to the system clock. Do not to exceed the maximum SPORT_ACLK frequency specified in the data sheet. If SYSCLK is greater than the data sheet limit, limit the SPORT_ACLK:SYSCLK ratio to 1:2. For other SYSCLK frequencies, program this ratio up to 1:1.

In certain operating modes, the serial port can be configured to generate gated clock which is active only during valid data. In some applications, a SPORT uses it to generate a general-purpose clock in the system. In this case, enable the SPORT with the appropriate SPORT_DIV_A.CLKDIV divisor field in internal clock mode.

If a SPORT is configured in external clock mode (SPORT_CTL_A.ICLK= 0), then the serial clock is an input signal making the SPORT operate in slave mode. The SPORT_DIV_A.CLKDIV is ignored. The optional loopback capability provided by the SPMUX block allows the slave SPORT to use the serial clock from the neighboring serial port.

The externally-supplied serial clock does not need to be synchronous with processor system clock. Further, the external clock can be a gated clock but it must comply the requirements described in Gated Clock Mode section. Refer to the appropriate product data sheet for exact AC timing specifications.

Frame Sync

The serial port frame sync (SPORT_AFS) signal is a receive frame sync when the transfer direction is configured as receiver. It is a transmit frame sync when configured as transmitter.

The frame sync is also a control signal used to determine the start of new word or frame. Upon detecting this signal, the serial port starts shifting in or out the new data bits serially based on the direction selected. The frame sync signal can be internally generated from its serial clock (SPORT_ACLK) or externally provided, based on the SPORT_CTL_A.IFS bit setting.

If SPORT is configured for internal frame sync mode (SPORT_CTL_A.IFS= 1), then the SPORT_DIV_A.FSDIV field specifies the divider to generate SPORT_AFS signal from the serial clock. This divisor is a 16-bit value, allowing a wide range of frame sync rates to initiate periodic transfers. The serial clock can be internally generated or externally supplied. It is counted equal to the divisor specified before a frame sync pulse is generated. The formula for the number of cycles between frame sync pulses is:

Number of serial clocks between frame syncs = (SPORT_DIV_A.FSDIV + 1)

Use the following equation to determine the value of SPORT_DIV_A.FSDIV, given the serial clock frequency, and desired frame sync frequency:

SPORT_DIV_A.FSDIV = [(SPORT_ACLK ÷ SPORT_AFS) – 1]

The frame sync is continuously active when SPORT_DIV_A.FSDIV= 0. The value of SPORT_DIV_A.FSDIV cannot be less than the serial word length minus one (the value of the SPORT_CTL_A.SLEN bit field). This error can cause an external device to abort the current operation or cause other unpredictable results.

NOTE: After enabling the SPORT, the first internal frame sync appears after a delay of (SPORT_DIV_A.FSDIV+ 3) serial clocks.

If a SPORT is configured in external frame sync mode (SPORT_CTL_A.IFS = 0), then SPORT_AFS is an input signal and the SPORT_DIV_A.FSDIV field of the SPORT_DIV_A register is ignored. By default, this external signal is level-sensitive, but can be configured as an edge-sensitive signal by setting the SPORT_CTL_A.FSED bit. The frame sync is expected to be synchronous with the serial clock. If not, it must meet the timing requirements that appear in the product-specific data sheet.

The serial port can be used as a counter for dividing an external clock to generate periodic pulses or periodic interrupts. Enable the SPORT with the appropriate SPORT_DIV_A.FSDIV divisor field in external clock, internal data-independent frame sync mode.

In some of the operating modes, the serial port can be programmed to treat the frame sync signal as an optional signal by clearing the SPORT_CTL_A.FSR bit. (SPORT uses the configuration to start the continuous transfers and it is then ignored). Characteristics of the frame sync depend on the settings in the SPORT control registers and the operating mode of the SPORT. For more information, refer to the SPORT control register bits and respective operating mode details.

Data Signals

Each SPORT half has two bidirectional data lines known as the primary transmit or receive data channel (SPORT_AD0) and the secondary transmit or receive data channel (SPORT_AD1). Both the data lines can be configured as either transmitters or receivers using the SPORT_CTL_A.SPTRAN bit. This configuration permits dual unidirectional data streams to increase the data throughput of the serial port.

Both data lines can be individually enabled or disabled using the SPORT_CTL_A.SPENPRI and the SPORT_CTL_A.SPENSEC bits. However, if using both, enable or disable them concurrently. They do not behave as separate SPORTs; rather, they operate in a synchronous manner (sharing a clock and frame sync) but on separate datapaths. All of the SPORT control settings are common for both channels but the single DMA channel per serial half serves both primary and secondary data channels. Also, both primary and secondary channels have separate data buffers, shift registers and optional companding logic in their path.

When a serial port is configured in multichannel transmit mode, the data pins three-states during inactive channel slots. This configuration allows multiple transmitters to operate on the same bus with different active channels.

See the Architectural Concepts for more details about data transfer operation.

Transmit Data Valid Signal

The transmit data valid (SPORT_ATDV) signal is available only in multichannel modes (including packed mode) of a SPORT configured as a transmitter. This signal is active during transmission of enabled multichannel slots and remains in an inactive state for the disabled channels. In other words, the SPORT_ATDV signal is active whenever a serial port drives the data pins and stays inactive when the data pins three-states. Therefore the SPORT_ATDV signal can serve as an output-enable signal for the data transmit pin.

Functional Description

The following section provides general information about functionality of the serial ports of processors.

- Architectural Concepts
- Data Types and Companding
- Transmit Path
- Receive Path

ADSP-CM40x SPORT Register List

The Serial Port (SPORT) controller, with its range of clock and frame synchronization options, supports a variety of serial communication protocols and provides a glueless hardware interface to many industry-standard data converters and CODECs. Each SPORT has two independent halves (A and B), and each half contains two channels (primary and secondary). A set of registers governs SPORT operations. For more information on SPORT functionality, see the SPORT register descriptions.

Table 25-2: ADSP-CM40x SPORT Register List

Name	Description
SPORT_CS0_A	Half SPORT 'A' Multi-channel 0-31 Select Register
SPORT_CS0_B	Half SPORT 'B' Multi-channel 0-31 Select Register
SPORT_CS1_A	Half SPORT 'A' Multi-channel 32-63 Select Register
SPORT_CS1_B	Half SPORT 'B' Multi-channel 32-63 Select Register
SPORT_CS2_A	Half SPORT 'A' Multi-channel 64-95 Select Register
SPORT_CS2_B	Half SPORT 'B' Multichannel 64-95 Select Register
SPORT_CS3_A	Half SPORT 'A' Multi-channel 96-127 Select Register
SPORT_CS3_B	Half SPORT 'B' Multichannel 96-127 Select Register
SPORT_CTL2_A	Half SPORT 'A' Control 2 Register
SPORT_CTL2_B	Half SPORT 'B' Control 2 Register
SPORT_CTL_A	Half SPORT 'A' Control Register
SPORT_CTL_B	Half SPORT 'B' Control Register

Table 25-2: ADSP-CM40x SPORT Register List (Continued)

Name	Description
SPORT_DIV_A	Half SPORT 'A' Divisor Register
SPORT_DIV_B	Half SPORT 'B' Divisor Register
SPORT_ERR_A	Half SPORT 'A' Error Register
SPORT_ERR_B	Half SPORT 'B' Error Register
SPORT_MCTL_A	Half SPORT 'A' Multi-channel Control Register
SPORT_MCTL_B	Half SPORT 'B' Multi-channel Control Register
SPORT_MSTAT_A	Half SPORT 'A' Multi-channel Status Register
SPORT_MSTAT_B	Half SPORT 'B' Multi-channel Status Register
SPORT_RXPRI_A	Half SPORT 'A' Rx Buffer (Primary) Register
SPORT_RXPRI_B	Half SPORT 'B' Rx Buffer (Primary) Register
SPORT_RXSEC_A	Half SPORT 'A' Rx Buffer (Secondary) Register
SPORT_RXSEC_B	Half SPORT 'B' Rx Buffer (Secondary) Register
SPORT_TXPRI_A	Half SPORT 'A' Tx Buffer (Primary) Register
SPORT_TXPRI_B	Half SPORT 'B' Tx Buffer (Primary) Register
SPORT_TXSEC_A	Half SPORT 'A' Tx Buffer (Secondary) Register
SPORT_TXSEC_B	Half SPORT 'B' Tx Buffer (Secondary) Register

ADSP-CM40x SPORT Interrupt List

Table 25-3: ADSP-CM40x SPORT Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
23	SPORT0_A_DMA_ERR	SPORT0 DMA Channel 0 Error	Level	
24	SPORT0_B_DMA_ERR	SPORT0 DMA Channel 1 Error	Level	
29	SPORT1_A_DMA_ERR	SPORT1 DMA Channel 6 Error	Level	
30	SPORT1_B_DMA_ERR	SPORT1 DMA Channel 7 Error	Level	
63	SPORT0_A_STAT	SPORT0 Channel A Status	Level	
64	SPORT0_A_DMA	SPORT0 Channel A DMA Transfer Complete	Level	0
65	SPORT0_B_STAT	SPORT0 Channel B Status	Level	
66	SPORT0_B_DMA	SPORT0 Channel B DMA Transfer Complete	Level	1
74	SPORT1_A_STAT	SPORT1 Channel A Status	Level	

Table 25-3: ADSP-CM40x SPORT Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
75	SPORT1_A_DMA	SPORT1 Channel A DMA Transfer Complete	Level	6
76	SPORT1_B_STAT	SPORT1 Channel B Status	Level	
77	SPORT1_B_DMA	SPORT1 Channel B DMA Transfer Complete	Level	7

Block Diagram

The serial port is configured in transmit mode, when the `SPORT_CTL_A.SPTRAN` control bit is set. If this bit is cleared, the serial port configures in receive mode. If `SPORT_CTL_A.SPENPRI` control bit is set, then serial port activates primary transmit or receive path. If `SPORT_CTL_A.SPENSEC` control bit is set, then it activates secondary transmit or receive path. Enable both data channels to allow synchronous dual-stream communication. Each path optionally supports hardware companding or expanding as well. Once a path is activated, data shifts in response to a frame sync at the rate of serial clock. Do not use or access inactive data buffers. An application program must use the appropriate data buffers.

These serial ports are not UARTs and cannot communicate with an RS-232 device or any other asynchronous communications protocol.

The ***Half Serial Port Block Diagram*** shows a detailed block diagram of a SPORT half side.

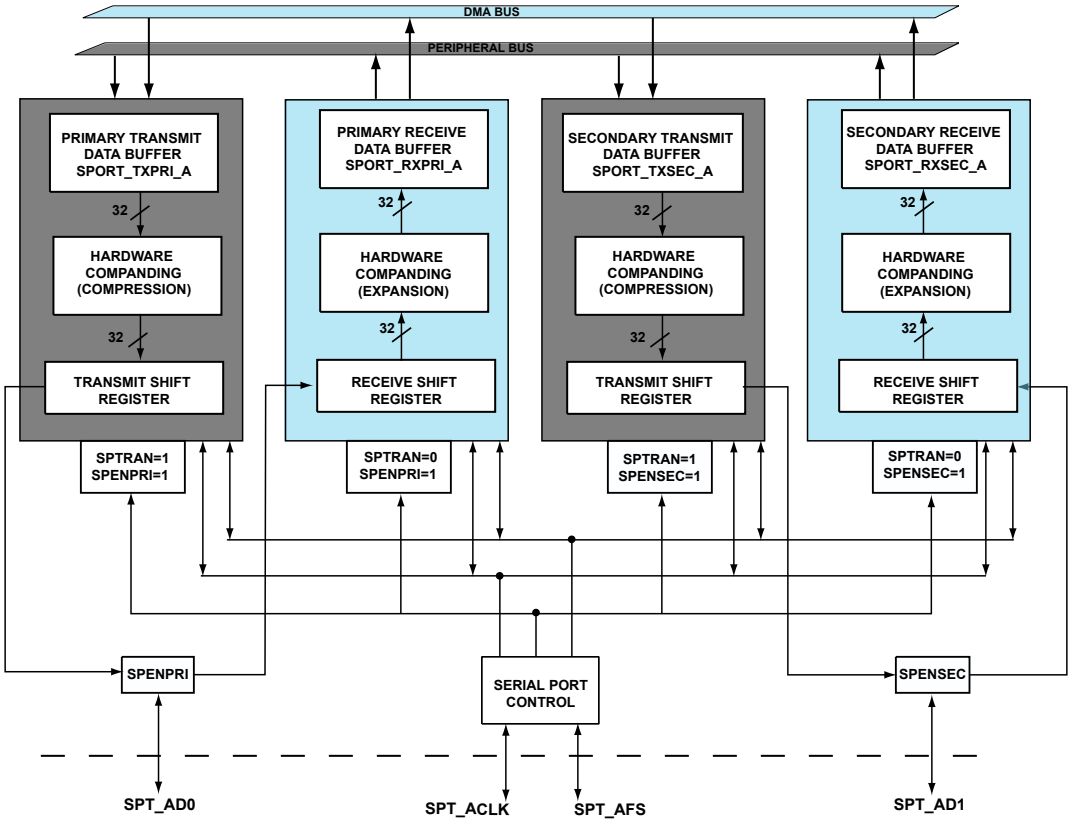

Figure 25-1: Half Serial Port Block Diagram

Architectural Concepts

Each SPORT module consists of two separate blocks, known as half-SPORT (HSPORT) A and B, with identical functionality. These blocks can be independently configurable as either transmitter or receiver; and optionally coupled together internally in a limited way. Each HSPORT also supports two synchronous bidirectional datapaths, referred to as primary (`D0`) and secondary (`D1`) data lines, as shown in the ***Top-Level SPORT Diagram*** figure.

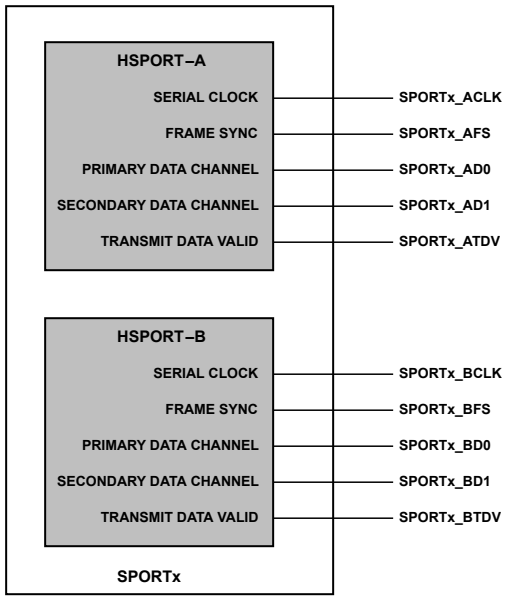

Figure 25-2: Top-Level SPORT Diagram

Each HSPORT can be configured as either transmitter or receiver, according to which the pair of data signals transmit or receive data bits synchronously. The SPORT uses the `SPORT_CTL_A.SPTRAN` bit to control the direction for both datapaths of the HSPORT. Each HSPORT has its own set of control registers and data buffers grouped per SPORT module. The dual data signals of each HSPORT cannot transmit and receive the data simultaneously for full-duplex operation. Two HSPORTs are combined to achieve full-duplex operation.

Serial communications are synchronized to the serial clock signal. A clock pulse must accompany every data bit. Each serial port can internally generate its own serial clock signal from the system clock of the processor using the divisor field of the `SPORT_DIV_A.CLKDIV` bit field. If programmed, serial ports can also operate in external clock mode. Both primary and secondary data channels shift data based on `SPORT_CLK` rate and the `SPORT_CTL_A.CKRE` bit.

In addition to the serial clock signal, a frame synchronization signal can signal data. The framing signal can occur at the beginning of an individual word or at the beginning of a block of words. The configuration of frame sync signal depends upon the type of serial device connected to the processor. Each serial port can generate its own frame sync signal (`SPORT_FS`) depending on the bit settings of SPORT control register. An internally generated frame sync is derived from the SPORT clock using the divisor field of the `SPORT_DIV_A.FSDIV` bit field. Serial ports can also accept an external `SPORT_FS` signal. Both primary and secondary datapaths start shifting data after detecting a valid frame sync signal. This event happens according to control bit settings and the operating mode of serial port. Various communication protocols for serial data can be emulated according to the frame sync format. All frame sync options are available whether the signal is generated internally or externally.

Multiplexer Logic

There is a local muxing block, known as SPMUX, that is integrated between the SPORT and the PinMux logic of processor. It allows flexibility to route and share the clock and frame sync signals between the SPORT half pair of a SPORT. Two independent SPORT halves of a SPORT can be coupled together in this block. This feature can

reduce the total number of pins for the interface. It is considered efficient when the SPORT half pair is used for full-duplex operation.

The SPORT_CTL2_A register is used to configure this loopback feature. The “Register Descriptions” section of this chapter describes the control bits of this register.

The multiplexing depends on the SPORT_CTL_A.IFS and SPORT_CTL_A.ICLK bit settings. The SPORT_CTL2_A.CKMUXSEL and SPORT_CTL2_A.FSMUXSEL bit settings of the SPORT half pair also control multiplexing. The ***Frame Sync Combinations*** and ***Clock Combinations*** tables show the valid combinations for the bit settings:

- SPORT_CTL_A.IFS
- SPORT_CTL_A.ICLK
- SPORT_CTL2_A.CKMUXSEL
- SPORT_CTL2_A.FSMUXSEL

All other settings are illegal. However, hardware does not check or prevent the illegal settings. Ensure that programs use only legal combinations.

The Routing column of the ***Frame Sync Combinations*** table uses the following abbreviations.

- HSx_FI = Frame sync input. An external device or the neighboring SPORT half provides this input.
- HSx_FO = Frame sync output. When SPORT is configured in internal frame sync mode
- SPx_FS = Signal appearing on frame sync pin of the SPORT half

NOTE: In the following tables, the half-SPORT pair of a SPORT, A and B, are referred to as HS0 and HS1 and are applicable for all SPORTs.

Table 25-4: Frame Sync Combinations

FS Combination ID	HS0_IFS	HS1_IFS	FS0MUX	FS1MUX	Routing
1	0	0	0	0	Native FS Operation
2	0	1	0	0	Native FS Operation
3	1	0	0	0	Native FS Operation
4	1	1	0	0	Native FS Operation
5	0	0	1	0	HS0_FI ≤ SP1_FS; HS1_FI ≤ SP1_FS
6	0	1	1	0	HS0_FI ≤ HS1_FO ≥ SP1_FS
7	0	0	0	1	HS1_FI ≤ SP0_FS; HS0_FI ≤ SP0_FS
8	1	0	0	1	HS1_FI ≤ HS0_FO ≥ SP0_FS

The Routing column of the ***Clock Combinations*** table uses the following abbreviations.

- HSx_CI = Serial clock input. An external device or the neighboring SPORT half provides this input.
- HSx_CO = Serial clock output. When SPORT is configured in internal clock mode
- SPx_CLK = Signal appearing on serial clock pin of the SPORT half

Table 25-5: Clock Combinations

CLK Combination ID	HS0_ICLK	HS1_ICLK	CK0MUX	CK1MUX	Routing
9	0	0	0	0	Native CLK Operation
10	0	1	0	0	Native CLK Operation
11	1	0	0	0	Native CLK Operation
12	1	1	0	0	Native CLK Operation
13	0	0	1	0	HS0_CI ≤ SP1_CLK; HS1_CI ≤ SP1_CLK
14	0	1	1	0	HS1_CI ≤ HS1_CO ≥ SP1_CLK
15	0	0	0	1	HS1_CI ≤ SP0_CLK; HS0_CI ≤ SP0_CLK
16	1	0	0	1	HS1_CI ≤ HS0_FO ≥ SP0_CLK

Note the following extra points on these combination settings:

- FS IDs 1–4 are supported with all CLK IDs 9–16
- FS ID 5 is only supported with CLK ID 13 and conversely
- FS ID 6 is only supported with CLK ID 14 and conversely
- FS ID 7 is only supported with CLK ID 15 and conversely
- FS ID 8 is only supported with CLK ID 16 and conversely
- CLK IDs 9–12 are supported with all FS IDs 1–8

NOTE: A SPORT half can import the serial clock signal from a paired HSPORT, only when it is configured in external clock mode. Similarly, a SPORT half can import the frame sync signal, only when it is configured in external frame sync mode. The neighboring SPORT can be a master or slave. A master generates a serial clock. A slave accepts an external clock or external frame sync. Program the `SPORT_CTL2` register only at the acceptor SPORT half side. Programming is not required at the donor SPORT half side for sharing. Polarity bits such as `SPORT_CTL_A.CKRE` and `SPORT_CTL_A.LFS` must have identical settings when using muxing between two SPORT halves.

Data Types and Companding

The SPORT uses the data type select field SPORT_CTL_A.DTYPE bit to specify one of the four data formats supported by serial ports. These formats apply to any of the operating modes of serial port.

Table 25-6: Data Type Bit Field Settings

DTYPE field	SPORT Receiver	SPORT Transmitter
00	Right-justify, zero-fill unused most significant bits	Normal operation
01	Right-justify, sign-extend unused most significant bits	Reserved
10	Expand using µ-law	Compress using µ-law
11	Expand using A-law	Compress using A-law

These formats apply to data words loaded into the SPORT transmit or receive data buffers. The first two data formats (00 and 01 values of SPORT_CTL_A.DTYPE) are applicable only when SPORT is configured as receiver. When SPORT is configured as transmitter, only the significant bits are transmitted (per the field defined in control register). Therefore, the transmit data buffers are not actually zero-filled or sign-extended.

The other two data formats enable the companding logic on the transmit or receive path. Companding (compressing or expanding) is the process of logarithmically encoding and decoding data to minimize the number of bits sent. The SPORTs of the processor support the two most widely used companding algorithms, A-law, and µ-law. The algorithms are performed according to the CCITT G.711 specification.

If selected, companding applies to both the enabled data channels. When enabled as SPORT transmitter, writes to transmit buffer make the content compressed to 8 bits according to algorithm selected.(The content is zero filled to the width of the transmit word). Similarly, if configured in receive mode, the 8 bits in the receive data buffers expand in right-justified, zero fill format per the algorithm selected. If companding is enabled in multichannel mode, it applies to all the active channels.

The compression for transmit data requires a minimum word length of 8 for proper function. If SPORT_CTL_A.SLEN is less than 7, then expansion does not work correctly. Also, if the data value is greater than 13-bit A-law or 14-bit µ-law maximum, it automatically compresses to the maximum value.

NOTE: The processor companding logic supports in-place companding feature. So, companding can be used for debug without enabling SPORT.

Companding as a Function

Since the values in the transmit and receive buffers are companded in place, the SPORT can use the companding hardware without transmitting (or receiving) any data. For example, during testing or debugging. For companding to execute properly, program the SPORT registers prior to loading data values into the SPORT buffers.

To compress data in place without transmitting, use the following procedure.

1. Set the serial port as transmitter (SPORT_CTL_A.SPTRAN = 1) with both primary and secondary data channels disabled (SPORT_CTL_A.SPENPRI = SPORT_CTL_A.SPENSEC= 0).

2. Enable companding in the SPORT_CTL_A.DTYPE field.
3. Write a 32-bit data word to the transmit buffer.
4. Wait for two system clock cycles. The SPORT can use any instruction not accessing the transmit buffer to cause this delay. This functionality allows the serial port companding hardware to reload the transmit buffer with the companded value.
5. Read the 8-bit compressed value from the transmit buffer.

To expand data in place, use the same sequence of operations with the receive buffer instead of the transmit buffer. When expanding data in this way, set the appropriate serial word length (SPORT_CTL_A.SLEN) bit.

Transmit Path

The SPORT uses the SPORT_CTL_A.SPTRAN control bit (when set =1) to configure transmit mode. It then enables primary and secondary transmit paths, based on the SPORT_CTL_A.SPENPRI and SPORT_CTL_A.SPENSEC bit settings. Both datapaths, primary and secondary, are separate but identical and include transmit data buffer, optional companding logic and a transmit shift register.

The data buffer on the primary transmit path is the primary transmit data buffer, or SPORT_TXPRI_A. The data buffer on the secondary transmit path is the secondary transmit data buffer, or SPORT_TXSEC_A. The transmit data buffer and output shift register form a FIFO type of structure. When packing is disabled (SPORT_CTL_A.PACK = 0), the serial port can hold as many as 3 data. If packing is enabled (SPORT_CTL_A.PACK = 1), the serial port can hold 2 packed data at any time.

The transmit data for primary or secondary channels is written to the SPORT_TXPRI_A, and SPORT_TXSEC_A transmit data buffers, respectively. The transmit data buffers can be accessed in core mode through the peripheral bus, or, in DMA mode, through the DMA bus. Do not access the inactive data buffer. This data is optionally compressed in hardware according to selected algorithm and then automatically transferred to transmit shift register. The shift register, clocked by SPORT_ACLK signal, then serially shifts out this data on SPORT_AD0 or SPORT_AD1 pins, synchronously. If the SPORT uses a framing signal, the SPORT_AFS signal indicates the start of the serial word transmission.

When using DMA mode, a single DMA feeds the data buffers of the enabled channels (primary and or secondary). When using both channels, interleave the data of these channels properly.

When SPORT is configured in non-multichannel mode as transmitter, the enabled SPORT data pins SPORT_AD0 or SPORT_AD1 are always driven. When a SPORT channel is enabled, data from the transmit data buffer is loaded into transmit shift register. The shift register then immediately latches the first bit of data, not based on the frame sync.(The data is either LSB or MSB based on the SPORT_CTL_A.LSBF bit setting). Similarly, if the frame sync duration is greater than the serial word length, then the data pins drive the first bit of the next word for transmission. The data is driven during inactive serial clock cycles. (The clock cycles after data transmission in the current frame). The first bit of the next word is loaded into shift register. This functionality does not cause any problem at the receiver end. The receiver starts sampling the data pin only after detecting a valid frame sync. In multichannel mode, data pin always three-states during inactive channel slots.

The serial port provides status of transmit data buffers and also error detection logic for transmit errors such as under-run. See the Error Detection section for more details.

When a serial port is configured in transmit mode, the receive paths are deactivated and do not respond to serial clock or frame sync signals. (The receive data buffers and receive shift registers on the paths are also deactivated). So, reading from an empty receive data buffer can cause the core to hang indefinitely.

Receive Path

The SPORT uses the SPORT_CTL_A.SPTRAN bit, when cleared, to configure the SPORT in receive mode. It then enables primary and secondary receive paths, based on the SPORT_CTL_A.SPENPRI and SPORT_CTL_A.SPENSEC bit settings. Both datapaths, primary and secondary, are separate but identical and include a receive shift register, optional companding logic, and a receive data buffer.

The data buffer on the primary receive path is the primary receive data buffer, or SPORT_RXPRI_A. The data buffer on the secondary receive path is the secondary receive data buffer, or SPORT_RXSEC_A. The receive paths act like a 3-word deep (32-bit) FIFO because they have two data registers plus an input shift register.

Upon enabling the serial port data channels, the input shift register shifts in data bits on the SPORT_AD0, and SPORT_AD1 pins, synchronous to the receive clock signal. If the SPORT uses a framing signal, the SPORT_AFS signal indicates the beginning of the serial word received. When an entire word is shifted in on the primary and secondary channels, the data is optionally expanded in hardware according to selected algorithm. The data is then automatically transferred to SPORT_RXPRI_A, and SPORT_RXSEC_A.

The receive data buffers can be read in core mode through peripheral bus, or, in DMA mode, through the DMA bus. When the serial port uses DMA mode, a single DMA reads the data buffers of enabled channels (primary and or secondary). When using both channels, de-interleave the data of these channels properly. The serial port provides the status of receive data buffers and also error detection logic for receive errors such as overflow. See the Error Detection section for more details.

When a serial port is configured in receive mode, the transmit paths are deactivated and do not respond to serial clock or frame sync signals. (The transmit data buffers and transmit shift registers on the paths are also deactivated). Therefore, programs must not try to access them.

DMA and Interrupt Multiplexing

Please refer to the *Direct Memory Access (DMA)* chapter for information about DMA multiplexing. Several interrupts and DMA channels in the SPORT can be multiplexed.

Sampling Edge

The serial port uses two control signals to sample or drive the serial data.

1. Serial clock (SPORT_ACLK) applies the bit clock for each serial data.
2. Frame sync (SPORT_AFS) divides the incoming data stream into frames.

These control signals can be internally generated or externally provided, as determined by the SPORT_CTL_A.ICLK and SPORT_CTL_A.IFS bit settings.

Data and frame syncs can be sampled on the rising or falling edges of the serial port clock signals. The SPORT uses the SPORT_CTL_A.CKRE bit to control the sampling edge. By default, when SPORT_CTL_A.CKRE = 0, the processor selects the falling edge of SPORT_ACLK signal for sampling receive data and external frame sync. The receive data and frame sync are sampled on the rising edge of SPORT_ACLK when SPORT_CTL_A.CKRE = 1.

Transmit data and internal frame sync signals are driven (change their state) on the serial clock edge that is not selected. By default, when (SPORT_CTL_A.CKRE = 0), the SPORTs drive data and frame sync signals on the rising edge of the SPORT_ACLK signal, and drive on the falling edge when SPORT_CTL_A.CKRE = 1.

Therefore, select the same value for SPORT_CTL_A.CKRE for transmit and receive functions of any two serial ports that are connected together. With this configuration, the internally generated signals are driven on one edge and the received signals are sampled on the opposite edge.

The serial port which drives serial clock and frame sync is usually called the master while the receiver of clock and frame sync is referred to as the slave. The ***Frame Sync and Data Driven on Rising Edge*** figure shows the typical SPORT signals at two sides of serial communication for SPORT_CTL_A.CKRE = 0. The SPORT that is configured as transmitter also drives the serial clock and frame sync signals as a master device.

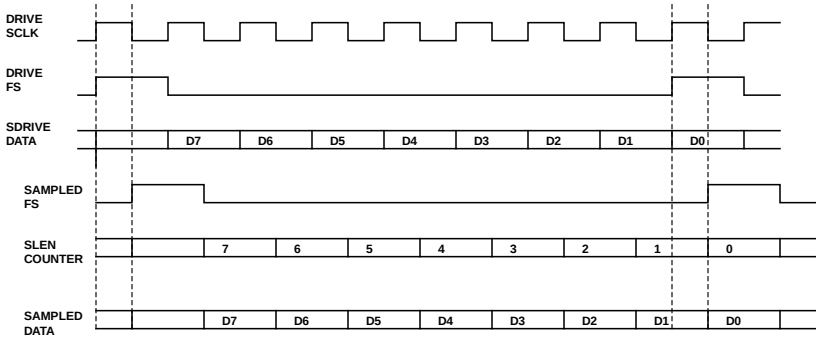

Figure 25-3: Frame Sync and Data Driven on Rising Edge

When a slave samples the frame sync signal, the SPORT_CTL_A.SLEN word counter is reloaded to the maximum setting. Each SPORT_ACLK decrements the SPORT_CTL_A.SLEN counter until the full frame is received.

If the transmitter drives the internal frame sync and data on the rising edge of serial clock, the receiver uses the falling edge to sample the external frame sync and data. The converse is also true.

Premature Frame Sync Error Detection

A SPORT framing signal is used to synchronize transmit or receive data. In external FS mode, any frame sync received when an active frame is in-progress is called premature and is invalid.

As an enhancement to the serial port of the processor, if a premature frame sync is received, the SPORT_ERR_A.FSERRSTAT bit is flagged to indicate this framing error. An optional error interrupt can be generated for this event by setting the SPORT_ERR_A.FSERRMSK bit.

This feature is applicable in all the operating modes of serial port.

NOTE: The SPORT_ERR_A.FSERRSTAT bit is not set in the presence of uncleared underflow or overflow errors.

In stereo or I²S mode, a premature frame sync can result in the SPORT receiving two consecutive left channels or two consecutive right channels and can cause channel swapping. In the serial port of the processor, swapping of channels due to a premature FS is avoided. If one data is corrupted due to a premature FS, data is always dropped in pairs to avoid channel swapping. The premature FS flagging in the error register is handled similarly.

Refer to the ***Frame Sync Error Detection*** figure. The frame sync error (which sets the error bit) is triggered when an early frame sync occurs during data transfer (transmission or reception). Or, the error is triggered for late frame sync when the period of the frame sync is smaller than the serial word length (SPORT_CTL_A.SLEN).

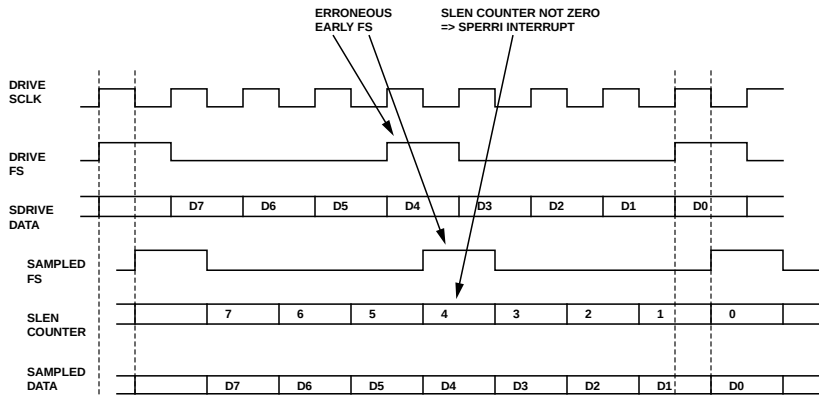

Figure 25-4: Frame Sync Error Detection

When a serial port is receiving or transmitting, its bit count is set to a word length (for example, 32 bits). After each clock edge, the bit count is decremented. After the word is received or transmitted, the bit count reaches zero, and on next frame sync it is set to 32. When active transmission or reception occurs, the bit count value is non-zero. When a frame sync with a bit count of non-zero is detected, a frame sync error occurs.

Support for Edge-Detected and Level-Sensitive Frame Syncs

The level-sensitive nature of frame sync operates well in a noise-free environment. However, if noise corrupts the signals coming into the SPORT, the internal logic can miss the start of frame sync. The miss is due to the corruption of either the clock or frame sync. The frame sync is sampled from the next clock edge onwards when it lasts for more than a bit clock period.

The ***Level-Sensitive Frame Sync versus Edge Sensitive Frame Sync*** figure describes a scenario when an external frame sync signal is corrupted due to noise. The slave SPORT module samples the signal incorrectly. Consider a frame sync, driven on the rising edge of serial clock at t_A. The slave SPORT normally samples the signal at the falling edge of serial clock at t_B. But, due to the noise, the SPORT misses the first edge of the FS and samples the FS only at t_C as shown in the figure. Then, the word length counter runs for a period equal to the SPORT_CTL_A.SLEN field of the control register. It expires at t_E, instead of correctly at t_D, receiving incorrect data. Furthermore, if a new frame sync edge has come at time t_D, in level-sensitive mode, the SPORT samples this framing signal again, at time t_E. So, the frame sync sampling continues to be misaligned with the external data.

The enhanced SPORT module provides an option to configure the frame sync signal as an edge-sensitive signal. The SPORT uses the edge-sensitive configuration for the signal to detect an edge in an external frame sync in considering it as a valid framing signal. In active-high frame syncs, the rising edge of frame sync is valid; while in active-low frame syncs, the falling edge is valid. Program `SPORT_CTL_A.FSED` (external frame sync edge select) bit to activate this optional feature.

NOTE: `SPORT_CTL_A.FSED` is valid only in external frame sync mode. In internal frame sync mode, this bit is a do-not-care.

In the example discussed, consider a frame sync configured as an edge-sensitive frame sync. In this case, the frame sync is not detected at t_E because the edge of framing signal has already occurred in the previous cycle (t_D). There is no edge to detect at t_E. So, the word length counter remains idle for this frame, ignoring the incorrect data. The counter resumes the operation correctly at t_F when a new edge detects.

This activity sets the `SPORT_ERR_A.FSERRSTAT` bit and optionally generates a premature FS error interrupt.

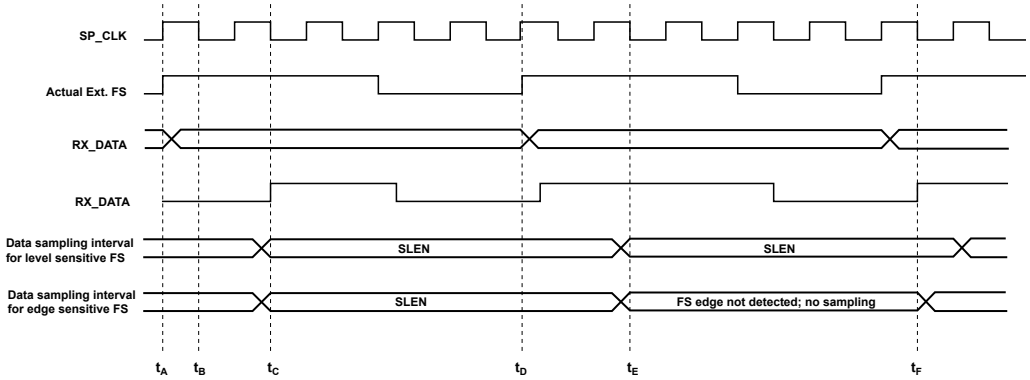

Figure 25-5: Level-Sensitive Frame Sync versus Edge Sensitive Frame Sync

Frame sync edge detection is used by default for stereo modes. MCM mode and DSP serial mode choose between edge detection and normal mode of FS detection.

NOTE: When the SPORT is enabled, an already active externally applied frame sync must not start operation. The SPORT waits for a valid state change from inactive to active for the external frame sync to consider it valid.

Serial Word Length

The SPORT uses the `SPORT_CTL_A.SLEN` field of the serial port control register to determine the word length of serial data to transmit and receive. Each SPORT half can independently handle word lengths up to 32 bits. The minimum allowable word length depends on operating mode selected. Words smaller than 32 bits are right-justified in the transmit or receive buffers to least significant bit (LSB) position. However, data can be shifted-in or out in MSB first or LSB first format according to `SPORT_CTL_A.LSBF` bit setting. Also, the received word can be sign-extended while storing it in processor memory.

Calculate the value of the `SPORT_CTL_A.SLEN` field as:

SLEN = Serial port word length - 1

The ***Data Length versus SPORT Operating Modes*** table shows the range of valid word lengths in the operating modes of SPORT.

Table 25-7: Data Length versus SPORT Operating Modes

Mode	Serial Port Word Length (SLEN+1)
Standard DSP Serial	4–32
I^2S	5–32
Left-Justified	5–32
Right-Justified	5–32
Multichannel	5–32
Packed I^2S	5–32

NOTE: If the companding feature is enabled on the datapath, it limits the word length settings. See Data Types and Companding for more details about word lengths required for companding. If more than 32 bits per frame sync are required to transmit or receive, use the multichannel mode (by enabling more than one channel).

Operating Modes

The SPORT has a number of operating modes:

- Standard serial mode
- I^2S mode
- Left-justified mode
- Right-Justified mode
- Multichannel mode
- Packed I^2S mode

The SPORT halves within a SPORT can be independently configured in any of these operating modes, unless they are not coupled together using SPMUX logic. Each SPORT half has its own set of control and data registers and is programmed similarly.

The main control register of serial port, `SPORT_CTL_A`, controls the operating modes of the SPORT. The ***Control Bits Comparison for Different Operating Modes*** table lists all the bits of the control register. The `SPORT_CTL_A` register is unique in that the bit function can change depending on the operating mode selected. Many bits in the control registers, controlling the function of the mode, are the same bit, but have a different name depending on the operating mode. The bits common across operating mode columns (for example `SPORT_CTL_A.SLEN`) signify a similar function across those operating modes. However, the bits divided as per operating modes (for example

SPORT_CTL_A.LFS) indicate different meanings depending on operating mode. Further, some bits are reserved depending on mode of operation. (For example, the SPORT_CTL_A.FSR bit is reserved in I^2S, left-justified sample pair, packed I^2S and in multichannel mode).

NOTE: When changing operating modes, clear the serial port control register before the new mode is written to the register.

Table 25-8: Control Bits Comparison for Different Operating Modes

Bit (NAME)	Standard Serial	I2S and Left-Justified	Right-Justified	Multichannel	Packed I2S
Control Bits					
0 (SPENPRI)	Yes				
2–1 (DTYPE)	Yes	Reserved		Yes	
3 (LSBF)	Yes	Reserved		Yes	
8–4 (SLEN)	Yes				
9 (PACK)	Yes				
10 (ICLK)	Yes				
11 (OPMODE)	Yes				
12 (CKRE)	Yes	Yes		Yes	
13 (FSR)	Yes	Reserved			
14 (IFS)	Yes	Yes		Yes	
15 (DIFS)	Yes	Yes	Yes	Reserved	
16 (LFS)	Yes	Yes		Yes	Yes
17 (LAFS)	Yes			Reserved	
18 (RJUST)	Reserved		Yes	Reserved	
19 (FSED)	Yes	Reserved		Yes	Reserved
20 (TFIEN)	Yes				
21 (GCLKEN)	Yes		Reserved		
24 (SPENSEC)	Yes				
25 (SPTRAN)	Yes				
Status Bits					
26 (DERRSEC)	Yes				
28–27 (DXSSEC)	Yes				
29 (DERRPRI)	Yes				
31–30 (DXSPRI)	Yes				

Mode Selection

The operating mode of the serial port is configured in the SPORT_CTL_A and SPORT_MCTL_A registers. The ***SPORT Operating Modes*** table provides values for each of the bits in the SPORT serial control registers used to configure a specific SPORT operation mode. The shaded columns indicate that the bits come from different control registers.

Table 25-9: SPORT Operating Modes

Operating Modes	OPMODE (11)	LAFS (17)	RJUST (18)	Multichannel Control Register (SPORT_MCTL_A)
Standard DSP Serial	0	Valid	0	0
I2S	1	0	0	0
Left-Justified	1	1	0	0
Right-Justified	1	1	1	0
Multichannel	0	x	0	1
Packed I^2S mode	1	x	0	1

The following sections provide detailed information on each operating mode available using the serial ports.

Standard Serial Mode

The SPORT can be configured in standard DSP serial mode by clearing the SPORT_CTL_A.OPMODE and SPORT_MCTL_A.MCE bits. The standard serial mode lets programs configure serial ports for use by various serial devices such as serial data converters and audio codecs. In order to connect to these devices, various clocking, framing, and data formatting options are available.

Timing Control Bits

Several bits in the serial port control register enable and configure the operation of standard serial mode.

- SLEN: serial word length select (4–32 bits)
- LSBF: little endian vs. big endian serial bit format
- ICLK: internal clock generation vs. external clock mode
- CKRE: sampling edge as rising edge vs. falling edge
- IFS: internal frame sync generation vs. external frame sync mode
- FSR: framed mode vs. unframed mode
- DIFS: data-dependent frame sync vs. data-independent frame sync
- LFS: active-high FS vs. active-low FS
- LAFS: early frame sync vs. late frame sync

- PACK: 16-bit to 32-bit packing enable vs. packing disable
- GCLKEN: normal free-running clock vs. gated clock mode

Clocking Options

In standard serial mode, the serial ports can either accept an external serial clock or generate it internally. The SPORT uses the SPORT_CTL_A.ICLK bit to determine the selection of these options. For internally generated serial clocks (SPORT_CTL_A.ICLK = 1), the SPORT_DIV_A.CLKDIV field configures the serial clock rate from the system clock.

In addition, the serial clock edge can be selected for the sampling or driving serial data and frame sync. The SPORT uses the SPORT_CTL_A.CKRE bit to perform this selection.

- If SPORT_CTL_A.CKRE = 0, incoming data and frame sync signals are sampled based on falling edge of serial clock. Data and frame sync output signals are driven at the rising edge of clock.
- If SPORT_CTL_A.CKRE = 1, incoming data and frame sync signals are sampled based on rising edge of serial clock. Data and frame sync output signals are driven at the falling edge of clock.
- The SPORT_CTL_A.GCLKEN bit enables clock gating option, in which serial clock is active only during the valid data bits.

Frame Sync Options

The following sections provide generic information about how the serial port uses the frame sync signal in an operating mode. SPORT halves within a SPORT are independently configurable. See Operating Modes for additional information about frame syncs and data sampling that apply to a specific operating mode.

Data-Dependent versus Data-Independent Frame Sync

The generation of a frame sync signal and data transmission are data-dependent. An internally generated transmit frame sync is output when:

- A new data word has been loaded into the channel transmit buffer of the SPORT, and
- Data-independent frame sync select (SPORT_CTL_A.DIFS) bit = 0, and
- SPORT is configured as a transmitter (SPORT_CTL_A.SPTRAN = 1)

This mode of operation allows data transmission only at specific times.

When SPORT is configured as receiver (SPORT_CTL_A.SPTRAN = 0) and if SPORT_CTL_A.DIFS = 0, then a receive frame sync signal is generated only when receive data buffer status is not full.

The data-independent frame sync mode allows the continuous generation of the framing signal, regardless of new data in the buffers. Setting SPORT_CTL_A.DIFS activates this mode. When SPORT_CTL_A.DIFS = 1, a transmit frame sync signal is generated regardless of the transmit data buffer status (if SPORT_CTL_A.SPTRAN = 1) or receive data buffer status (if SPORT_CTL_A.SPTRAN = 0).

The SPORT DMA controller typically keeps the transmit buffer full. The application is responsible for filling the transmit buffers with data.

Early versus Late Frame Syncs

The frame sync signals can be early or late. Frame sync signals can occur during the first bit of each data word (late) or during the serial clock cycle immediately preceding the first bit (early). The SPORT uses the `SPORT_CTL_A.LAFS` bit of the serial port control register to configure this option.

By default, when `SPORT_CTL_A.LAFS` is cleared (=0), the frame sync signal is configured as an early framing signal. This configuration is the normal mode of operation. In this mode, the first bit of the transmit data word is available in the serial clock cycle after the frame sync is asserted.(The first bit of the receive data word is latched). The frame sync is not checked again until the entire word has been transmitted (or received). In multichannel operation, this activity occurs when the frame delay is one.

If data transmission is continuous in early framing mode, then the frame sync signal occurs during the last bit of each word. (The last bit of each word precedes the first bit of the next word). Internally generated frame syncs are asserted for one clock cycle in early framing mode. This event is not an error condition, so the `SPORT_ERR_A.FSERRSTAT` bit is not flagged.

When `SPORT_CTL_A.LAFS` is set (=1), late frame syncs are configured; this mode is the alternate mode of operation. In this mode, the first bit of the transmit data word is available in the same serial clock cycle that the frame sync is asserted. (The first bit of the receive data word is latched). In multichannel operation, this state occurs when frame delay is zero. Serial clock edges latch the receive data bits, but the frame sync signal is checked only during the first bit of each word. Internally generated frame syncs remain asserted for the entire length of the data word in late framing mode. Externally generated frame syncs are only checked during the first bit. The frame syncs do not need assertion after that time period.

Therefore, for early framing, the frame sync precedes data by 1 cycle; for late framing, the frame sync is checked on the first bit only. The ***Normal Framing (Early Frame Sync) Versus Alternate Framing (Late Frame Sync)*** figure illustrates the two modes of frame signal timing.

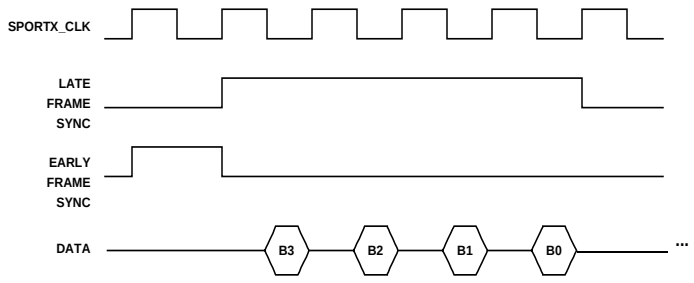

Figure 25-6: Normal Framing (Early Frame Sync) Versus Alternate Framing (Late Frame Sync)

Framed versus Unframed Frame Syncs

The use of a frame sync signal is optional in serial port communications. The SPORT uses the `SPORT_CTL_A.FSR` (frame sync required) bit to determine whether framing signal is required.

When SPORT_CTL_A.FSR bit is set (=1), a frame sync signal is required for every data word. To allow continuous transmission from the processor, load each new data word into the transmit buffer. Load the word before the previous word is shifted out and transmitted.

When SPORT_CTL_A.FSR is cleared (=0), the corresponding frame sync signal is not required. A single frame sync is required to initiate communications but it is ignored after the first bit transfers. Data words are then transferred continuously in what is referred to as an unframed mode. Unframed mode is appropriate for continuous reception. The ***Framed versus Unframed Data Stream*** figure shows the framed vs unframed mode of serial port operation.

NOTE: When DMA is enabled in a mode where frame syncs are not required, chaining can delay DMA requests. DMA requests are not always serviced frequently enough to guarantee continuous unframed data flow. Monitor status bits or check for a SPORT error interrupt to detect underflow or overflow of data.

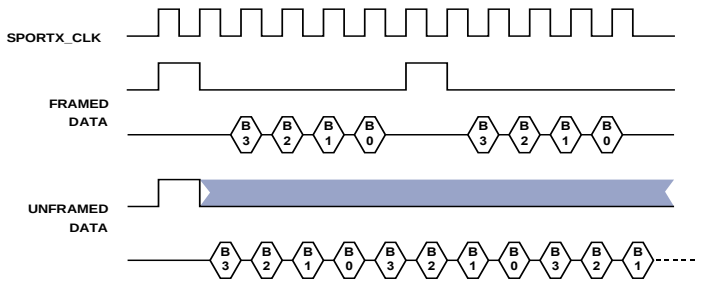

Figure 25-7: Framed versus Unframed Data Stream

Logic Level

The framing signals can be active high or active low. The SPORT uses the SPORT_CTL_A.LFS bit to select the logic level of the frame sync signals.

- When SPORT_CTL_A.LFS = 0, the corresponding frame sync signal is active high.
- When SPORT_CTL_A.LFS = 1, the corresponding frame sync signal is active low.

Active high is the default polarity of frame sync signal.

Stereo Modes

The processor serial port supports three widely used stereo modes, which are I^2S mode, left-justified mode, and right-justified mode. In these modes, the serial data stream consists of left and right channels. The following sections describe these modes.

Channel Order First

The active low frame sync (SPORT_CTL_A.LFS) bit holds different meaning for stereo modes of SPORT operation. The SPORT uses this bit to determine the polarity of frame sync level in DSP serial mode or multichannel mode. For left-justified, I^2S and packed I^2S modes, the ***Channel Order First Bit Settings*** table demonstrates which word is transmitted or received first. The priority depends on the SPORT_CTL_A.LFS bit setting.

Table 25-10: Channel Order First Bit Settings

OPMODE	LFS=0	LFS=1
Left-Justified or Right-Justified	Left channel first	Right channel first
I^2S or Packed I^2S	Right channel first	Left channel first

I^2S Mode

I^2S mode is a commonly used stereo mode. In this mode, for each frame sync cycle, two samples of data are transmitted or received. One sample is transmitted or received on the low segment of the frame sync, which is known as left channel. The other sample is transmitted or received on the high segment of the frame sync, which is known as right channel.

To configure the SPORT in I^2S mode, set `SPORT_CTL_A.OPMODE` = 1, `SPORT_CTL_A.LFS` = 0 and `SPORT_MCTL_A.MCE` = 0.

Protocol Configuration Options

Several bits in the `SPORT_CTL_A` control register enable and configure I^2S mode of operation:

- SLEN: serial word length select. For I^2S mode, the range of allowable word length is 5–32 bits.
- LSBF: little endian or big endian serial bit format. For I^2S mode, configure the serial data in big endian format (MSB bit is transmitted or received first). But, the format can be changed, depending on preferences, when emulating similar non-standard protocol
- ICLK: Internal bit clock generation or external bit clock mode
- IFS: Internal frame sync generation or external FS mode. In I^2S mode, master serial port is the one which generates the bit clock and L/R clock (frame sync) internally for the serial communication. The serial port which accepts these clocking signals is known as slave. So, in standard I^2S mode, the `SPORT_CTL_A.IFS` bit depends (equal to) on the `SPORT_CTL_A.ICLK` bit setting. However, as an enhancement, the `SPORT_CTL_A.IFS` bit setting can be changed depending on the application when emulating a non-standard protocol.
- LFS: left channel first or right channel first. This bit setting can be changed depending on user preference to sample right channel first (LFS = 0) or left channel first (LFS = 1).
- CKRE: sampling edge as rising edge or falling edge.
- PACK: 16-bit to 32-bit packing enable or packing disable.

Serial Clock and Frame Sync Rates

Set the serial bit clock rate for internal clocks using the `SPORT_DIV_A.CLKDIV` bit field. Set the L/R clock rate for internal frame sync using the `SPORT_DIV_A.FSDIV` bit field in the same register, depending on `SPORT_CTL_A.ICLK` and `SPORT_CTL_A.IFS` bit settings.

The ***Word Select Timing in I²S Mode*** figure shows the timing in I²S mode. In I²S mode, one SYSCLK cycle delays the data and the operation transfer starts on the left channel first.

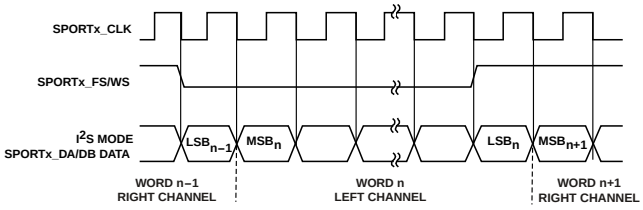

Figure 25-8: Word Select Timing in I²S Mode

The serial port of the processor does not generate a frame sync (L/R clock) edge after the transmission of the last word in the DMA.

Standard I²S receivers looks for the edge to latch and read data. Therefore, I²S slave receivers connected to the SPORT are not always able to latch the last word of the Tx DMA.

Left-Justified Mode

Left-justified mode is a stereo mode. In this mode, for each frame sync cycle, two samples of data are transmitted or received. One sample is transmitted or received on the high segment of the frame sync, which is known as left channel. The other sample is transmitted or received on the low segment of the frame sync, which is known as right channel.

This operating mode is simply a subset of the I²S mode. To configure the SPORT in left-justified mode, set the `SPORT_CTL_A.OPMODE` and `SPORT_CTL_A.LAFS` bits, and clear the `SPORT_MCTL_A.MCE` bit.

Protocol Configuration Options

Several bits in the `SPORT_CTL_A` control register enable and configure the left-justified mode of operation:

- SLEN: serial word length select. For left-justified mode, the range of allowable word length is 5–32 bits.
- LSBF: little endian or big endian serial bit format. For left-justified mode, put serial data in big endian format (MSB bit is transmitted or received first). Format can be changed, depending on the preferences, when emulating similar non-standard protocol
- ICLK: Internal bit clock generation or external bit clock mode
- IFS: Internal frame sync generation or external FS mode. In left-justified mode, master serial port is the one which generates bit clock and L/R clock (frame sync) internally for the serial communication. The serial port which accepts these clocking signals is known as slave. In standard left-justified mode, the `SPORT_CTL_A.IFS` bit depends (equal to) on `SPORT_CTL_A.ICLK` bit setting. However, as an enhancement, change the `SPORT_CTL_A.IFS` bit setting, depending on the application, when emulating a non-standard protocol.
- LFS: left channel first or right channel first. In standard left-justified mode, left channel is sampled first (corresponds to `SPORT_CTL_A.LFS` = 0). However, `SPORT_CTL_A.LFS` bit setting can be changed depending on user preference to sample right channel first (first data after rising edge of L/R clock).

- CKRE: sampling edge as rising edge or falling edge.
- PACK: 16-bit to 32-bit packing enable or packing disable.

Serial Clock and Frame Sync Rates

Set the serial bit clock rate for internal clocks using a `SPORT_DIV_A.CLKDIV` field. Set the L/R clock rate for internal frame sync using the `SPORT_DIV_A.FSDIV` field, depending on `SPORT_CTL_A.ICLK` and `SPORT_CTL_A.IFS` bit settings. The ***Word Select Timing in Left-Justified Mode*** figure shows the serial port timing in left-justified mode. (The figure shows MSB bit first format, but LSB bit first format is also possible). As shown, the first bit of a word is transmitted or received in the same clock cycle as the word select (SPORT_FS) signal changes.

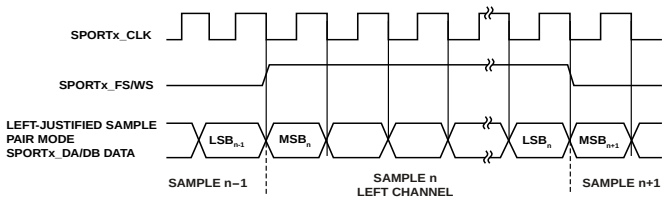

Figure 25-9: Word Select Timing in Left-Justified Mode

Right-Justified Mode

Right-justified mode is a standard commonly available in most of the SPORT compatible devices such as ADC and DACs. For right-justified mode, the design aligns the data to the end of the frame sync. The SPORT uses the `SPORT_CTL_A.RJUST` bit to align the serial data to the end of the frame sync.

The ***Word Select Timing in Right-Justified Mode*** figure shows the SPORT timing in right-justified mode. The transmitter aligns the transmit data. The last bit of the serial word is sent in the last clock cycle of the word select (frame sync) signal marking the channels. The timing seems similar to left-justified mode, but data is shifted such that it aligns to the end of the channel. In left-justified mode, the transmitter sends the MSB bit of the serial word in the same clock cycle as the word select signal changes.

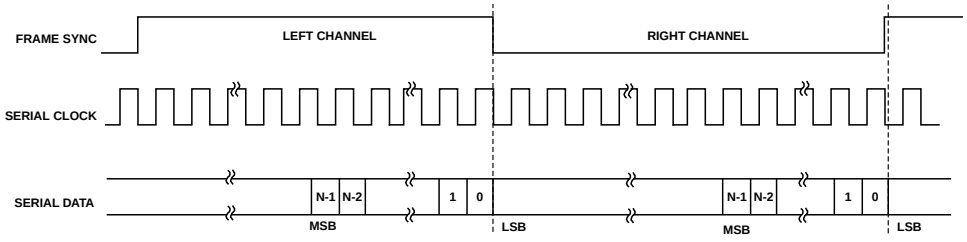

Figure 25-10: Word Select Timing in Right-Justified Mode

NOTE: For some SPORT-compatible ADCs or DACs such as the AD1871, right-justified mode is limited to commonly used ratios such as 64 FS and 128 FS. FS is the sampling frequency of ADCs and DACs, known as L/R clock or the frame sync of the SPORT.

As an illustration, consider the SPORT timing for right-justified mode, as shown in the ***Timing Comparison Between Different Stereo Modes*** figure. The frame sync width is limited to 32 SPORT clock periods or 32 bits per channel if:

- The L/R clock runs at the FS rate, and
- The SPORT clock runs at the 64 FS rate

The limitation applies to the frame sync width of either channel. If the data is confined to 24 bits, the SPORT introduces a 32–24=8-bit clock delay before it starts to transmit or capture data.

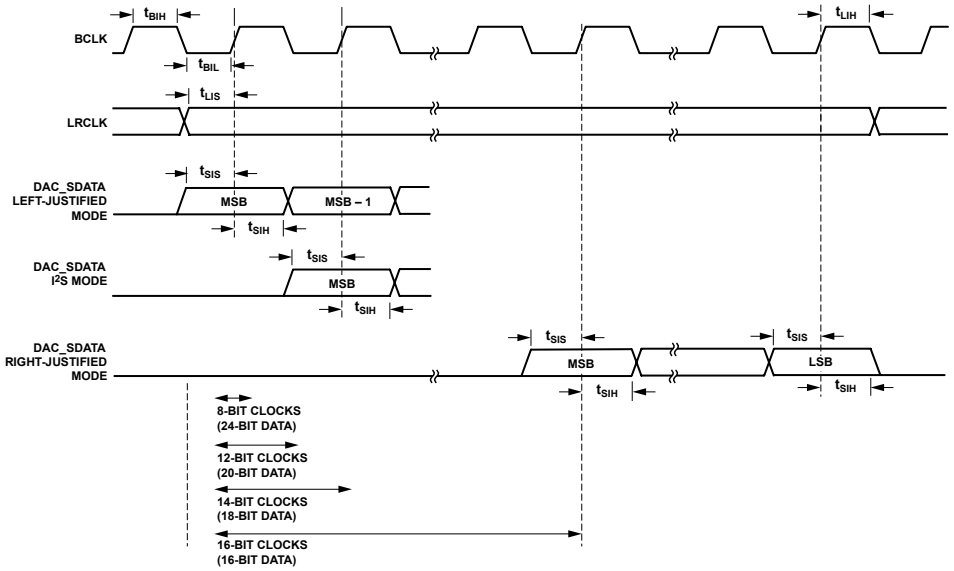

Figure 25-11: Timing Comparison Between Different Stereo Modes

Similarly, to support the 128 FS bit clock frequency, the frame sync width becomes 64 serial clock periods (bits) per channel. In this case, the delay can be a maximum of (64 – minimum serial data length in right-justified mode) = 59 bits (the minimum `SPORT_CTL_A.SLEN` setting is 4).

The starting point of the first bit is delayed so that the serial data aligns properly with the end of the channel. A 6-bit counter is added for this purpose in the stereo mode counter, which is programmed by writing into the least significant 6 bits of the `SPORT_MCTL_A.WOFFSET` field. The SPORT uses these bits to configure the window offset size in multichannel mode. Since stereo serial mode and multichannel mode are mutually exclusive, the separation of role of this field in each mode is clearly defined and implemented. The software has to configure this register with the appropriate delay. Consider the word length (`SPORT_CTL_A.SLEN`) and the number of bit clocks in one channel (left or right).

Timing Control Bits

The SPORT uses the following bits in the `SPORT_CTL_A` register to enable and configure right-justified mode.

- SLEN: serial word length select. For right-justified mode, the range of allowable word length is 5–32 bits.

- LSBF: little endian or big endian serial bit format. For right-justified mode, put serial data in big endian format (MSB bit transmitted or received first). Change the format, if necessary, depending on the application, when using non-standard protocol.
- ICLK: internal bit clock generation or external bit clock mode
- IFS: internal frame sync generation or external FS mode
- LFS: left channel first or right channel first. In standard right-justified mode, the left channel is sampled first (corresponds to SPORT_CTL_A.LFS = 0). However, the SPORT_CTL_A.LFS bit setting can be changed depending on the application to sample the right channel first (first data after falling edge of L/R clock).
- CKRE: sampling edge as rising edge or falling edge
- PACK: 16-bit to 32-bit packing enable or packing disable

NOTE: The 6 bit counter depends on SPORT_CTL_A.SLEN and the number of bit clocks in a channel. The least significant 6 bits of the SPORT_MCTL_A.WOFFSET field can also be programmed to insert delays to align data within a channel.

Serial Clock and Frame Sync Rates

Set the serial bit clock rate for internal clocks using SPORT_DIV_A.CLKDIV. Set the L/R clock rate for internal frame sync using the SPORT_DIV_A.FSDIV bit field in the same register, depending on SPORT_CTL_A.ICLK and SPORT_CTL_A.IFS bit settings.

Multichannel Mode

The SPORTs of the processor offer a multichannel mode of operation, which allows the SPORT to communicate in a time division multiplexed (TDM) serial system. In multichannel communications, each data word of the serial bit stream occupies a separate channel. For example, a 24-word block of data contains one word for each of the 24 channels.

Select the multichannel mode of the SPORT by setting SPORT_CTL_A.OPMODE = 0 and SPORT_MCTL_A.MCE = 1.

Up to 128 channels are available for transmitting or receiving. The SPORT can automatically select some words for particular channels while ignoring others. In other words, each SPORT can receive or transmit data selectively from any of the 128 channels. The 128 channels can be any 128 out of the 1024 total channels in the system. The SPORT can do any of the following on each channel:

- Transmit data (SPORT_CTL_A.SPTRAN = 1)
- Receive data (SPORT_CTL_A.SPTRAN = 0)
- Do nothing during inactive channels

Optionally, the SPORT can use data companding and DMA transfers in multichannel mode on both primary and secondary data lines.

Program the SPORT multichannel select registers (SPORT_CS0_A) before enabling SPORT operation for multichannel mode. Programming is especially important in DMA data unpacked mode, since the SPORT data buffers begin operation immediately after the SPORT data lines are enabled. Also, enable the SPORT_MCTL_A.MCE prior to enabling SPORT operation.

Multichannel mode operates independently. Each SPORT uses its own serial clock and frame sync signal that is either internally generated or externally provided.

Protocol Configuration Options

The SPORT uses the following bits in the SPORT_CTL_A and SPORT_MCTL_A registers to enable and configure multichannel mode.

- SLEN: serial word length select
- LSBF: little endian or big endian serial bit format
- ICLK: internal clock generation or external clock mode
- CKRE: sampling edge as rising edge or falling edge
- IFS: internal frame sync generation or external FS mode
- LFS: active-high FS or active-low FS
- PACK: 16-bit to 32-bit packing enable or packing disable
- MFD: multichannel frame delay
- WSIZE: number of multichannel channels
- WOFFSET: window offset size
- MCPDE: multichannel DMA packing enable

Clocking Options

In multichannel mode, the SPORTs can either accept an external serial clock or generate it internally. The SPORT uses the SPORT_CTL_A.ICLK bit to determine the selection of these options. For internally generated serial clock (SPORT_CTL_A.ICLK = 1), the SPORT uses the SPORT_DIV_A.CLKDIV bit field to configure the serial clock from the system clock.

In addition, the serial clock edge can be selected for the sampling or driving serial data and frame sync. The SPORT uses the SPORT_CTL_A.CKRE bit to select the serial clock edge.

- If SPORT_CTL_A.CKRE= 0, incoming data and frame sync signals are sampled based on the falling edge of serial clock. Data and frame sync output signals are driven at the rising edge of clock.
- If SPORT_CTL_A.CKRE = 1, incoming data and frame sync signals are sampled based on the rising edge of serial clock. Data and frame sync output signals are driven at the falling edge of clock.

Frame Sync Options

The frame sync signal synchronizes the channels and restarts each multichannel sequence. The SPORT_FS signal initiates the start of the channel 0 data word. The frame sync period in multichannel is defined as:

FS period = [(SPORT_CTL_A.SLEN + 1) × number of channels] - 1

Configure the frame sync in master or slave mode based on the setting of the SPORT_CTL_A.IFS bit. Change the logic level using the SPORT_CTL_A.LFS bit.

In multichannel mode, frame sync timing resembles late frame mode (although the SPORT_CTL_A.LAFS bit is reserved in this mode). The first bit of the transmit data word is available and the first bit of the receive data word is sampled in the same serial clock cycle. The sampling occurs when the frame sync is asserted and multichannel frame delay (SPORT_MCTL_A.MFD) is set to 0.

The block or frame-start references the frame sync signal. Then, the word transfers are performed continuously. No further frame sync signals are required during the ongoing frame for different channels. Therefore, internally generated frame syncs are always data independent (SPORT_CTL_A.DIFS bit is reserved).

Transmit Data Valid (TDV)

Each serial port has its own transmit data valid signal (SPORT_ATDV) which is active during the transmission of enabled words. Because the serial port signals are three-stated when the time slot is not active, the SPORT_ATDV signal specifies if the processor is driving the SPORT data. It serves as an output-enabled signal for the data transmit pin. After the transmit data buffer is loaded, transmission begins and the SPORT_TDV signal is asserted.

The polarity of this transmit data valid signal is always active high. SPORT_ATDV is asserted high when data is transmitted during the active channel slot of serial port.

The ***Multichannel Timing*** figure shows an example of timing for a multichannel transfer having following characteristics.

- The half SPORT pair of SPORT0, A and B, is configured as a transmitter and receiver respectively. Half SPORT A of SPORT1 is configured as transmitter.
- The serial clock and frame sync signals are input to all of these HSPORTs.
- Only primary channels of these SPORTs are enabled (0).
- Multichannel is configured to 8 channels.
- SPORT0_A drives data (on its primary data line) during slot 1–0 which asserts SPORT0_ATDV for 2 slots.
- SPORT1_A drives data (on its primary data line) during slot 3–2 which asserts SPORT1_ATDV for 2 slots.
- SPORT0_B receives data (from its primary data line) during slot 3–0.

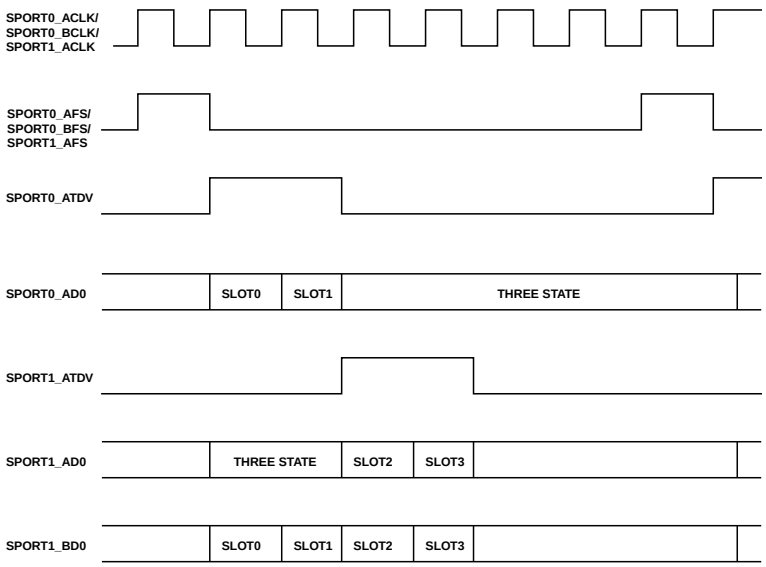

Figure 25-12: Multichannel Timing

Active Channel Selection Registers (SPORT_CS0_A)

In multichannel mode, SPORT supports up to 128 channels for transmitting or receiving. It can receive or transmit data selectively from any of the 128 channels. Specific channels can be individually enabled or disabled, using multichannel selection registers (CSx), to select the words that are transmitted or received during multichannel communications. Data words from the enabled channels are transmitted or received, while disabled channel words are three-stated or ignored.

Each of the four multichannel selection registers is 32 bits in length. Therefore, these registers provide channel selection for 128 (0–127) channels. Setting a bit enables that channel so that the serial port selects its word from the multiple-word block of data (for either receive or transmit). The 128 channels are sequentially numbered from bit 0 in the CS0 register to bit 31 of SPORT_CS3_A register. For example, setting bit 13 of the SPORT_CS1_A register enables channel number 45 (31+13+1). Similarly, setting bit 5 of the SPORT_CS3_A register enables channel number 101 (31+32+32+5+1).

Multichannel Frame Delay (MFD)

The 4-bit multichannel frame delay (SPORT_MCTL_A.MFD) field in the multichannel control registers (MCTL_x) specifies a delay between the frame sync pulse and the first data bit in frame. The value of SPORT_MCTL_A.MFD is the number of serial clock cycles of the delay. This multichannel frame delay allows the processor to work with different types of telephony interface devices.

A value of zero for SPORT_MCTL_A.MFD causes the frame sync to be concurrent with the first data bit. The maximum value allowed for SPORT_MCTL_A.MFD is 15. If SPORT_MCTL_A.MFD>0, a new frame sync can occur during the last channels of a previous frame, which is considers as valid frame sync signal.

NOTE: If more than 15-bits frame delay is required, use the field for window offset to delay the start of channel 0.

Number of Multichannel Slots (WSIZE)

Select the number of channels used in multichannel operation by using the 7-bit SPORT_MCTL_A.WSIZE field in the multichannel control register. Set SPORT_MCTL_A.WSIZE to the actual number of channels minus one (SPORT_MCTL_A.WSIZE = Number of channels - 1). So, the granularity of number of channels selected is 1.

A 10-bit field in the multichannel mode status register, SPORT_MSTAT_A, holds the channel number being serviced in the multichannel operation.

Window Offset (WOFFSET)

The window offset (SPORT_MCTL_A.WOFFSET) field register specifies where in the 1024-channel range to place the start of the active window. A value of 0 specifies no offset. 896 (1024 – 128) is the largest value that permits using all 128 channels.

As an example, a program could define an active window with 8 multichannel slots (SPORT_MCTL_A.WSIZE = 7) and an offset of 93 (SPORT_MCTL_A.WOFFSET = 93). This 8-channel window then resides in the range from 93 to 100.

Do not change the window offset or the number of multichannel slots (SPORT_MCTL_A.WSIZE) while the SPORT is enabled. If the combination of the window size and offset place any portion of the window out-of-range of the channel counter, none of these channels in the frame are enabled.

Companding Selection

Like all other operating modes, companding logic can be applied to serial data. In transmit mode, compression logic is applied to the transmit data; while in receive mode, expansion logic can be applied to the received data. The two widely used companding algorithms, A-law and μ-law can be applied by configuring SPORT_CTL_A.DTYPE field of the control register.

If companding is enabled, the companding algorithm is applied to both datapaths. In multichannel mode, companding can be applied to either all or none of the enabled channels.(Companding cannot be selected on a per-channel basis).

Multichannel DMA Data Packing (MCPDE)

Multichannel DMA data packing and unpacking are specified with the SPORT_MCTL_A.MCPDE bit setting.

If the bits are set, indicating that data is packed, the SPORT expects that the data contained by the DMA buffer corresponds only to the enabled SPORT channels. For example, if an MCM frame contains ten enabled channels, the SPORT expects the DMA buffer to contain ten consecutive words for each frame.

If the bits are cleared, the SPORT expects the DMA buffer to have a word for each of the channels in the active window, whether enabled or not. (Bits are cleared by default indicating that data is not packed). The DMA buffer size must be equal to the size of the window. For example, if only channel number 1 and 10 are enabled, then DMA buffer size is ten words. (Unless the secondary side is enabled). The data to be transmitted or received is placed at addresses 1 and 10 of the buffer. The rest of the words in the DMA buffer are ignored.

Multichannel Frame

A multichannel frame contains more than one channel, as specified by the SPORT_MCTL_A.WSIZE field and window offset field of multichannel control register. A complete multichannel frame consists of 1–1024 channels, starting with channel 0. The particular channels of the multichannel frame that are selected for the SPORT are a combination of the window offset, the window size, and the multichannel select registers.

The ***Relationships for Multichannel Parameter*** figure illustrates the relationship between different parameters of multichannel timings. The frame sync divider or external frame sync period sets the frame length.

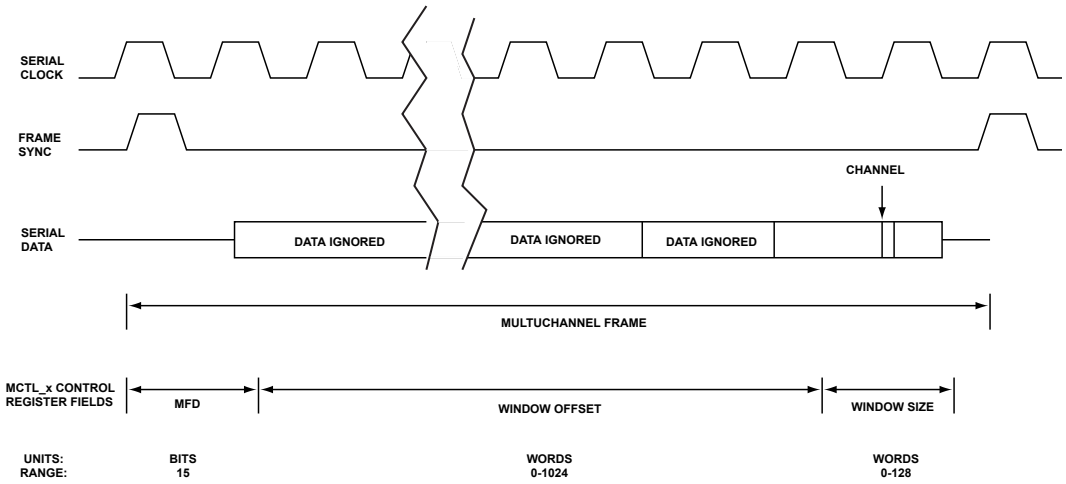

Figure 25-13: Relationships for Multichannel Parameter

Packed I²S Mode

A packed I²S mode is available in the SPORT and used for audio codec communications using multiples channels. This mode allows applications to send more than the standard 32 bits per channel available through standard I²S mode. Packed mode is implemented using standard multichannel mode (and is therefore programmed similarly to multichannel mode).

The SPORT can be configured in packed I²S mode by setting the SPORT_CTL_A.OPMODE bit and the SPORT_MCTL_A.MCE bit.

Like multichannel mode, packed I²S mode also supports the maximum of 128 channels as the maximum of (128 x 32) bits per left or right channel.

Refer to the ***Packed I²S Mode 128 Operation*** figure. Packed waveforms are the same as the waveforms used in multichannel mode, except that the frame sync is toggled for every frame and emulates I²S mode. So it is a hybrid between multichannel and I²S mode.

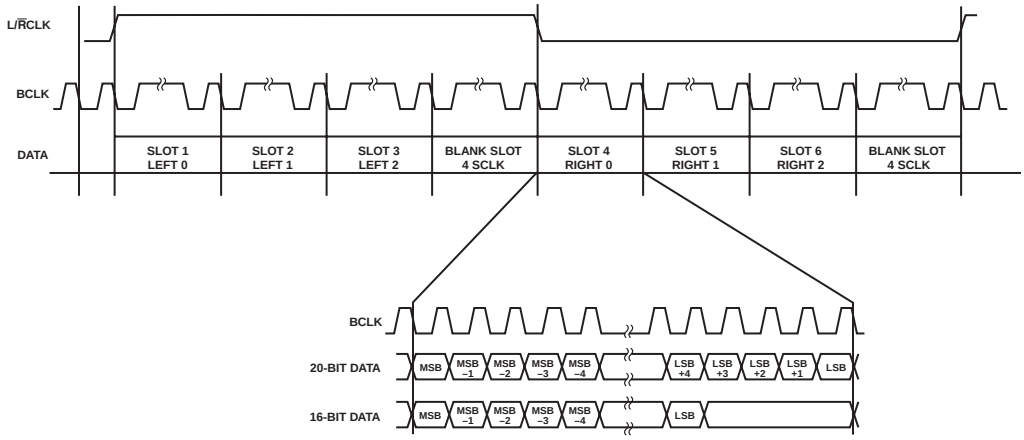

Figure 25-14: Packed I^{2}S Mode 128 Operation

Protocol Configuration Options

The SPORT uses several bits in the SPORT_CTL_A and SPORT_MCTL_A registers to enable and configure packed I^{2}S mode.

- SLEN: serial word length select
- LSBF: little endian or big endian serial bit format
- ICLK: internal clock generation or external clock mode
- CKRE: sampling edge as rising edge or falling edge
- IFS: internal frame sync generation or external FS mode
- LFS: left-channel first or right channel first
- PACK: 16-bit to 32-bit packing enable or packing disable
- MFD: multichannel frame delay
- WSIZE: number of multichannel channels
- WOFFSET: window offset size
- MCPDE: multichannel DMA packing enable

Clocking Options

In packed mode, the serial ports can either accept an external serial clock or generate it internally. The SPORT uses the SPORT_CTL_A.ICLK register to determine the selection of these options. For internally generated serial clock (SPORT_CTL_A.ICLK = 1), the SPORT uses the SPORT_DIV_A.CLKDIV bit field to configure the serial clock rate from the system clock.

The programs can also select the serial clock edge that is used for sampling or driving serial data and frame syncs. This selection is performed using the SPORT_CTL_A.CKRE bit.

Frame Sync Options

The frame sync period in packed mode is defined as:

FS period = [(SPORT_CTL_A.SLEN + 1) × number of channels] - 1.

The frame sync can be configured in master or slave mode depending on the SPORT_CTL_A.IFS bit setting of serial port control register. The logic level can be changed with the SPORT_CTL_A.LFS bit setting.

Gated Clock Mode

Some of the ADC/DACs support the SPI compatible protocol for the interface. To communicate with such ADC/DACs, the serial port must support the gated clock mode of operation, in which the data valid information is embedded into the clock. Therefore, in gated clock mode, the clock should be active only during active data transmission or reception.

The processor features the gated clock function. The serial port uses the SPORT_CTL_A.GCLKEN (Gated clock mode select) control bit to configure gated clock mode. To enable gated clock mode of operation, program the SPORT to comply with the following requirements.

- Gated clock mode feature is available in standard serial mode, left-justified and I^2S mode of operation only.

  Among the stereo modes, the serial port cannot operate right justified mode in gated clock mode. During the inactive period in a frame, there is a delay counter running inside the serial port. (The inactive period is the period between the leading edge of the frame sync and the first active data bit). The counter operates on the serial clock and any interruption in clock makes the counter go out of sync.

- Gated clock mode has the following valid settings for other control bits.
  - Both serial clock and frame sync signals generated internally
  - Both serial clock and frame sync signals provided externally
  - Frame sync not required mode (SPORT_CTL_A.FSR = 0) not supported
  - Program SPORT_CTL_A.DIFS as 0 in transmit mode; program it as 1 in receive mode.
- Satisfy the following necessary conditions when gated clock mode is enabled:
  - Need at least 7 serial clock cycles between enabling the SPORT and first frame sync. If this requirement is not met, the SPORT can drop the first data. (For subsequent data, this requirement is not applicable).
  - In external clock and external frame sync mode, the frame sync must be inactive during clock synchronization, after the SPORT has been enabled.
  - For edge detected frame sync, the frame sync must transition back to inactive state before the current word transmission or reception is complete (or when the clock is still running). If this requirement is not met, the SPORT does not recognize the next valid frame sync and skips the channel. The SPORT continues to skip the frame syncs until the frame sync transitions back to an inactive state when the clock is active.

Data Transfers

Serial port data can be transferred to or from internal or external memory in two different methods:

- Core-driven single word transfers
- DMA-driven multiple words transfers with multiple work units

DMA transfers can be set up to transfer a configurable number of serial words between the serial port transmit or receive data buffers and internal memory automatically. Core-driven transfers use SPORT interrupts to signal the processor core to perform single word transfers to or from the serial port data buffers.

The following sections provide information on core-driven and DMA-driven data transfers.

Data Buffers

When programming the serial port data channels (primary or secondary) as a transmitter by setting SPORT_CTL_A.SPTRAN = 1, only the corresponding transmit data buffers (SPORT_TXPRI_A and SPORT_TXSEC_A) become active. The receive data buffers (SPORT_RXPRI_A and SPORT_RXSEC_A) remain inactive. Similarly, when the SPORT data channels are programmed for receive operation (SPORT_CTL_A.SPTRAN = 0), then only corresponding receive data buffers (SPORT_RXPRI_A and SPORT_RXSEC_A) are active. Do not attempt to read or write inactive data buffers. If the processor operates on the inactive transmit or receive buffers while the SPORT is enabled, unpredictable results can occur.

Each of these buffers is 32-bit wide (corresponds to maximum serial data word length). When using word lengths less than 32 bits for SPORT operation, the data in these buffers is automatically right-justified. (The LSB bit of data is at the bit 0 location of the buffer). The upper unused bits can be zero-filled or sign-extended depending on SPORT_CTL_A.DTYPE field.

Transmit Data Buffers (SPORT_TXPRI_A and SPORT_TXSEC_A)

When enabled as a transmitter (SPORT_CTL_A.SPTRAN = 1), each SPORT half has its own set of transmit data buffers. The primary (0) and secondary (1) datapaths of each SPORT half have separate data buffers, referred to as SPORT_TXPRI_A and SPORT_TXSEC_A respectively.

These transmit data buffers are 32 bits wide. Load these buffers with the data for transmission on the primary and secondary data channels. The DMA controller loads the data automatically. Or, the program running on the processor core loads the data manually.

Together with the output shift register, transmit data buffers act like a two-location FIFO. If data packing is disabled (SPORT_CTL_A.PACK = 0), the transmit path can hold as many as three data words. If data packing is enabled (SPORT_CTL_A.PACK = 1), it can hold two packed data words at any given time.

When the transmit shift register becomes empty (transfer out all the bits of previous word), data in the transmit data buffer is automatically loaded into it. An interrupt occurs when the output transmit shift register has been loaded, signifying that the transmit data buffer is empty and ready to accept the next word. This interrupt does not occur when serial port is operating in DMA mode or when the corresponding interrupt enable mask bit is set.

If only the primary datapath of a SPORT half is enabled, programs must not write to the inactive secondary transmit data buffer and conversely. If the core keeps writing to the inactive buffer, the status of that transmit buffer becomes full. This state can cause the core to hang indefinitely, since data is never transmitted to the output shift register.

Receive Data Buffers (SPORT_RXPRI_A and SPORT_RXSEC_A)

When enabled as receiver (SPORT_CTL_A.SPTRAN = 0), each SPORT half has its own set of receive data buffers. The primary (0) and secondary (1) datapaths of each SPORT half have separate data buffers, referred as SPORT_RXPRI_A and SPORT_RXSEC_A respectively. Together with input shift register, the receive data buffers act like a three-location FIFO, as the receive path has two data registers.

These receive data buffers are the 32 bits wide. These buffers are automatically loaded from the receive shift register when a complete word has been received into it. An interrupt occurs when the receive data buffer is loaded, signifying that new data is available in the receive data buffer and is ready to read. This interrupt does not occur when the serial port is operating in DMA mode or when the corresponding interrupt enable mask bit is set.

If only the primary datapath of a SPORT half is enabled, programs must not read from the inactive secondary receive data buffer and conversely. If the core keeps reading from the inactive buffer, the status of that receive buffer becomes empty. This state can cause the core to hang indefinitely since new data is never received through the input shift register.

Data Buffer Status

Serial ports provide status information about data buffers through the SPORT_CTL_A.DXSPRI (primary channel data buffer status) and SPORT_CTL_A.DXSSEC (secondary channel data buffer status) bits. Serial ports provide error status through SPORT_CTL_A.DERRPRI (primary channel error status) and SPORT_CTL_A.DERRSEC (secondary channel error status) bits. Depending on the SPORT_CTL_A.SPTRAN bit setting, these bits reflect the status of either SPORT_TXPRI_A and SPORT_TXSEC_A transmit data buffers or SPORT_RXPRI_A and SPORT_RXSEC_A receive data buffers. These bits indicate whether the buffers are full, partially full, or empty.

When attempting to read from an empty receive buffer or to write to a full transmit buffer, the SPORT delays access until the buffer is ready. This delay is called a core processor hang. To avoid these conditions, always check the buffer status to determine if the access can be made. The SPORT updates the status bits in the SPORT_CTL_A register during reads and writes from the core processor even when the serial port is disabled.

Two complete 32-bit words can be stored in the receive buffer while a third word shifts in. Therefore, almost three complete words can be received without the receive buffer being read before an overflow occurs. After receiving the third word completely, the shift register contents overwrite the second word. This operation occurs if the first word has not been read out (by the processor core or the DMA controller). The receive overflow status is flagged through the error status bits of the SPORT_CTL_A register. The overflow status is generated on the last bit of the third word. The SPORT_CTL_A.DERRPRI and SPORT_CTL_A.DXSSEC status bits are sticky read-only bits. The bits are cleared by disabling the serial port.

NOTE: The status bits in the SPORT_CTL_A register are updated during reads and writes from the core processor even when the serial port is disabled.

Data Buffer Packing

When the SPORT is configured as a receiver with a serial data word length of 16 or less, the received data words can be packed into a 32-bit word. Similarly, if the SPORT is configured as transmitter with a serial data word length of 16 or less, then 32-bit words being transmitted can be unpacked into 16-bit words. The SPORT uses the SPORT_CTL_A.PACK bit to select this packing or unpacking feature.

When SPORT_CTL_A.PACK = 1, two successive words received are packed into a single 32-bit word, or each 32-bit word is unpacked and transmitted as two 16-bit words. The first 16-bit (or smaller) word is right-justified in bits 15–0 of the packed word, and the second 16-bit (or smaller) word is right-justified in bits 31–16. This packing method applies to both receive (packing) and transmit (unpacking) operations. In this case, the transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.

Companding can be used with word packing or unpacking.

NOTE: When 16-bit received data is packed into 32-bit words and stored in normal word space in processor internal memory, the 16-bit words can be read or written with short word space addresses.

Single Word (Core) Transfers

The serial ports can also transmit or receive individual data words with interrupts occurring as each data word is transmitted or received. When a serial port is enabled and corresponding DMA is disabled, the SPORT interrupts are generated when:

- A complete word has been received in the receive data buffer, or
- The transmit data buffer is not full

When performing core driven transfers, write to the buffer designated by the SPORT_CTL_A.SPTRAN bit setting. For DMA driven transfers, the SPORT logic performs the data transfer from internal memory to or from the appropriate buffer depending on the SPORT_CTL_A.SPTRAN bit setting. If the inactive SPORT data buffers are read or written to by the core while the SPORT is being enabled, the core can hang. For example, if a SPORT is programmed to be a transmitter, while the core reads from the receive buffer of the same SPORT, the core can hang. The core hangs as if it were reading an empty buffer that is active and waiting for the status to change. These events can lock up the core until the SPORT is reset.

Avoid hanging the processor core. Check the status of appropriate data buffers when the processor core tries to read a word from a serial port receive buffer or writes to its transmit buffer. The SPORT can read the full or empty status using the SPORT_CTL_A.DXSPRI and SPORT_CTL_A.DXSSEC bits.

DMA Transfers

Direct memory access (DMA) provides a mechanism for receiving or transmitting an entire block of serial data before an interrupt is generated. When SPORT DMA is not enabled, the SPORT generates an interrupt every time it

receives or starts to transmit a data word. The on-chip DMA controller of the processor handles the DMA transfer. It allows the processor core to continue running until the entire block of data is transmitted or received. Service routines can then operate on the block of data rather than on single words, significantly reducing overhead.

Therefore, set the direction bit, DMA enable bit, and the serial port enable bit before initiating any operations on the SPORT data buffers. Do not try to access data buffers when the associated DMA channel of serial port is enabled. If the processor operates on the inactive transmit or receive buffers while the SPORT is enabled, it can cause unpredictable results.

Each SPORT half has a dedicated DMA channel, which serves both primary and secondary datapaths. In transmit mode, the DMA channel alternately writes to the primary transmit data buffer and the secondary transmit data buffer. Software must interleave the data of primary and secondary channels into the transmit buffer of the DMA. Similarly, in receive mode, the DMA channel alternately reads from the primary receive data buffer and the secondary data buffer. Software must de-interleave the data corresponding to the primary and secondary channels from the receive buffer of the DMA.

If the SPORT is configured in stereo mode, the same DMA channel drives or receives both the left and right channels of the enabled datapaths (primary or secondary). Therefore, in transmit mode, software must interleave the data of left and right channels (of the enabled datapaths) into the transmit buffer of the DMA. Similarly in receive mode, software must de-interleave the left and right channel data (of the enabled paths) from the receive buffer of the DMA.

Since both primary and secondary datapaths share the single DMA channel, each SPORT half share a common interrupt vector. Optionally, the DMA controller can generate an interrupt at the end of the completion of a DMA transfer and at the end of each work unit of DMA.

The SPORT DMA channels are assigned a higher priority than all other DMA channels (for example, the SPI port). The priority is due to their relatively low service rate and inability to hold off incoming data. Having higher priority causes the SPORT DMA transfers to execute first when multiple DMA requests occur in the same cycle. The serial port DMA channels are numbered and prioritized in the DMA channel list table in the DMA chapter for the processor.

Although the DMA transfers execute with 32-bit words, the SPORTs can handle word sizes from 4 to 32 bits (as defined by `SPORT_CTL_A.SLEN` field). If serial data length is 16 bits or smaller, two data can pack into 32-bit words for each DMA transfer. This option is selected by setting the `SPORT_CTL_A.PACK` bit. When serial port data packing is enabled (`SPORT_CTL_A.PACK` = 1),the SPORT generates the transmit and receive interrupts for the 32-bit packed words, not for each 16-bit word. For more information, see Data Buffer Status.

NOTE: The DMA channel of a SPORT can access both internal memory and external memory of the processor without any core overhead.

Error Detection

When the serial port is configured as a transmitter, the `SPORT_CTL_A.DERRPRI` (primary channel error status) bit provides transmit data buffer underflow status for the primary datapath. Similarly, the `SPORT_CTL_A.DERRSEC` (secondary channel error status) bit provides transmit data buffer underflow status for the secondary datapath.

(These events indicate that the frame sync signal occurred when the transmit data buffer was empty). The serial port transmits data whenever it detects a framing signal.

- 0 = No frame sync signal occurred when TX data buffer is empty (no underflow).
- 1 = Framing signal occurred when TX buffer was empty (underflow).

Similarly, if SPORT configured as a receiver, the SPORT_CTL_A.DERRPRI and SPORT_CTL_A.DERRSEC bits provide receive overflow status of primary and secondary receive data buffers. In other words, the SPORT indicates that a channel has received new data when the receive buffer is full, so new data overwrites existing data. The serial port receives data whenever it detects a framing signal.

- 0 = No frame sync signal occurred when RX data buffer is full (no overflow).
- 1 = Frame sync signal occurred when RX data buffer was full (overflow).

In addition to status flagging for underflow and overflow errors, each SPORT half contains an error register (SPORT_ERR_A) and a dedicated interrupt channel, known as status interrupt. This interrupt can be optionally triggered based on the error status of primary and secondary data lines as reflected in SPORT_ERR_A.DERRPSTAT and SPORT_ERR_A.DERRSSTAT bits respectively. The SPORT_ERR_A.DERRPMSK (primary channel data error interrupt enable) and SPORT_ERR_A.DERRSMSK (secondary channel data error interrupt enable) bits unmask the status interrupt for primary and secondary data errors.

In addition to data underflow and data overflow errors, the status interrupt is also triggered optionally when a frame sync error is detected. The SPORT_ERR_A.FSERRMSK (frame sync error interrupt enable) bit unmasks the status interrupt for this frame sync error. This frame sync error is generated because of premature frame sync, as explained in Premature Frame Sync Error Detection section.

A frame sync error is not detected in following cases.

- When there is no active data transmit or receive and the frame sync pulse occurs due to noise in the input signal.
- If there is an underflow or overflow error. SPORT error logic does not run (the bit count is not set and decremented) when there is an underflow error. Therefore, frame sync errors cannot be detected.
- When the frame sync pulse > system clock period.

The channel error status bits, SPORT_CTL_A.DERRPRI, and SPORT_CTL_A.DERRSEC, in the control register are sticky read-only bits. Clear the bits in one of two ways:

- By disabling the SPORT (for frame sync error) or disabling the corresponding channel only (for SPORT_CTL_A.DERRPRI, SPORT_CTL_A.DERRSEC).
- By writing to R/W the SPORT_ERR_A.FSERRSTAT, SPORT_ERR_A.DERRPSTAT or SPORT_ERR_A.DERRSSTAT status bits.

When sticky bits are cleared, interrupts are also cleared.

Interrupts

This section describes the various scenarios in which an interrupt is generated. Both the core and DMA are able to generate data interrupts for receive or transmit operations. Moreover, the SPORT modules generate error conditions which have a separate status interrupt.

Internal Transfer Completion

Each SPORT half has an interrupt associated with it. Both primary and secondary data channels share the interrupt vector, regardless of whether they are configured as a transmitter or receiver. To determine the source of an interrupt, applications can check the transmit or receive data buffer status (`SPORT_CTL_A.DXSPRI`, `SPORT_CTL_A.DXSSEC`) bits. In core mode, this interrupt signifies that either the transmit data buffer is empty (when `SPORT_CTL_A.SPTRAN` = 1) or new data is available in the receive data buffer (when `SPORT_CTL_A.SPTRAN` = 0). When serial port data packing is enabled (`SPORT_CTL_A.PACK` = 1), the transmit and receive interrupts are generated for 32-bit packed words, not for each 16-bit word.

The same interrupt can be used to indicate the completion of the transfer of a block of serial data when the serial ports are configured for DMA. The count register of DMA must be initialized with a word count that specifies the number of words to transfer. The count register decrements after each DMA transfer on the channel. When the word count reaches zero (or if a work unit has finished), the DMA completion interrupt is generated.

Multiple interrupts can occur if both data channels of serial port transmit or receive data in the same cycle. Any interrupt can be masked in the `IMASK` register.

Transfer Finish Interrupt (TFI)

When a serial port in DMA mode is configured as a transmitter (`SPORT_CTL_A.SPTRAN` = 1), the SPORT can use the transmit finish interrupt to signal the end of the transmission. Set `SPORT_CTL_A.TFIEN` bit to enable this feature. When DMA transfers the last word to the FIFO, it also gives a signal to the SPORT indicating DMA has finished. The SPORT uses this information and then waits until all the data in the FIFO is transmitted out (including the transmit shift register) and generates the transmit finish interrupt. Configure the interrupt type field in the DMA configuration register for peripheral interrupt.

SPORT Programming Model

The programming model describes some programming procedures that are used to enable and operate the SPORTs. See the following topics:

- Setting Up and Starting Transfers
- Setting Up and Starting Multichannel Transfers
- Using DMA for SPORT Transfers
- Using Companding as a Function

Setting Up and Starting Transfers

When in Standard Serial Mode, I²S Mode, Left-Justified Mode, or Right-Justified Mode; the SPORT is in a normal (standard and stereo) operational mode. The Gated Clock Mode (an additional clock mode feature) is available for these normal operational modes, except for right-justified mode.

NOTE: For brevity, this example uses half SPORT A registers. With appropriate changes to register and bit usage, this example also applies for half SPORT B.

1. Clear the SPORT_CTL_A and SPORT_MCTL_A registers.

   ADDITIONAL INFORMATION: Clearing these registers disables the primary or secondary transmit/receive for the half SPORT, disables transmit/receive buffers, clears all mode or option selections, and disables multi-channel operation.

2. Write the SPORT_DIV_A register to set up the needed serial clock and frame sync options for the design.

   ADDITIONAL INFORMATION: The SPORT supports many clocking and frame sync options. For more information, see the descriptions in the "Operating Modes" section and refer to the "Register Descriptions" section.

3. Write the SPORT_CTL_A register to set up the standard serial or stereo operating mode, clock and frame sync options, serial word length, and data options.

   ADDITIONAL INFORMATION: The SPORT supports many options for standard serial mode and stereo mode transfers and also for emulating any non-standard protocol. For more information, see the descriptions in Operating Modes and refer to the "Register Descriptions" section.

4. Configure the SPORT_CTL2_A register, if coupling with a paired half SPORT is required.

   ADDITIONAL INFORMATION: The SPMUX logic supports sharing clock and frame sync signals between the paired half SPORTs.

5. Write to SPORT_ERR_A register if an interrupt on SPORT error conditions is required.

   ADDITIONAL INFORMATION: The error interrupt can be generated on underrun or overflow errors on primary or secondary SPORT channels. Also, an interrupt may be generated to detect a premature frame sync.

6. Write the DCNT field (right-justified mode usage of the SPORT_MCTL_A.WOFFSET bit field) with the required delay counter, if the half SPORT is configured for right-justified mode.

7. Enable the primary/or secondary channels of the half SPORT by setting respective bits of SPORT_CTL_A register.

   ADDITIONAL INFORMATION: Note that the write to the SPORT_CTL_A register (enabling primary and/or secondary for transmit or receive) enables the corresponding data buffers.

8. If the SPORT is to be controlled in core mode, write data to be transmitted to the transmit buffer (SPORT_TXPRI_A and/or SPORT_TXSEC_A) if the SPORT is configured as a transmitter *or* read data that has

been received from the receive buffer (SPORT_RXPRI_A and/or SPORT_RXSEC_A) if the SPORT is configured as a receiver.

ADDITIONAL INFORMATION: Do not attempt to read or write inactive data buffers. If the processor operates on the inactive transmit or receive buffers while the SPORT is enabled, unpredictable results may occur.

9. If the SPORT is used with a DMA operation, do not access the SPORT_TXPRI_A and/or SPORT_TXSEC_A data registers.

Setting Up and Starting Multichannel Transfers

When in Multichannel Mode or Packed I²S Mode, the SPORT is in a multichannel operational mode. The Gated Clock Mode (an additional clock mode feature) *is not available* for multichannel channel operational mode.

NOTE: For brevity, this example uses half SPORT A registers. With appropriate changes to register and bit usage, this example also applies for half SPORT B.

1. Clear the SPORT_CTL_A and SPORT_MCTL_A registers.

   ADDITIONAL INFORMATION: Clearing these registers disables the primary/secondary transmit/receive for the half SPORT, disables transmit/receive buffers, clears all mode/option selections, and disables multichannel operation.

2. Write the SPORT_DIV_A register to set up the needed serial clock and frame sync options for the design.

   ADDITIONAL INFORMATION: The SPORT supports many clocking and frame sync options. For more information, see the descriptions in Operating Modes and refer to the "Register Descriptions" section.

3. Write the SPORT_MCTL_A register to set up the needed multichannel options for the design. Write the bits of the SPORT_CS0_A through SPORT_CS3_A registers as needed to enable the selected channels.

   ADDITIONAL INFORMATION: The SPORT supports many multichannel options. For more information, see the descriptions in Operating Modes and refer to the "Register Descriptions" section.

4. Write the SPORT_CTL_A register to set up the multichannel mode or packed I²S mode, clock and frame sync options, serial word length and data options.

   ADDITIONAL INFORMATION: The SPORT supports many options for multichannel mode or packed I²S mode. For more information, see the descriptions in Operating Modes and refer to the "Register Descriptions" section.

5. Configure the SPORT_CTL2_A register, if coupling with paired half SPORT is required.

   ADDITIONAL INFORMATION: The SPMUX logic supports sharing clock and frame sync signals between the paired half SPORTs.

6. Write to the SPORT_ERR_A register if an interrupt on a SPORT error conditions is required.

ADDITIONAL INFORMATION: The error interrupt can be generated on underrun or overflow errors on primary or secondary SPORT channels. Also, an interrupt may be generated detection of a premature frame sync.

7. Set the SPORT_MCTL_A.MCE bit to enable multichannel mode, then enable the primary/or secondary channels of SPORT by setting respective bits of the SPORT_CTL_A register.

   ADDITIONAL INFORMATION: The write to the SPORT_CTL_A register (enabling primary and/or secondary for transmit or receive) enables the corresponding data buffers. For multichannel modes, it is recommended to use DMA mode of operation. For information, see Using DMA for SPORT Transfers.

Using DMA for SPORT Transfers

The SPORT supports DMA operations in all operation modes (Standard Serial Mode, I^2S Mode, Left-Justified Mode, Right-Justified Mode, Multichannel Mode or Packed I^2S Mode). To use DMA with the SPORT, follow the steps described in this section after setting up and starting the SPORT. (See Setting Up and Starting Transfers or Setting Up and Starting Multichannel Transfers). Instead of using the single word read or write operations described in those topics, the DMA performs the read/write access to the SPORT data buffers.

NOTE: For brevity, this example uses half SPORT A registers. With appropriate changes to register and bit usage, this example also applies for half SPORT B.

1. Use the steps in Setting Up and Starting Transfers or Setting Up and Starting Multichannel Transfers to set up and start SPORT operations.
2. Set up and start the DMA process as described in the Direct Memory Access (DMA) chapter.

   ADDITIONAL INFORMATION: It is important to apply SPORT data buffer status as a control for the DMA operation. If the routine monitoring DMA operation does not include monitoring for buffer overflow (on transmit) and buffer underrun (on receive), these conditions could cause the core to hang indefinitely. For more information, see Data Buffers.

Using Companding as a Function

Because the values in the transmit and receive buffers are actually companded in place, the companding hardware can be used without transmitting (or receiving) any data, for example during testing or debugging. For companding to execute properly, program the SPORT registers prior to loading data values into the SPORT buffers.

NOTE: For brevity, this example uses half SPORT A registers. With appropriate changes to register and bit usage, this example also applies for half SPORT B.

To compress data in place without transmitting use the following procedure.

1. Set the serial port as transmitter (SPORT_CTL_A.SPTRAN =1) with both primary and secondary data channels disabled (SPORT_CTL_A.SPENPRI =0, SPORT_CTL_A.SPENSEC =0).
2. Enable companding in the SPORT_CTL_A.DTYPE field.

3. Write a 32-bit data word to the transmit buffer
4. Wait for two system clock cycles. Any instruction not accessing the transmit buffer can be used to cause this delay. This allows the serial port companding hardware to reload the transmit buffer with the companded value.
5. Read the 8-bit compressed value from the transmit buffer.

To expand data in place, use the same sequence of operations with the receive buffer instead of the transmit buffer. When expanding data in this way, set the appropriate serial word length (`SPORT_CTL_A.SLEN`) bit.

ADSP-CM40x SPORT Register Descriptions

Serial Port (SPORT) contains the following registers.

Table 25-11: ADSP-CM40x SPORT Register List

Name	Description
SPORT_CS0_A	Half SPORT 'A' Multi-channel 0-31 Select Register
SPORT_CS0_B	Half SPORT 'B' Multi-channel 0-31 Select Register
SPORT_CS1_A	Half SPORT 'A' Multi-channel 32-63 Select Register
SPORT_CS1_B	Half SPORT 'B' Multi-channel 32-63 Select Register
SPORT_CS2_A	Half SPORT 'A' Multi-channel 64-95 Select Register
SPORT_CS2_B	Half SPORT 'B' Multichannel 64-95 Select Register
SPORT_CS3_A	Half SPORT 'A' Multi-channel 96-127 Select Register
SPORT_CS3_B	Half SPORT 'B' Multichannel 96-127 Select Register
SPORT_CTL2_A	Half SPORT 'A' Control 2 Register
SPORT_CTL2_B	Half SPORT 'B' Control 2 Register
SPORT_CTL_A	Half SPORT 'A' Control Register
SPORT_CTL_B	Half SPORT 'B' Control Register
SPORT_DIV_A	Half SPORT 'A' Divisor Register
SPORT_DIV_B	Half SPORT 'B' Divisor Register
SPORT_ERR_A	Half SPORT 'A' Error Register
SPORT_ERR_B	Half SPORT 'B' Error Register
SPORT_MCTL_A	Half SPORT 'A' Multi-channel Control Register
SPORT_MCTL_B	Half SPORT 'B' Multi-channel Control Register
SPORT_MSTAT_A	Half SPORT 'A' Multi-channel Status Register
SPORT_MSTAT_B	Half SPORT 'B' Multi-channel Status Register
SPORT_RXPRI_A	Half SPORT 'A' Rx Buffer (Primary) Register

Table 25-11: ADSP-CM40x SPORT Register List (Continued)

Name	Description
SPORT_RXPRI_B	Half SPORT 'B' Rx Buffer (Primary) Register
SPORT_RXSEC_A	Half SPORT 'A' Rx Buffer (Secondary) Register
SPORT_RXSEC_B	Half SPORT 'B' Rx Buffer (Secondary) Register
SPORT_TXPRI_A	Half SPORT 'A' Tx Buffer (Primary) Register
SPORT_TXPRI_B	Half SPORT 'B' Tx Buffer (Primary) Register
SPORT_TXSEC_A	Half SPORT 'A' Tx Buffer (Secondary) Register
SPORT_TXSEC_B	Half SPORT 'B' Tx Buffer (Secondary) Register

Half SPORT 'A' Multi-channel 0-31 Select Register

The SPORT_CS0_A register's bits (when set, =1) each corresponds to an active channel for the half SPORT in multi-channel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register de-activates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

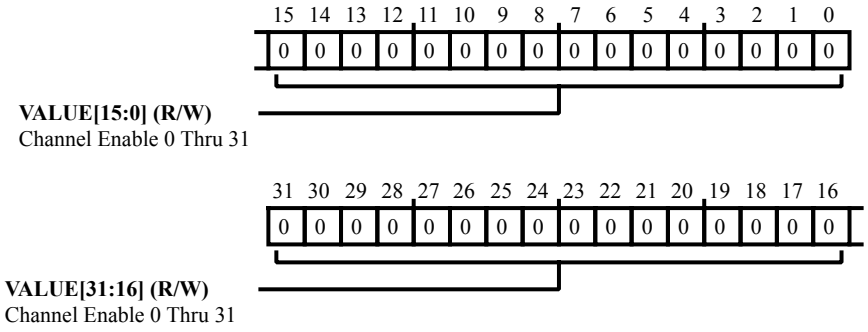

Figure 25-15: SPORT_CS0_A Register Diagram

Table 25-12: SPORT_CS0_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 0 Thru 31.

Half SPORT 'B' Multi-channel 0-31 Select Register

The SPORT_CS0_B register's bits (when set, =1) each corresponds to an active channel for the half SPORT in multi-channel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register de-activates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

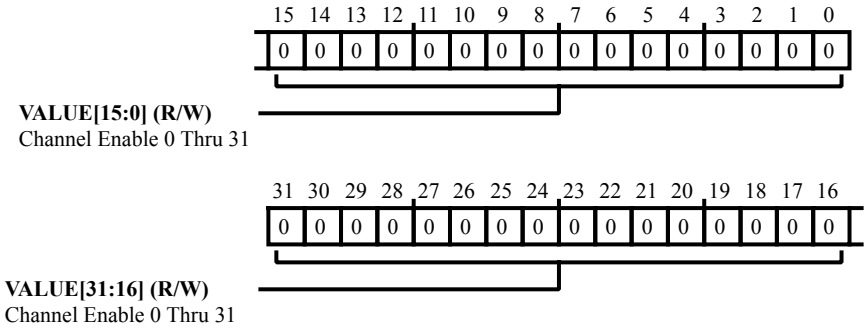

Figure 25-16: SPORT_CS0_B Register Diagram

Table 25-13: SPORT_CS0_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 0 Thru 31.

Half SPORT 'A' Multi-channel 32-63 Select Register

The SPORT_CS1_A register's bits (when set, =1) each corresponds to an active channel for the half SPORT in multi-channel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register de-activates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

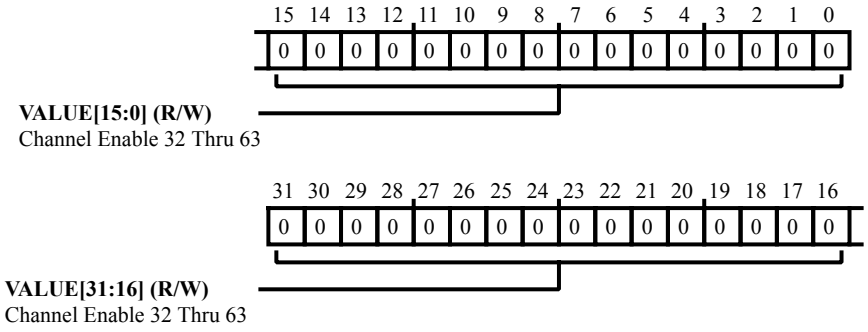

Figure 25-17: SPORT_CS1_A Register Diagram

Table 25-14: SPORT_CS1_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 32 Thru 63.

Half SPORT 'B' Multi-channel 32-63 Select Register

The SPORT_CS1_B register's bits (when set, =1) each corresponds to an active channel for the half SPORT in multi-channel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register de-activates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

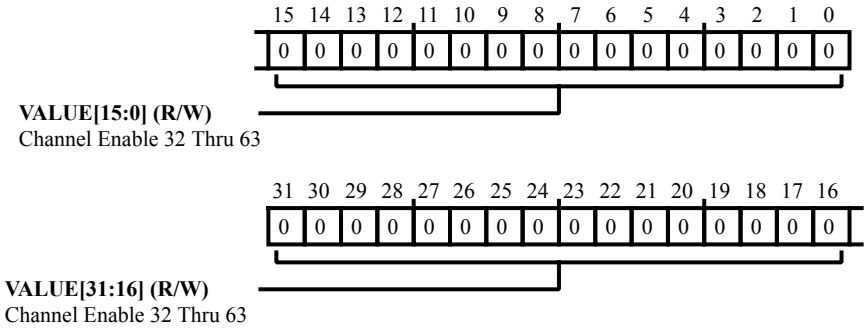

Figure 25-18: SPORT_CS1_B Register Diagram

Table 25-15: SPORT_CS1_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 32 Thru 63.

Half SPORT 'A' Multi-channel 64-95 Select Register

The SPORT_CS2_A register's bits (when set, =1) each corresponds to an active channel for the half SPORT in multi-channel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register de-activates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

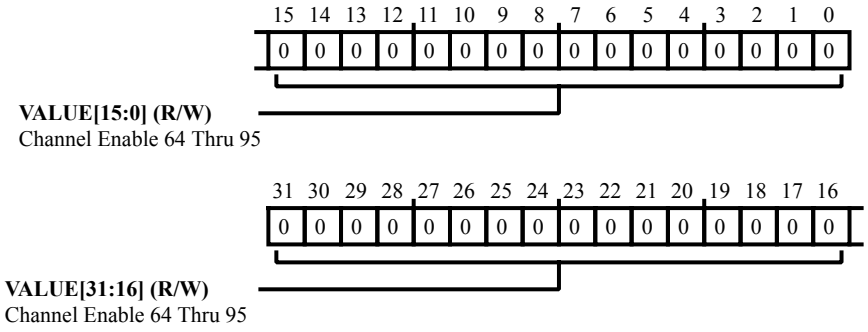

Figure 25-19: SPORT_CS2_A Register Diagram

Table 25-16: SPORT_CS2_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 64 Thru 95.

Half SPORT 'B' Multichannel 64-95 Select Register

The SPORT_CS2_B register's bits (when set, =1) each corresponds to an active channel for the half SPORT in multi-channel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register de-activates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

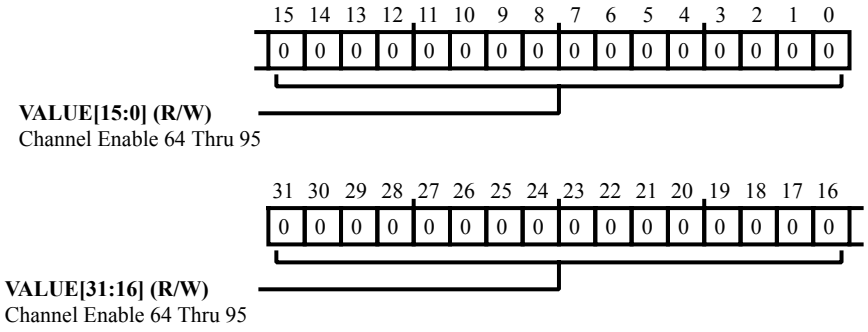

Figure 25-20: SPORT_CS2_B Register Diagram

Table 25-17: SPORT_CS2_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 64 Thru 95.

Half SPORT 'A' Multi-channel 96-127 Select Register

The SPORT_CS3_A register's bits (when set, =1) each corresponds to an active channel for the half SPORT in multi-channel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register de-activates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

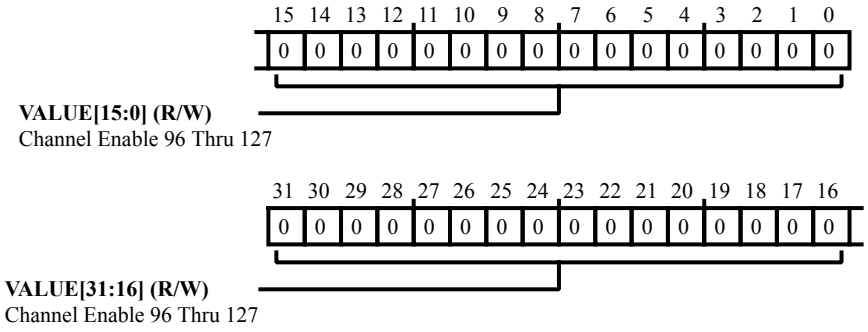

Figure 25-21: SPORT_CS3_A Register Diagram

Table 25-18: SPORT_CS3_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 96 Thru 127.

Half SPORT 'B' Multichannel 96-127 Select Register

The SPORT_CS3_B register's bits (when set, =1) each corresponds to an active channel for the half SPORT in multi-channel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register de-activates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

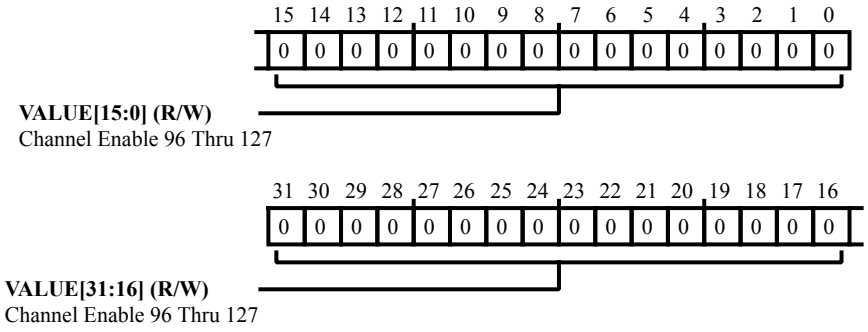

Figure 25-22: SPORT_CS3_B Register Diagram

Table 25-19: SPORT_CS3_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 96 Thru 127.

Half SPORT 'A' Control 2 Register

The SPORT_CTL2_A register controls multiplexing options for sharing serial clock and frame sync signals across the related half SPORTs.

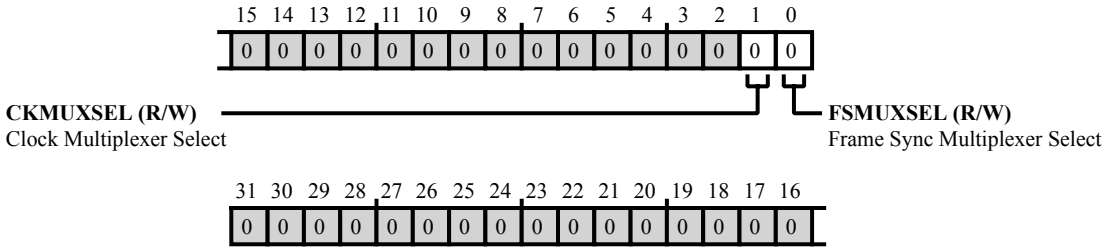

Figure 25-23: SPORT_CTL2_A Register Diagram

Table 25-20: SPORT_CTL2_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	CKMUXSEL	Clock Multiplexer Select. The SPORT_CTL2_A.CKMUXSEL bit enables multiplexing of the half SPORT' serial clock. In this mode, the serial clock of the related half SPORT is used instead of the half SPORT's own serial clock. For example, if SPORT_CTL2_A.CKMUXSEL is enabled, half SPORT 'A' uses SPORT_BCLK instead of SPORT_ACLK.	
		0	Disable serial clock multiplexing
		1	Enable serial clock multiplexing
0 (R/W)	FSMUXSEL	Frame Sync Multiplexer Select. The SPORT_CTL2_A.FSMUXSEL bit enables multiplexing of the half SPORT' frame sync. In this mode, the frame sync of the related half SPORT is used instead of the half SPORT's own frame sync. For example, if SPORT_CTL2_A.FSMUXSEL is enabled, half SPORT 'A' uses SPORT_BFS instead of SPORT_AFS.	
		0	Disable frame sync multiplexing
		1	Enable frame sync multiplexing

Half SPORT 'B' Control 2 Register

The SPORT_CTL2_B register controls multiplexing options for sharing serial clock and frame sync signals across the related half SPORTs.

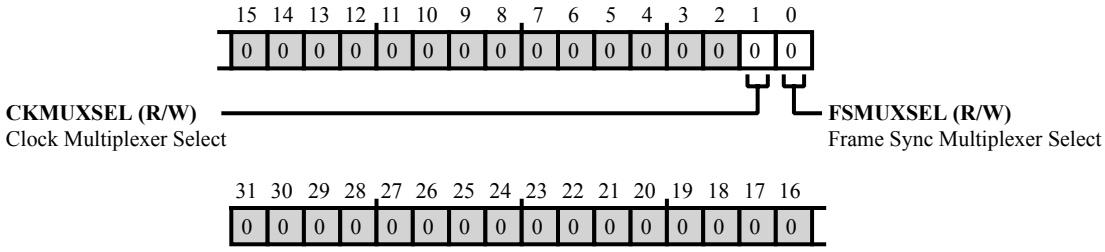

Figure 25-24: SPORT_CTL2_B Register Diagram

Table 25-21: SPORT_CTL2_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	CKMUXSEL	Clock Multiplexer Select. The SPORT_CTL2_B.CKMUXSEL bit enables multiplexing of the half SPORT' serial clock. In this mode, the serial clock of the related half SPORT is used instead of the half SPORT's own serial clock. For example, if SPORT_CTL2_B.CKMUXSEL is enabled, half SPORT 'B' uses SPORT_ACLK instead of SPORT_BCLK.	
		0	Disable serial clock multiplexing
		1	Enable serial clock multiplexing
0 (R/W)	FSMUXSEL	Frame Sync Multiplexer Select. The SPORT_CTL2_B.FSMUXSEL bit enables multiplexing of the half SPORT' frame sync. In this mode, the frame sync of the related half SPORT is used instead of the half SPORT's own frame sync. For example, if SPORT_CTL2_B.FSMUXSEL is enabled, half SPORT 'B' uses SPORT_AFS instead of SPORT_BFS.	
		0	Disable frame sync multiplexing
		1	Enable frame sync multiplexing

Half SPORT 'A' Control Register

The SPORT_CTL_A contains transmit and receive control bits for SPORT half 'A', including serial port mode selection for the half SPORT's primary and secondary channels. The function of some bits in SPORT_CTL_A vary, depending on the SPORT's operating mode. For more information, see the SPORT operating modes description. If reading reserved bits, the read value is the last written value to these bits or is the reset value of these bits.

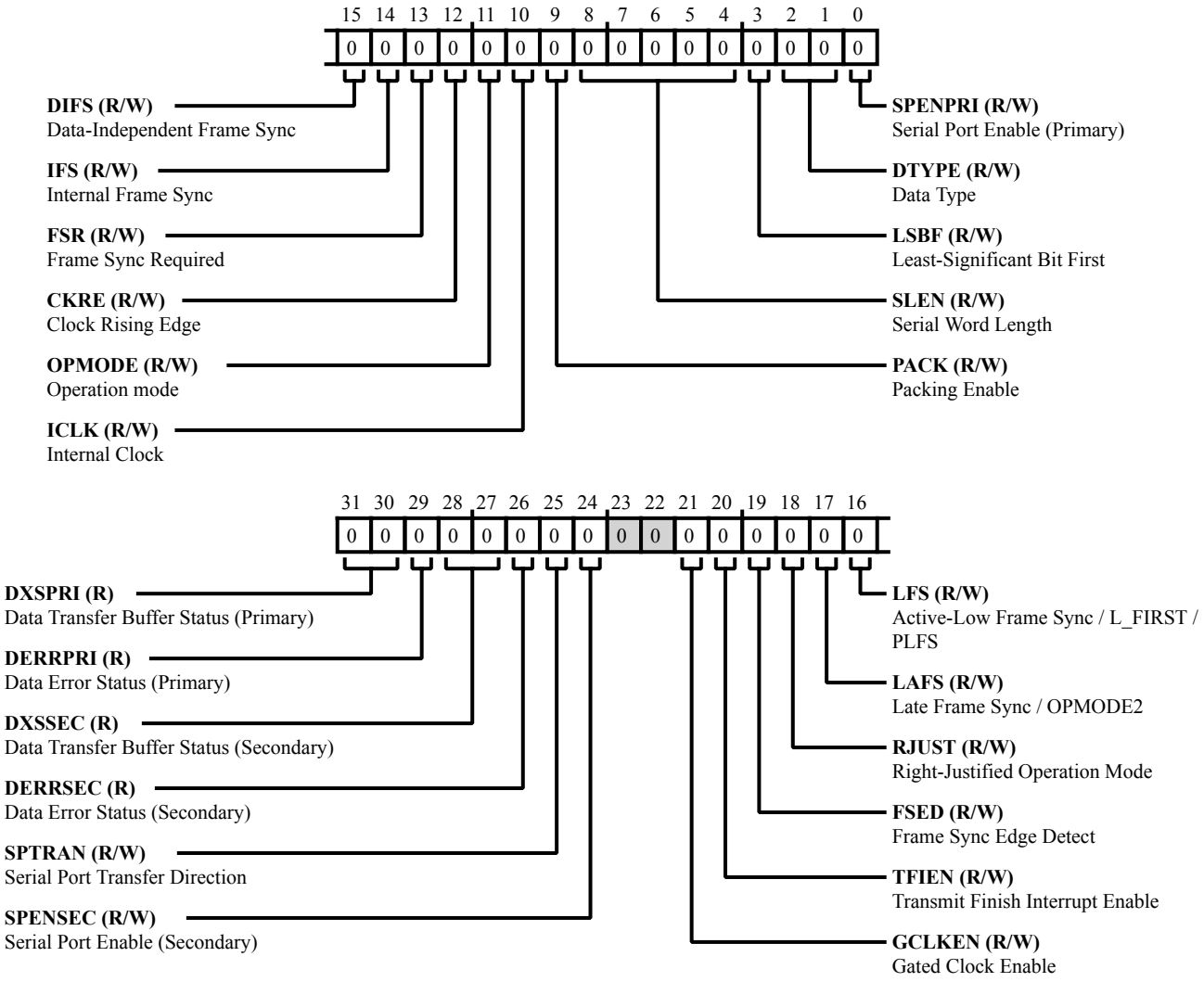

Figure 25-25: SPORT_CTL_A Register Diagram

Table 25-22: SPORT_CTL_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:30 (R/NW)	DXSPRI	Data Transfer Buffer Status (Primary). The SPORT_CTL_A.DXSPRI indicates the status of the half SPORT's primary channel data buffer.	
		0	Empty
		1	Reserved
		2	Partially full
		3	Full
29 (R/NW)	DERRPRI	Data Error Status (Primary). The SPORT_CTL_A.DERRPRI reports the half SPORT's primary channel transmit underflow status or receive overflow status, depending on the SPORT transfer direction. If the SPORT_CTL_A.FSR bit =1, SPORT_CTL_A.DERRPRI indicates whether the SPORT_AFS signal (from an internal or external source) occurred while the SPORT_TXPRI_A data buffer was empty (during transmit) or the SPORT_RXPRI_A data buffer was full (during receive). The SPORT transmits or receives data whenever it detects the SPORT_AFS signal. It is important to note that, as a receiver, the SPORT_CTL_A.DERRPRI indicates when the channel has received new data while the SPORT_RXPRI_A receive buffer is full. This new data overwrites existing data. If the SPORT_CTL_A.FSR bit =0, SPORT_CTL_A.DERRPRI is set whenever the SPORT is required to transmit while the SPORT_TXPRI_A transmit buffer is empty and is set whenever the SPORT is required to receive while the SPORT_RXPRI_A receive buffer is full. The SPORT clears the SPORT_CTL_A.DERRPRI bit if the SPORT_ERR_A.DERRPSTAT bit is cleared.	
		0	No error
		1	Error (Tx underflow or Rx overflow)
28:27 (R/NW)	DXSSEC	Data Transfer Buffer Status (Secondary). The SPORT_CTL_A.DXSSEC indicates the status of the half SPORT's secondary channel data buffer.	
		0	Empty
		1	Reserved
		2	Partially full
		3	Full

Table 25-22: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/NW)	DERRSEC	Data Error Status (Secondary). The SPORT_CTL_A.DERRSEC reports the half SPORT's secondary channel transmit underflow status or receive overflow status, depending on the SPORT transfer direction. If the SPORT_CTL_A.FSR bit =1, SPORT_CTL_A.DERRSEC indicates whether the SPORT_AFS signal (from an internal or external source) occurred while the SPORT_TXSEC_A data buffer was empty (during transmit) or the SPORT_RXSEC_A data buffer was full (during receive). The SPORT transmits or receives data whenever it detects the SPORT_AFS signal. It is important to note that, as a receiver, the SPORT_CTL_A.DERRSEC indicates when the channel has received new data while the SPORT_RXSEC_A receive buffer is full. This new data overwrites existing data. If the SPORT_CTL_A.FSR bit =0, SPORT_CTL_A.DERRSEC is set whenever the SPORT is required to transmit while the SPORT_TXSEC_A transmit buffer is empty and is set whenever the SPORT is required to receive while the SPORT_RXSEC_A receive buffer is full. The SPORT clears the SPORT_CTL_A.DERRSEC bit if the SPORT_ERR_A.DERRSSTAT bit is cleared.	
		0	No error
		1	Error (Tx underflow or Rx overflow)
25 (R/W)	SPTRAN	Serial Port Transfer Direction. The SPORT_CTL_A.SPTRAN bit selects the transfer direction (receive or transmit) for the half SPORT's primary and secondary channels. When the direction is receive, the half SPORT activates the receive buffers, and the SPORT_ACLK and SPORT_AFS pins control the receive buffers. The transmit buffers are inactive when the half SPORT's transfer direction is receive. When the direction is transmit, the half SPORT activates the transmit buffers, and the SPORT_ACLK and SPORT_AFS pins control the transmit shift registers. The receive buffers are inactive when the half SPORT's transfer direction is transmit.	
		0	Receive
		1	Transmit
24 (R/W)	SPENSEC	Serial Port Enable (Secondary). The SPORT_CTL_A.SPENSEC bit enables the half SPORT's secondary channel. When this bit is cleared (changes from =1 to =0), the half SPORT automatically flushes the channel's data buffers.	
		0	Disable
		1	Enable

Table 25-22: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W)	GCLKEN	Gated Clock Enable. The SPORT_CTL_A.GCLKEN bit enables gated clock operation for the half SPORT when in DSP serial mode or left-justified stereo modes (SPORT_CTL_A.OPMODE = 0 or 1). This bit is ignored when the half SPORT is in right-justified mode (SPORT_CTL_A.RJUST =1) or multi-channel mode (SPORT_MCTL_A.MCE =1). When SPORT_CTL_A.GCLKEN is enabled, the SPORT clock is active when the SPORT is transferring data or when the frame sync changes (transitions to active state).	
		0	Disable
		1	Enable
20 (R/W)	TFIEN	Transmit Finish Interrupt Enable. The SPORT_CTL_A.TFIEN bit selects when the half SPORT issues its transmission complete interrupt if a DMA complete interrupt is enabled by the DMA_CFG.INT configuration. When enabled (SPORT_CTL_A.TFIEN =1), the DMA complete peripheral interrupt is generated when the last bit of last word in the DMA is shifted out. When disabled (SPORT_CTL_A.TFIEN =0), the DMA interrupt is generated when the DMA counter expires (the last word goes to the transmit buffer).	
		0	Last word sent (DMA count done) interrupt
		1	Last bit sent (Tx buffer done) interrupt
19 (R/W)	FSED	Frame Sync Edge Detect. The SPORT_CTL_A.FSED bit enables the half SPORT to start transmitting or receiving after detecting an active edge of an external frame sync. The SPORT_CTL_A.FSED may be enabled even during an active frame sync, and the half SPORT starts the transfer on the next valid rising or falling edge of external frame sync. If disabled (SPORT_CTL_A.FSED =0), the half SPORT operates in the standard level-sensitive detection mode for external frame sync.	
		0	Level detect frame sync
		1	Edge detect frame sync

Table 25-22: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	RJUST	Right-Justified Operation Mode. The SPORT_CTL_A.RJUST bit enables the half SPORT (if SPORT_CTL_A.OPMODE =1) to transfer data in right-justified operation mode. In this mode, the half SPORT aligns data to the end of the frame sync, rather than the start of the frame sync. When using right-justified mode, systems should program an appropriate delay count to introduce a clock delay before the half SPORT state machine starts to capture data. This value is set in the DCNT field (right-justified mode usage of the SPORT_MCTL_A.WOFFSET field). For information about appropriate delay selections, see the SPORT operating modes section.	
		0	Disable
		1	Enable
17 (R/W)	LAFS	Late Frame Sync / OPMODE2. When the half SPORT is in DSP standard mode (SPORT_CTL_A.OPMODE =0) or in right-justified mode (SPORT_CTL_A.RJUST =1), the SPORT_CTL_A.LAFS bit selects whether the half SPORT generates a late frame sync (SPORT_AFS during first data bit) or generates an early frame sync signal (SPORT_AFS during serial clock cycle before first data bit). When the half SPORT is in I2S / left-justified mode (SPORT_CTL_A.OPMODE =1), the SPORT_CTL_A.LAFS bit acts as OPMODE2, selecting whether the half SPORT is in left-justified mode or I2S mode. When the half SPORT is in multi-channel mode (SPORT_MCTL_A.MCE =1), the SPORT_CTL_A.LAFS bit is reserved.	
		0	Early frame sync (or I2S mode)
		1	Late frame sync (or left-justified mode)
16 (R/W)	LFS	Active-Low Frame Sync / L_FIRST / PLFS. When the half SPORT is in DSP standard mode and multichannel mode (SPORT_CTL_A.OPMODE =0), the SPORT_CTL_A.LFS bit selects whether the half SPORT uses active low or active high frame sync. When the half SPORT is in I2S / packed / left-justified mode (SPORT_CTL_A.OPMODE =1), the SPORT_CTL_A.LFS bit acts as L_FIRST, selecting whether the half SPORT transfers data first for the left or right channel.	
		0	Active high frame sync (DSP standard mode) or rising edge frame sync (multi-channel mode) or right channel first (I2S/packed mode) or left channel first (left-justified mode)
		1	Active low frame sync (DSP standard mode) or falling edge frame sync (multi-channel mode) or left channel first (I2S/packed mode) or right channel first (left-justified mode)

Table 25-22: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	DIFS	Data-Independent Frame Sync. The SPORT_CTL_A.DIFS bit selects whether the half SPORT uses a data-independent or data-dependent frame sync. When using a data-independent frame sync, the half SPORT generates the sync at the interval selected by SPORT_DIV_A.FSDIV. When using a data-dependent frame sync, the half SPORT generates the sync on the selected interval when the transmit buffer is not empty or when the receive buffer is not full. Note that the SPORT_CTL_A.DIFS bit is automatically set when the half SPORT is in packed or multichannel modes.	
		0	Data-dependent frame sync
		1	Data-independent frame sync
14 (R/W)	IFS	Internal Frame Sync. The SPORT_CTL_A.IFS bit selects whether the half SPORT uses an internal frame sync or uses an external frame sync. Note that the externally-generated frame sync does not need to be synchronous with the processor's system clock.	
		0	External frame sync
		1	Internal frame sync
13 (R/W)	FSR	Frame Sync Required. The SPORT_CTL_A.FSR selects whether or not the half SPORT requires frame sync for data transfer. This bit is automatically set when the half SPORT is in I2S / packed / left-justified mode (SPORT_CTL_A.OPMODE =1) or is in multi-channel mode (SPORT_MCTL_A.MCE =1).	
		0	No frame sync required
		1	Frame sync required
12 (R/W)	CKRE	Clock Rising Edge. The SPORT_CTL_A.CKRE selects the rising or falling edge of the SPORT_ACLK clock for the half SPORT to sample receive data and frame sync. Note that the half SPORT changes the state of transmit data and frame sync signals on the non-selected edge of the SPORT_ACLK. Also note that the transmit and receive related SPORT halves (A and B) should be programmed with the same value for SPORT_CTL_A.CKRE. This programming drives the internally-generated signals on one edge of SPORT_ACLK and samples the received signals on the opposite edge.	
		0	Clock falling edge
		1	Clock rising edge

Table 25-22: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	OPMODE	Operation mode. The SPORT_CTL_A.OPMODE bit selects whether the half SPORT operates in DSP standard / multi-channel mode or operates in I2S / packed / left-justified mode. The mode selection affects the operation of the SPORT_CTL_A.LAFS and SPORT_CTL_A.LFS bits. Also, the SPORT_CTL_A.OPMODE bit enables or disables operation of the SPORT_CTL_A.GCLKEN, SPORT_CTL_A.FSED, SPORT_CTL_A.RJUST, SPORT_CTL_A.DIFS, SPORT_CTL_A.FSR, and SPORT_CTL_A.CKRE bits.	
		0	DSP standard/multi-channel mode
		1	I2S/packed/left-justified mode
10 (R/W)	ICLK	Internal Clock. When the half SPORT is in DSP standard mode (SPORT_CTL_A.OPMODE =0), the SPORT_CTL_A.ICLK bit selects whether the half SPORT uses an internal or external clock. For internal clock enabled, the half SPORT generates the SPORT_ACLK clock signal, and the SPORT_ACLK is an output. The SPORT_DIV_A.CLKDIV serial clock divisor value determines the clock frequency. For internal clock disabled, the SPORT_ACLK clock signal is an input, and the serial clock divisor is ignored. Note that the externally-generated serial clock does not need to be synchronous with the processor's system clock.	
		0	External clock
		1	Internal clock
9 (R/W)	PACK	Packing Enable. The SPORT_CTL_A.PACK bit enables the half SPORT to perform 16- to 32-bit packing on received data and to perform 32- to 16-bit unpacking on transmitted data. The receive packing operation packs two successive received words into a single 32-bit word. The transmit unpacking operation unpacks each 32-bit word and transmits it as two 16-bit words. The first 16-bit (or smaller) word is right-justified in bits 15-0 of the packed word, and the second 16-bit (or smaller) word is right-justified in bits 31-16. This format applies to both receive (packing) and transmit (unpacking) operations. Companding may be used with word packing or unpacking. The half SPORT generates data transfer related interrupts when packing is enabled. The transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.	
		0	Disable
		1	Enable
8:4 (R/W)	SLEN	Serial Word Length. The SPORT_CTL_A.SLEN bits selects word length in bits for the half SPORT's data transfers. Word may be from 4- to 32-bits in length. The formula for selecting the word length in bits is: SPORT_CTL_A.SLEN = (serial word length in bits) - 1	

Table 25-22: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		For DSP standard mode (SPORT_CTL_A.OPMODE =0), use SPORT_CTL_A.SLEN of 3 to 31 bits. For I2S / packed / left-justified mode (SPORT_CTL_A.OPMODE =1), use SPORT_CTL_A.SLEN of 4 to 31 bits.	
3 (R/W)	LSBF	Least-Significant Bit First. The SPORT_CTL_A.LSBF bit selects whether the half SPORT transmits or receives data LSB first or MSB first.	
		0	MSB first sent/received (big endian)
		1	LSB first sent/received (little endian)
2:1 (R/W)	DTYPE	Data Type. The SPORT_CTL_A.DTYPE bits selects the data type formatting for the half SPORT's data transfers in DSP standard mode (SPORT_CTL_A.OPMODE =0).	
		0	Right-justify data, zero-fill unused MSBs
		1	Right-justify data, sign-extend unused MSBs
		2	-law compand data
		3	A-law compand data
0 (R/W)	SPENPRI	Serial Port Enable (Primary). The SPORT_CTL_A.SPENPRI bit enables the half SPORT's primary channel. When this bit is cleared (changes from =1 to =0), the half SPORT automatically flushes the channel's data buffers.	
		0	Disable
		1	Enable

Half SPORT 'B' Control Register

The SPORT_CTL_B contains transmit and receive control bits for SPORT half 'B', including serial port mode selection for the half SPORT's primary and secondary channels. The function of some bits in SPORT_CTL_B vary, depending on the SPORT's operating mode. For more information, see the SPORT operating modes description. If reading reserved bits, the read value is the last written value to these bits or is the reset value of these bits.

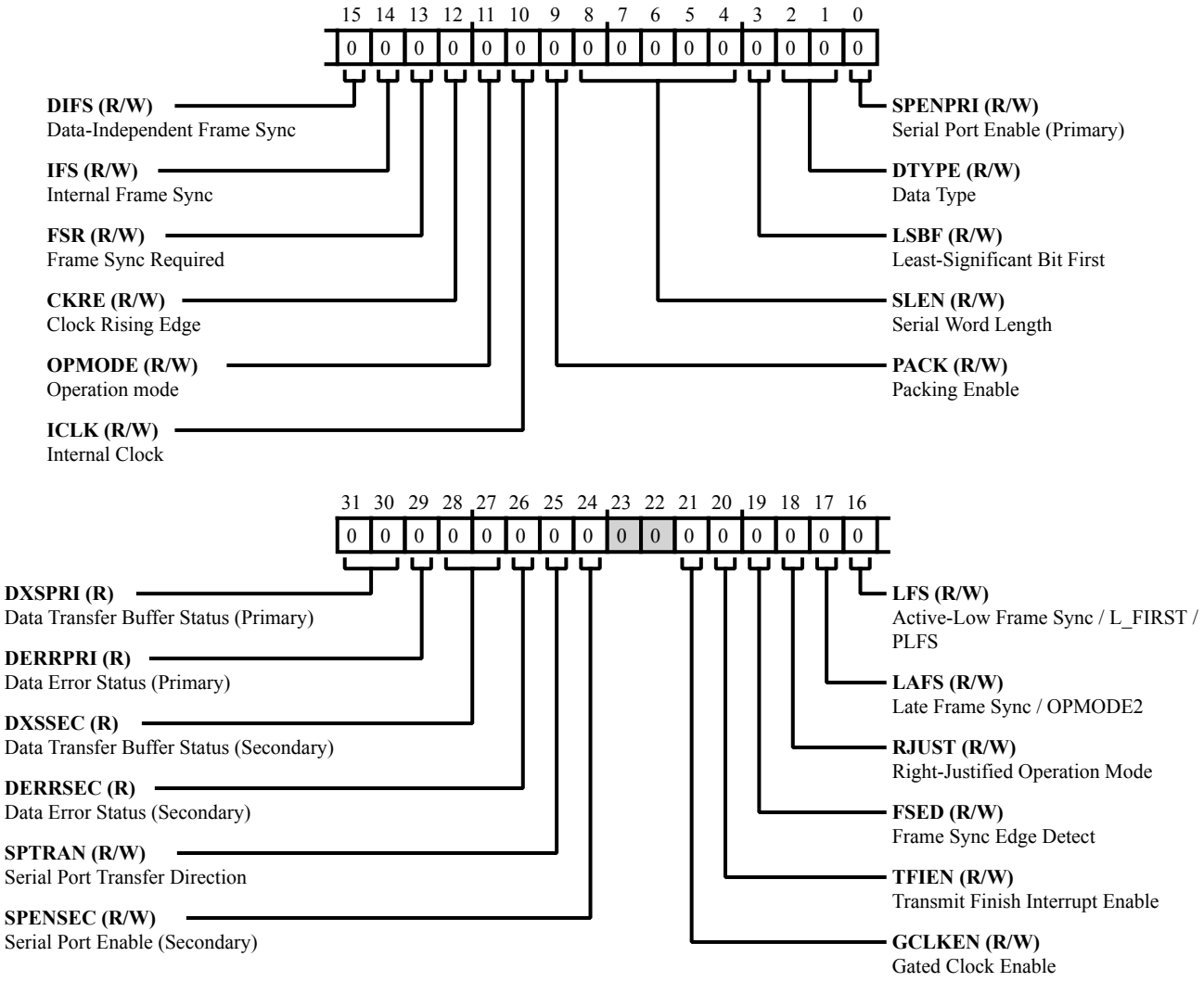

Figure 25-26: SPORT_CTL_B Register Diagram

Table 25-23: SPORT_CTL_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:30 (R/NW)	DXSPRI	Data Transfer Buffer Status (Primary). The SPORT_CTL_B.DXSPRI indicates the status of the half SPORT's primary channel data buffer.	
		0	Empty
		1	Reserved
		2	Partially full
		3	Full
29 (R/NW)	DERRPRI	Data Error Status (Primary). The SPORT_CTL_B.DERRPRI reports the half SPORT's primary channel transmit underflow status or receive overflow status, depending on the SPORT transfer direction. If the SPORT_CTL_B.FSR bit =1, SPORT_CTL_B.DERRPRI indicates whether the SPORT_BFS signal (from an internal or external source) occurred while the SPORT_TXPRI_B data buffer was empty (during transmit) or the SPORT_RXPRI_B data buffer was full (during receive). The SPORT transmits or receives data whenever it detects the SPORT_BFS signal. It is important to note that, as a receiver, the SPORT_CTL_B.DERRPRI indicates when the channel has received new data while the SPORT_RXPRI_B receive buffer is full. This new data overwrites existing data. If the SPORT_CTL_B.FSR bit =0, SPORT_CTL_B.DERRPRI is set whenever the SPORT is required to transmit while the SPORT_TXPRI_B transmit buffer is empty and is set whenever the SPORT is required to receive while the SPORT_RXPRI_B receive buffer is full. The SPORT clears the SPORT_CTL_B.DERRPRI bit if the SPORT_ERR_B.DERRPSTAT bit is cleared.	
		0	No error
		1	Error (Tx underflow or Rx overflow)
28:27 (R/NW)	DXSSEC	Data Transfer Buffer Status (Secondary). The SPORT_CTL_B.DXSSEC indicates the status of the half SPORT's secondary channel data buffer.	
		0	Empty
		1	Reserved
		2	Partially full
		3	Full

Table 25-23: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/NW)	DERRSEC	Data Error Status (Secondary). The SPORT_CTL_B.DERRSEC reports the half SPORT's secondary channel transmit underflow status or receive overflow status, depending on the SPORT transfer direction. If the SPORT_CTL_B.FSR bit =1, SPORT_CTL_B.DERRSEC indicates whether the SPORT_BFS signal (from an internal or external source) occurred while the SPORT_TXSEC_B data buffer was empty (during transmit) or the SPORT_RXSEC_B data buffer was full (during receive). The SPORT transmits or receives data whenever it detects the SPORT_BFS signal. It is important to note that, as a receiver, the SPORT_CTL_B.DERRSEC indicates when the channel has received new data while the SPORT_RXSEC_B receive buffer is full. This new data overwrites existing data. If the SPORT_CTL_B.FSR bit =0, SPORT_CTL_B.DERRSEC is set whenever the SPORT is required to transmit while the SPORT_TXSEC_B transmit buffer is empty and is set whenever the SPORT is required to receive while the SPORT_RXSEC_B receive buffer is full. The SPORT clears the SPORT_CTL_B.DERRSEC bit if the SPORT_ERR_B.DERRSTAT bit is cleared.	
		0	No error
		1	Error (Tx underflow or Rx overflow)
25 (R/W)	SPTRAN	Serial Port Transfer Direction. The SPORT_CTL_B.SPTRAN bit selects the transfer direction (receive or transmit) for the half SPORT's primary and secondary channels. When the direction is receive, the half SPORT activates the receive buffers, and the SPORT_BCLK and SPORT_BFS pins control the receive buffers. The transmit buffers are inactive when the half SPORT's transfer direction is receive. When the direction is transmit, the half SPORT activates the transmit buffers, and the SPORT_BCLK and SPORT_BFS pins control the transmit shift registers. The receive buffers are inactive when the half SPORT's transfer direction is transmit.	
		0	Receive
		1	Transmit
24 (R/W)	SPENSEC	Serial Port Enable (Secondary). The SPORT_CTL_B.SPENSEC bit enables the half SPORT's secondary channel. When this bit is cleared (changes from =1 to =0), the half SPORT automatically flushes the channel's data buffers.	
		0	Disable
		1	Enable

Table 25-23: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W)	GCLKEN	Gated Clock Enable. The SPORT_CTL_B.GCLKEN bit enables gated clock operation for the half SPORT when in DSP serial mode or left-justified stereo modes (SPORT_CTL_B.OPMODE = 0 or 1). This bit is ignored when the half SPORT is in right-justified mode (SPORT_CTL_B.RJUST =1) or multi-channel mode (SPORT_MCTL_B.MCE =1). When SPORT_CTL_B.GCLKEN is enabled, the SPORT clock is active when the SPORT is transferring data or when the frame sync changes (transitions to active state).	
		0	Disable
		1	Enable
20 (R/W)	TFIEN	Transmit Finish Interrupt Enable. The SPORT_CTL_B.TFIEN bit selects when the half SPORT issues its transmission complete interrupt if a DMA complete interrupt is enabled by the DMA_CFG.INT configuration. When enabled (SPORT_CTL_B.TFIEN =1), the DMA complete peripheral interrupt is generated when the last bit of last word in the DMA is shifted out. When disabled (SPORT_CTL_B.TFIEN =0), the DMA interrupt is generated when the DMA counter expires (the last word goes to the transmit buffer).	
		0	Last word sent (DMA count done) interrupt
		1	Last bit sent (Tx buffer done) interrupt
19 (R/W)	FSED	Frame Sync Edge Detect. The SPORT_CTL_B.FSED bit enables the half SPORT to start transmitting or receiving after detecting an active edge of an external frame sync. The SPORT_CTL_B.FSED may be enabled even during an active frame sync, and the half SPORT starts the transfer on the next valid rising or falling edge of external frame sync. If disabled (SPORT_CTL_B.FSED =0), the half SPORT operates in the standard level-sensitive detection mode for external frame sync.	
		0	Level detect frame sync
		1	Edge detect frame sync

Table 25-23: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	RJUST	Right-Justified Operation Mode. The SPORT_CTL_B.RJUST bit enables the half SPORT (if SPORT_CTL_B.OPMODE =1) to transfer data in right-justified operation mode. In this mode, the half SPORT aligns data to the end of the frame sync, rather than the start of the frame sync. When using right-justified mode, systems should program an appropriate delay count to introduce a clock delay before the half SPORT state machine starts to capture data. This value is set in the DCNT field (right-justified mode usage of the SPORT_MCTL_B.WOFFSET field). For information about appropriate delay selections, see the SPORT operating modes section.	
		0	Disable
		1	Enable
17 (R/W)	LAFS	Late Frame Sync / OPMODE2. When the half SPORT is in DSP standard mode (SPORT_CTL_B.OPMODE =0) or in right-justified mode (SPORT_CTL_B.RJUST =1), the SPORT_CTL_B.LAFS bit selects whether the half SPORT generates a late frame sync (SPORT_BFS during first data bit) or generates an early frame sync signal (SPORT_BFS during serial clock cycle before first data bit). When the half SPORT is in I2S / left-justified mode (SPORT_CTL_B.OPMODE =1), the SPORT_CTL_B.LAFS bit acts as OPMODE2, selecting whether the half SPORT is in left-justified mode or I2S mode. When the half SPORT is in multi-channel mode (SPORT_MCTL_B.MCE =1), the SPORT_CTL_B.LAFS bit is reserved.	
		0	Early frame sync (or I2S mode)
		1	Late frame sync (or left-justified mode)

Table 25-23: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	LFS	Active-Low Frame Sync / L_FIRST / PLFS. When the half SPORT is in DSP standard mode and multichannel mode (SPORT_CTL_B.OPMODE =0), the SPORT_CTL_B.LFS bit selects whether the half SPORT uses active low or active high frame sync. When the half SPORT is in I2S / packed / left-justified mode (SPORT_CTL_B.OPMODE =1), the SPORT_CTL_B.LFS bit acts as L_FIRST, selecting whether the half SPORT transfers data first for the left or right channel.	
		0	Active high frame sync (DSP standard mode) or rising edge frame sync (multi-channel mode) or right channel first (I2S/packed mode) or left channel first (left-justified mode)
		1	Active low frame sync (DSP standard mode) or falling edge frame sync (multi-channel mode) or left channel first (I2S/packed mode) or right channel first (left-justified mode)
15 (R/W)	DIFS	Data-Independent Frame Sync. The SPORT_CTL_B.DIFS bit selects whether the half SPORT uses a data-independent or data-dependent frame sync. When using a data-independent frame sync, the half SPORT generates the sync at the interval selected by SPORT_DIV_B.FSDIV. When using a data-dependent frame sync, the half SPORT generates the sync on the selected interval when the transmit buffer is not empty or when the receive buffer is not full. Note that the SPORT_CTL_B.DIFS bit is automatically set when the half SPORT is in packed or multichannel modes.	
		0	Data-dependent frame sync
		1	Data-independent frame sync
14 (R/W)	IFS	Internal Frame Sync. The SPORT_CTL_B.IFS bit selects whether the half SPORT uses an internal frame sync or uses an external frame sync. Note that the externally-generated frame sync does not need to be synchronous with the processor's system clock.	
		0	External frame sync
		1	Internal frame sync

Table 25-23: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	FSR	Frame Sync Required. The SPORT_CTL_B.FSR selects whether or not the half SPORT requires frame sync for data transfer. This bit is automatically set when the half SPORT is in I2S / packed / left-justified mode (SPORT_CTL_B.OPMODE =1) or is in multi-channel mode (SPORT_MCTL_B.MCE =1).	
		0	No frame sync required
		1	Frame sync required
12 (R/W)	CKRE	Clock Rising Edge. The SPORT_CTL_B.CKRE selects the rising or falling edge of the SPORT_BCLK clock for the half SPORT to sample receive data and frame sync. Note that the half SPORT changes the state of transmit data and frame sync signals on the non-selected edge of the SPORT_BCLK. Also note that the transmit and receive related SPORT halves (A and B) should be programmed with the same value for SPORT_CTL_B.CKRE. This programming drives the internally-generated signals on one edge of SPORT_BCLK and samples the received signals on the opposite edge.	
		0	Clock falling edge
		1	Clock rising edge
11 (R/W)	OPMODE	Operation mode. The SPORT_CTL_B.OPMODE bit selects whether the half SPORT operates in DSP standard / multi-channel mode or operates in I2S / packed / left-justified mode. The mode selection affects the operation of the SPORT_CTL_B.LAFS and SPORT_CTL_B.LFS bits. Also, the SPORT_CTL_B.OPMODE bit enables or disables operation of the SPORT_CTL_B.GCLKEN, SPORT_CTL_B.FSED, SPORT_CTL_B.RJUST, SPORT_CTL_B.DIFS, SPORT_CTL_B.FSR, and SPORT_CTL_B.CKRE bits.	
		0	DSP standard/multi-channel mode
		1	I2S/packed/left-justified mode
10 (R/W)	ICLK	Internal Clock. When the half SPORT is in DSP standard mode (SPORT_CTL_B.OPMODE =0), the SPORT_CTL_B.ICLK bit selects whether the half SPORT uses an internal or external clock. For internal clock enabled, the half SPORT generates the SPORT_BCLK clock signal, and the SPORT_BCLK is an output. The SPORT_DIV_B.CLKDIV serial clock divisor value determines the clock frequency. For internal clock disabled, the SPORT_BCLK clock signal is an input, and the serial clock divisor is ignored. Note that the externally-generated serial clock does not need to be synchronous with the processor's system clock.	
		0	External clock
		1	Internal clock

Table 25-23: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	PACK	Packing Enable. The SPORT_CTL_B.PACK bit enables the half SPORT to perform 16- to 32-bit packing on received data and to perform 32- to 16-bit unpacking on transmitted data. The receive packing operation packs two successive received words into a single 32-bit word. The transmit unpacking operation unpacks each 32-bit word and transmits it as two 16-bit words. The first 16-bit (or smaller) word is right-justified in bits 15-0 of the packed word, and the second 16-bit (or smaller) word is right-justified in bits 31-16. This format applies to both receive (packing) and transmit (unpacking) operations. Companding may be used with word packing or unpacking. The half SPORT generates data transfer related interrupts when packing is enabled. The transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.	
		0	Disable
		1	Enable
8:4 (R/W)	SLEN	Serial Word Length. The SPORT_CTL_B.SLEN bits selects word length in bits for the half SPORT's data transfers. Word may be from 4- to 32-bits in length. The formula for selecting the word length in bits is: SPORT_CTL_B.SLEN = (serial word length in bits) - 1 For DSP standard mode (SPORT_CTL_B.OPMODE =0), use SPORT_CTL_B.SLEN of 3 to 31 bits. For I2S / packed / left-justified mode (SPORT_CTL_B.OPMODE =1), use SPORT_CTL_B.SLEN of 4 to 31 bits.	
3 (R/W)	LSBF	Least-Significant Bit First. The SPORT_CTL_B.LSBF bit selects whether the half SPORT transmits or receives data LSB first or MSB first.	
		0	MSB first sent/received (big endian)
		1	LSB first sent/received (little endian)
2:1 (R/W)	DTYPE	Data Type. The SPORT_CTL_B.DTYPE bits selects the data type formatting for the half SPORT's data transfers in DSP standard mode (SPORT_CTL_B.OPMODE =0).	
		0	Right-justify data, zero-fill unused MSBs
		1	Right-justify data, sign-extend unused MSBs
		2	-law compand data
		3	A-law compand data

Table 25-23: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	SPENPRI	Serial Port Enable (Primary). The `SPORT_CTL_B.SPENPRI` bit enables the half SPORT's primary channel. When this bit is cleared (changes from =1 to =0), the half SPORT automatically flushes the channel's data buffers.	
		0	Disable
		1	Enable

Half SPORT 'A' Divisor Register

The SPORT_DIV_A contains divisor values that determine frequencies of internally-generated clocks and frame syncs for half SPORT 'A'.

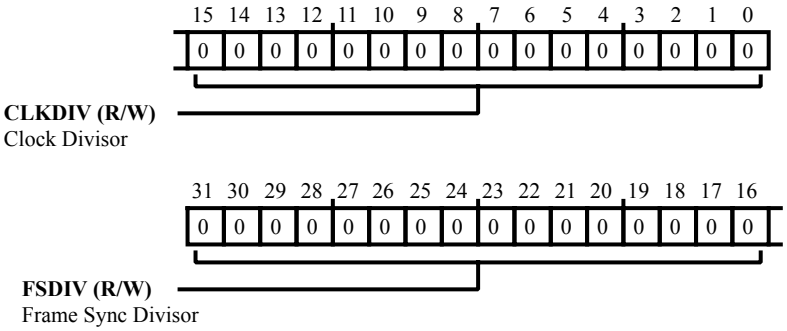

Figure 25-27: SPORT_DIV_A Register Diagram

Table 25-24: SPORT_DIV_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	FSDIV	Frame Sync Divisor. The SPORT_DIV_A.FSDIV bits select the number of transmit or receive clock cycles that the half SPORT counts before generating a frame sync pulse. The half SPORT counts serial clock cycles whether these are from an internally- or an externally-generated serial clock. The formula relating SPORT_DIV_A.FSDIV to the number of cycles between frame sync pulses is: SPORT_DIV_A.FSDIV = (number of serial clocks between frame syncs) - 1 Use the following equation to determine the value of SPORT_DIV_A.FSDIV, given the serial clock frequency and desired frame sync frequency: FSDIV = (SPORT_ACLK / SPORT_AFS) - 1 Note that the frame sync is continuously active when SPORT_DIV_A.FSDIV = 0. The value of SPORT_DIV_A.FSDIV should not be less than the serial word length (SPORT_CTL_A.SLEN), as this may cause an external device to abort the current operation or cause other unpredictable results.
15:0 (R/W)	CLKDIV	Clock Divisor. The SPORT_DIV_A.CLKDIV bits select the divisor that the half SPORT uses to calculate the serial clock (SPORT_ACLK) from the processor system clock (SYSCLK). The divisor is a 16-bit value, allowing a wide range of serial clock rates. When configured for internal clock (SPORT_CTL_A.ICLK =1), legal SPORT_DIV_A.CLKDIV values are 0 to 65535. Given the processor system clock frequency and desired serial clock frequency, use the following formula to calculate the value of SPORT_DIV_A.CLKDIV: CLKDIV = (SYSCLK / SPORT_ACLK) - 1 For the maximum serial clock frequency, see the processor data sheet.

Half SPORT 'B' Divisor Register

The SPORT_DIV_B contains divisor values that determine frequencies of internally-generated clocks and frame syncs for SPORT half 'B'.

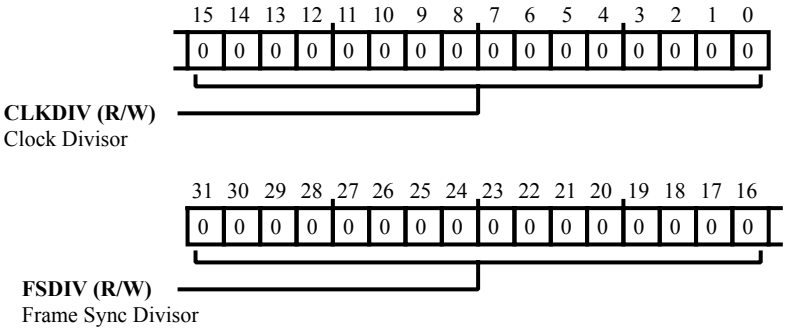

Figure 25-28: SPORT_DIV_B Register Diagram

Table 25-25: SPORT_DIV_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	FSDIV	Frame Sync Divisor. The SPORT_DIV_B.FSDIV bits select the number of transmit or receive clock cycles that the half SPORT counts before generating a frame sync pulse. The half SPORT counts serial clock cycles whether these are from an internally- or an externally-generated serial clock. The formula relating SPORT_DIV_B.FSDIV to the number of cycles between frame sync pulses is: SPORT_DIV_B.FSDIV = (number of serial clocks between frame syncs) - 1 Use the following equation to determine the value of SPORT_DIV_B.FSDIV, given the serial clock frequency and desired frame sync frequency: FSDIV = (SPORT_BCLK / SPORT_BFS) - 1 Note that the frame sync is continuously active when SPORT_DIV_B.FSDIV = 0. The value of SPORT_DIV_B.FSDIV should not be less than the serial word length (SPORT_CTL_B.SLEN), as this may cause an external device to abort the current operation or cause other unpredictable results.
15:0 (R/W)	CLKDIV	Clock Divisor. The SPORT_DIV_B.CLKDIV bits select the divisor that the half SPORT uses to calculate the serial clock (SPORT_BCLK) from the processor system clock (SYSCLK). The divisor is a 16-bit value, allowing a wide range of serial clock rates. When configured for internal clock (SPORT_CTL_B.ICLK =1), legal SPORT_DIV_B.CLKDIV values are 0 to 65535. Given the processor system clock frequency and desired serial clock frequency, use the following formula to calculate the value of SPORT_DIV_B.CLKDIV: CLKDIV = (SYSCLK SPORT_BCLK) - 1 For the maximum serial clock frequency, see the processor data sheet.

Half SPORT 'A' Error Register

The SPORT_ERR_A contains error status and error interrupt mask bits for SPORT half 'A', including error handling bits for the half SPORT's primary and secondary channels and frame sync. Detected errors are frame sync violations or buffer over/underflow conditions.

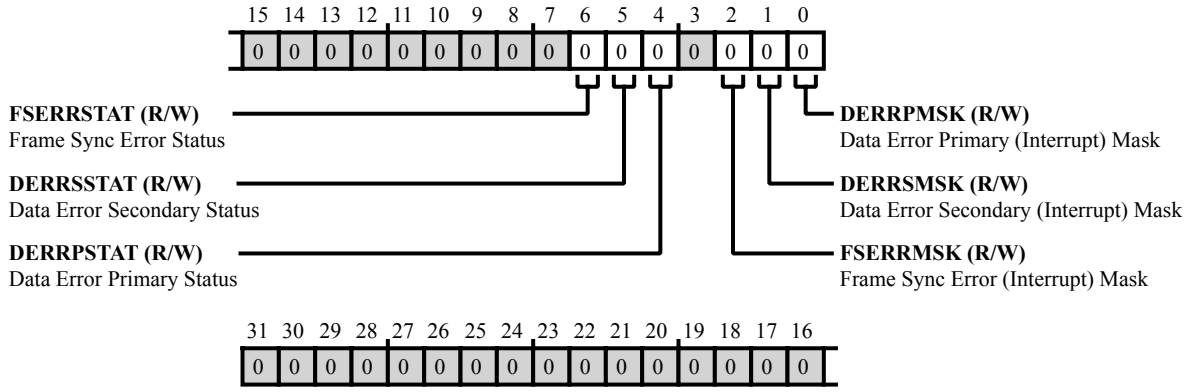

Figure 25-29: SPORT_ERR_A Register Diagram

Table 25-26: SPORT_ERR_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	FSERRSTAT	Frame Sync Error Status. The SPORT_ERR_A.FSERRSTAT bit indicates that the half SPORT has detected a frame sync when the bit count (bits remaining in the frame) is non-zero. When a half SPORT is receiving or transmitting, its bit count is set to a word length (for example, SPORT_CTL_A.SLEN = 32). After each serial clock edge, the half SPORT decrements the transfer's bit count. After the word is received or transmitted, the transfer's bit count reaches zero, and the half SPORT resets it (for example, to 32) on next frame sync. Normal SPORT data transfers always have a non-zero bit count value when active transmission or reception is occurring. Normal SPORT frame syncs occur during a zero bit count.	
		0	No error
		1	Error (non-zero bit count at frame sync)

Table 25-26: SPORT_ERR_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	DERRSSTAT	Data Error Secondary Status. The SPORT_ERR_A.DERRSSTAT bit indicates the error status for the half SPORT's secondary channel data buffers. During transmit (SPORT_CTL_A.SPTRAN =1), SPORT_ERR_A.DERRSSTAT indicates transmit underflow status. During receive (SPORT_CTL_A.SPTRAN =0), SPORT_ERR_A.DERRSSTAT indicates receive overflow status. This bit is used to clear the latch of SPORT status interrupt when triggered by secondary Data Error. This bit can also be used to clear the read-only SPORT_CTL_A.DERRSEC status bit.	
		0	No error
		1	Error (transmit underflow or receive overflow)
4 (R/W)	DERRPSTAT	Data Error Primary Status. The SPORT_ERR_A.DERRPSTAT bit indicates the error status for the half SPORT's primary channel data buffers. During transmit (SPORT_CTL_A.SPTRAN =1), SPORT_ERR_A.DERRPSTAT indicates transmit underflow status. During receive (SPORT_CTL_A.SPTRAN =0), SPORT_ERR_A.DERRPSTAT indicates receive overflow status. This bit is used to clear the latch of SPORT status interrupt when triggered by Primary Data Error. This bit can also be used to clear the read-only SPORT_CTL_A.DERRPRI status bit.	
		0	No error
		1	Error (transmit underflow or receive overflow)
2 (R/W)	FSERRMSK	Frame Sync Error (Interrupt) Mask. The SPORT_ERR_A.FSERRMSK unmasks (enables) the half SPORT to generate the frame sync error interrupt.	
		0	Mask (disable)
		1	Unmask (enable)
1 (R/W)	DERRSMSK	Data Error Secondary (Interrupt) Mask. The SPORT_ERR_A.DERRSMSK unmasks (enables) the half SPORT to generate the data error interrupt for the secondary channel.	
		0	Mask (disable)
		1	Unmask (enable)
0 (R/W)	DERRPMSK	Data Error Primary (Interrupt) Mask. The SPORT_ERR_A.DERRPMSK unmasks (enables) the half SPORT to generate the data error interrupt for the primary channel.	
		0	Mask (disable)
		1	Unmask (enable)

Half SPORT 'B' Error Register

The SPORT_ERR_B contains error status and error interrupt mask bits for SPORT half 'B', including error handling bits for the half SPORT's primary and secondary channels and frame sync. Detected errors are frame sync violations or buffer over/underflow conditions.

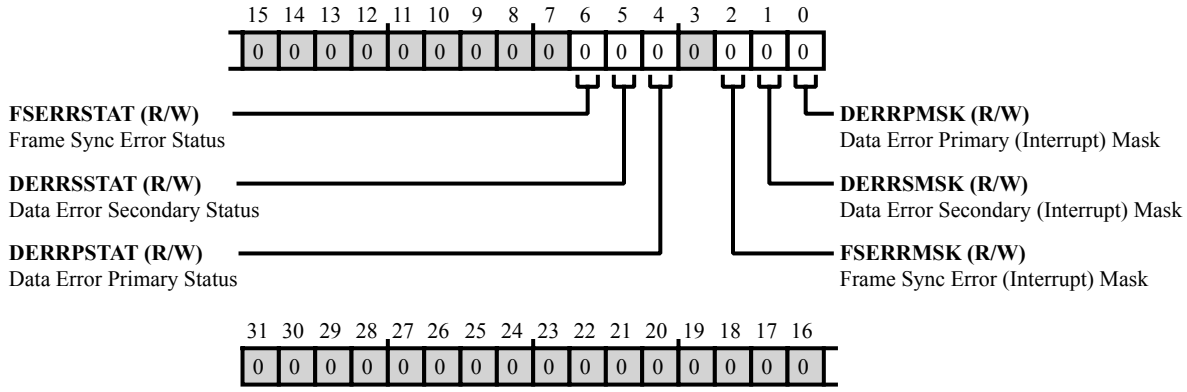

Figure 25-30: SPORT_ERR_B Register Diagram

Table 25-27: SPORT_ERR_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	FSERRSTAT	Frame Sync Error Status. The SPORT_ERR_B.FSERRSTAT bit indicates that the half SPORT has detected a frame sync when the bit count (bits remaining in the frame) is non-zero. When a half SPORT is receiving or transmitting, its bit count is set to a word length (for example, SPORT_CTL_B.SLEN = 32). After each serial clock edge, the half SPORT decrements the transfer's bit count. After the word is received or transmitted, the transfer's bit count reaches zero, and the half SPORT resets it (for example, to 32) on next frame sync. Normal SPORT data transfers always have a non-zero bit count value when active transmission or reception is occurring. Normal SPORT frame syncs occur during a zero bit count.	
		0	No error
		1	Error (non-zero bit count at frame sync)

Table 25-27: SPORT_ERR_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	DERRSSTAT	Data Error Secondary Status. The SPORT_ERR_B.DERRSSTAT bit indicates the error status for the half SPORT's secondary channel data buffers. During transmit (SPORT_CTL_B.SPTRAN =1), SPORT_ERR_B.DERRSSTAT indicates transmit underflow status. During receive (SPORT_CTL_B.SPTRAN =0), SPORT_ERR_B.DERRSSTAT indicates receive overflow status. This bit is used to clear the latch of SPORT status interrupt when triggered by secondary Data Error. This bit can also be used to clear the read-only SPORT_CTL_B.DERRSEC status bit.	
		0	No error
		1	Error (transmit underflow or receive overflow)
4 (R/W)	DERRPSTAT	Data Error Primary Status. The SPORT_ERR_B.DERRPSTAT bit indicates the error status for the half SPORT's primary channel data buffers. During transmit (SPORT_CTL_B.SPTRAN =1), SPORT_ERR_B.DERRPSTAT indicates transmit underflow status. During receive (SPORT_CTL_B.SPTRAN =0), SPORT_ERR_B.DERRPSTAT indicates receive overflow status. This bit is used to clear the latch of SPORT status interrupt when triggered by Primary Data Error. This bit can also be used to clear the read-only SPORT_CTL_B.DERRPRI status bit.	
		0	No error
		1	Error (transmit underflow or receive overflow)
2 (R/W)	FSERRMSK	Frame Sync Error (Interrupt) Mask. The SPORT_ERR_B.FSERRMSK unmasks (enables) the half SPORT to generate the frame sync error interrupt.	
		0	Mask (disable)
		1	Unmask (enable)
1 (R/W)	DERRSMSK	Data Error Secondary (Interrupt) Mask. The SPORT_ERR_B.DERRSMSK unmasks (enables) the half SPORT to generate the data error interrupt for the secondary channel.	
		0	Mask (disable)
		1	Unmask (enable)
0 (R/W)	DERRPMSK	Data Error Primary (Interrupt) Mask. The SPORT_ERR_B.DERRPMSK unmasks (enables) the half SPORT to generate the data error interrupt for the primary channel.	
		0	Mask (disable)
		1	Unmask (enable)

Half SPORT 'A' Multi-channel Control Register

The SPORT_MCTL_A register controls the half SPORT's multi-channel operations. This register enables multi-channel operation, enables multi-channel data packing, selects the multi-channel frame delay, selects the number of multi-channel slots, and selects the multi-channel window offset size.

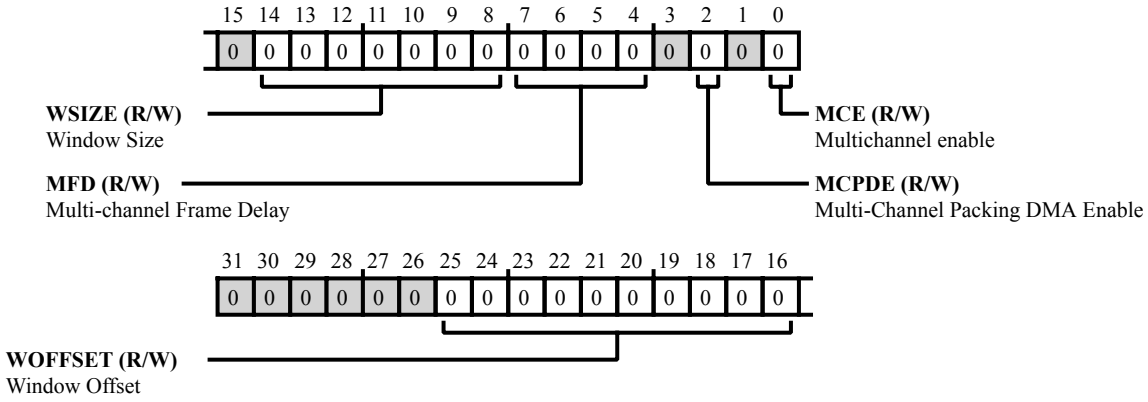

Figure 25-31: SPORT_MCTL_A Register Diagram

Table 25-28: SPORT_MCTL_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:16 (R/W)	WOFFSET	Window Offset. The SPORT_MCTL_A.WOFFSET bits select the start location for the half SPORT's active window of channels within the 1024-channel range. A value of 0 specifies no offset and 896 is the largest value that permits using all 128 channels. When multi-channel mode is disabled (SPORT_MCTL_A.MCE =0) and the right-justified mode is enabled (SPORT_CTL_A.RJUST =1), the least significant 6 bits of SPORT_MCTL_A.WOFFSET serve as the delay count (DCNT) field. These bits introduce a clock delay before the half SPORT state machine starts to capture data. For information about appropriate delay selections, see the SPORT operating modes section.
14:8 (R/W)	WSIZE	Window Size. The SPORT_MCTL_A.WSIZE bits select the window size for the half SPORT's active window of channels. Use the following formula to calculate the window size value: SPORT_MCTL_A.WSIZE = (number of channel slots) -1
7:4 (R/W)	MFD	Multi-channel Frame Delay. The SPORT_MCTL_A.MFD bits select the delay (in serial clock cycles) between the half SPORT's multi-channel frame sync pulse and channel 0. The 4-bit field allows selecting Multichannel Frame delay of 0-15 serial clocks.

Table 25-28: SPORT_MCTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	MCPDE	Multi-Channel Packing DMA Enable. The SPORT_MCTL_A.MCPDE bit enables DMA data packing for transmit and enables DMA data unpacking for the half SPORT's multi-channel data transfers.	
		0	Disable
		1	Enable
0 (R/W)	MCE	Multichannel enable. The SPORT_MCTL_A.MCE bit enables multi-channel operations for the half SPORT. The Half SPORT is configured in normal multichannel mode if SPORT_CTL_A.OPMODE=0; while it is configured in Packed mode if SPORT_CTL_A.OPMODE=1. When Configuring in these modes, the Multichannel Enable bit (SPORT_MCTL_A.MCE) should be set before enabling SPORT data channel enable bits (SPORT_CTL_A.SPENPRI and/or SPORT_CTL_A.SPENSEC). When these channel bits transition from 1 to 0, note that the half SPORT's data transfer buffers are cleared, and the SPORT_CTL_A.DERRPRI and SPORT_CTL_A.DERRSEC bits are cleared.	
		0	Disable
		1	Enable

Half SPORT 'B' Multi-channel Control Register

The SPORT_MCTL_B register controls the half SPORT's multi-channel operations. This register enables multi-channel operation, enables multi-channel data packing, selects the multi-channel frame delay, selects the number of multi-channel slots, and selects the multi-channel window offset size.

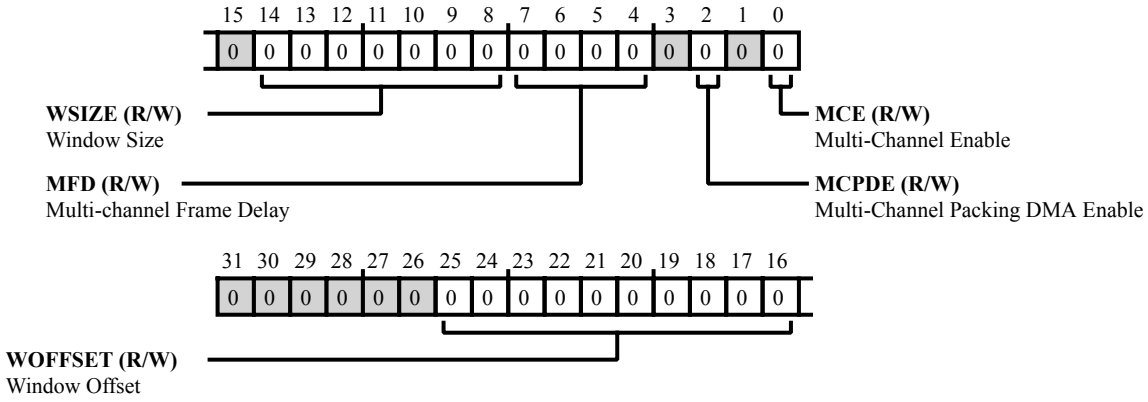

Figure 25-32: SPORT_MCTL_B Register Diagram

Table 25-29: SPORT_MCTL_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:16 (R/W)	WOFFSET	Window Offset. The SPORT_MCTL_B.WOFFSET bits select the start location for the half SPORT's active window of channels within the 1024-channel range. A value of 0 specifies no offset and 896 is the largest value that permits using all 128 channels. When multi-channel mode is disabled (SPORT_MCTL_B.MCE =0) and right-justified mode is enabled (SPORT_CTL_B.RJUST =1), the least significant 6 bits of SPORT_MCTL_B.WOFFSET serve as the delay count (DCNT) field. These bits introduce a clock delay before the half SPORT state machine starts to capture data. For information about appropriate delay selections, see the SPORT operating modes section.
14:8 (R/W)	WSIZE	Window Size. The SPORT_MCTL_B.WSIZE bits select the window size for the half SPORT's active window of channels. Use the following formula to calculate the window size value: SPORT_MCTL_B.WSIZE = (number of channel slots) -1
7:4 (R/W)	MFD	Multi-channel Frame Delay. The SPORT_MCTL_B.MFD bits select the delay (in serial clock cycles) between the half SPORT's multi-channel frame sync pulse and channel 0. The 4-bit field allows selecting Multichannel Frame delay of 0-15 serial clocks.

Table 25-29: SPORT_MCTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	MCPDE	Multi-Channel Packing DMA Enable. The SPORT_MCTL_B.MCPDE bit enables DMA data packing for transmit and enables DMA data unpacking for the half SPORT's multi-channel data transfers.	
		0	Disable
		1	Enable
0 (R/W)	MCE	Multi-Channel Enable. The SPORT_MCTL_B.MCE bit enables multi-channel operations for the half SPORT. The Half SPORT is configured in normal multichannel mode if SPORT_CTL_B.OPMODE=0; while it is configured in Packed mode if SPORT_CTL_B.OPMODE=1. When Configuring in these modes, the Multichannel Enable bit (SPORT_MCTL_B.MCE) should be set before enabling SPORT data channel enable bits (SPORT_CTL_B.SPENPRI and/or SPORT_CTL_B.SPENSEC). When these channel bits transition from 1 to 0, note that the half SPORT's data transfer buffers are cleared, and the SPORT_CTL_B.DERRPRI and SPORT_CTL_B.DERRSEC bits are cleared.	
		0	Disable
		1	Enable

Half SPORT 'A' Multi-channel Status Register

The SPORT_MSTAT_A register indicates the current multi-channel being serviced among the half SPORT's active channels in multi-channel mode. The half SPORT increments the value by one in this register as each channel is serviced. The value in the SPORT_MSTAT_A register restarts at 0 at each frame sync.

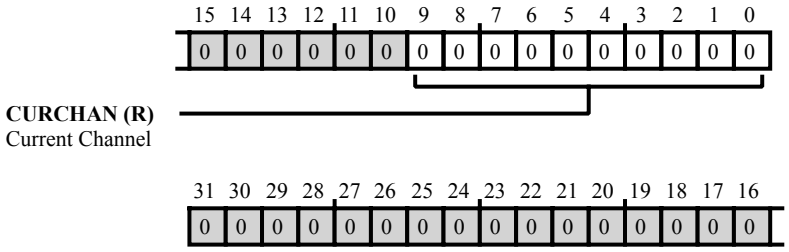

Figure 25-33: SPORT_MSTAT_A Register Diagram

Table 25-30: SPORT_MSTAT_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/NW)	CURCHAN	Current Channel. The SPORT_MSTAT_A.CURCHAN bits indicate the half SPORT's current channel being serviced in multi-channel mode.

Half SPORT 'B' Multi-channel Status Register

The SPORT_MSTAT_B register indicates the current multi-channel being serviced among the half SPORT's active channels in multi-channel mode. The half SPORT increments the value by one in this register as each channel is serviced. The value in the SPORT_MSTAT_B register restarts at 0 at each frame sync.

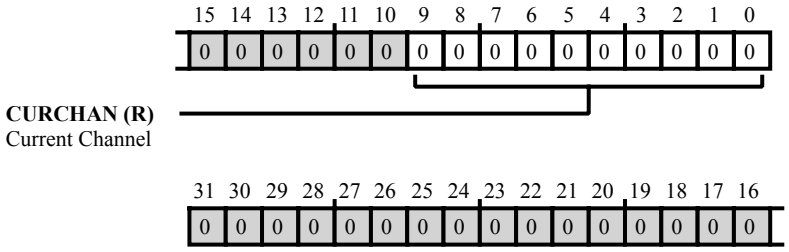

Figure 25-34: SPORT_MSTAT_B Register Diagram

Table 25-31: SPORT_MSTAT_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/NW)	CURCHAN	Current Channel. The SPORT_MSTAT_B.CURCHAN bits indicate the half SPORT's current channel being serviced in multi-channel mode.

Half SPORT 'A' Rx Buffer (Primary) Register

The SPORT_RXPRI_A register buffers the half SPORT's primary channel receive data. This buffer becomes active when the half SPORT is configured to receive data on the primary channel. After a complete word has been received in the receive shifter, it is placed into the SPORT_RXPRI_A register. This data can be read in core mode (in interrupt-based or polling-based mechanism) or directly DMA'd into processor memory using the DMA controller. With a data buffer and an input shift register, the SPORT_RXPRI_A register acts as a two-location buffer. So, the SPORT can keep a maximum of two 32-bit received words at any given time (independent of the SPORT_CTL_A.PACK bit setting).

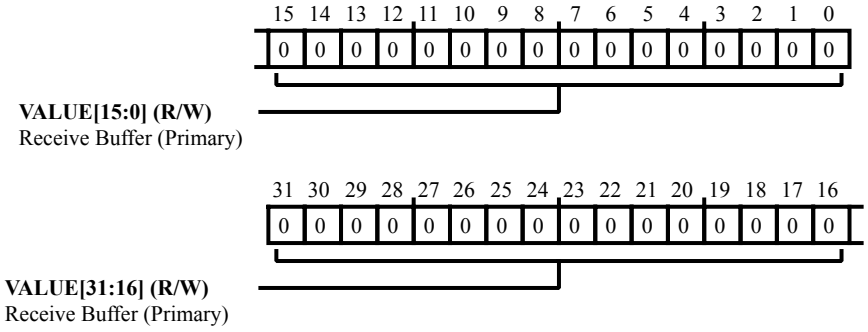

Figure 25-35: SPORT_RXPRI_A Register Diagram

Table 25-32: SPORT_RXPRI_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Primary). The SPORT_RXPRI_A.VALUE bits hold the half SPORT's primary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_A.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_A and SPORT_MCTL_A register descriptions.

Half SPORT 'B' Rx Buffer (Primary) Register

The SPORT_RXPRI_B register buffers the half SPORT's primary channel receive data. This buffer becomes active when the half SPORT is configured to receive data on the primary channel. After a complete word has been received in the receive shifter, it is placed into the SPORT_RXPRI_B register. This data can be read in core mode (in interrupt-based or polling-based mechanism) or directly DMA'd into processor memory using the DMA controller. With a data buffer and an input shift register, the SPORT_RXPRI_B register acts as a two-location buffer. So, the SPORT can keep a maximum of two 32-bit received words at any given time (independent of the SPORT_CTL_A.PACK bit setting).

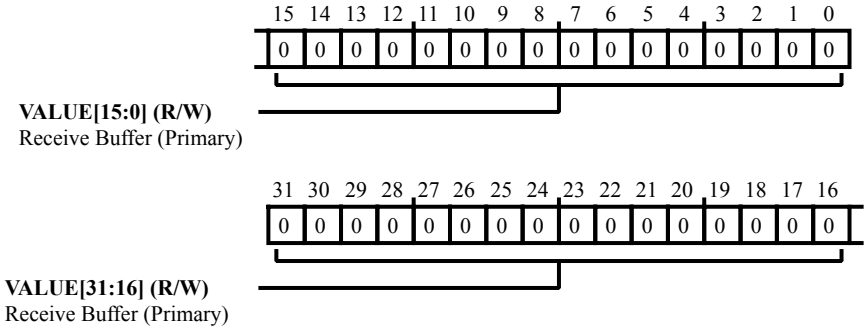

Figure 25-36: SPORT_RXPRI_B Register Diagram

Table 25-33: SPORT_RXPRI_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Primary). The SPORT_RXPRI_B.VALUE bits hold the half SPORT's primary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_B.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_B and SPORT_MCTL_B register descriptions.

Half SPORT 'A' Rx Buffer (Secondary) Register

The SPORT_RXSEC_A register buffers the half SPORT's secondary channel receive data. This buffer becomes active when the half SPORT is configured to receive data on the secondary channel. After a complete word has been received in the receive shifter, it is placed into the SPORT_RXSEC_A register. This data can be read in core mode (in interrupt-based or polling-based mechanism) or directly DMA'd into processor memory using the DMA controller. With a data buffer and an input shift register, the SPORT_RXSEC_A register acts as a two-location buffer. So, the SPORT can keep a maximum of two 32-bit received words at any given time (independent of the SPORT_CTL_A.PACK bit setting).

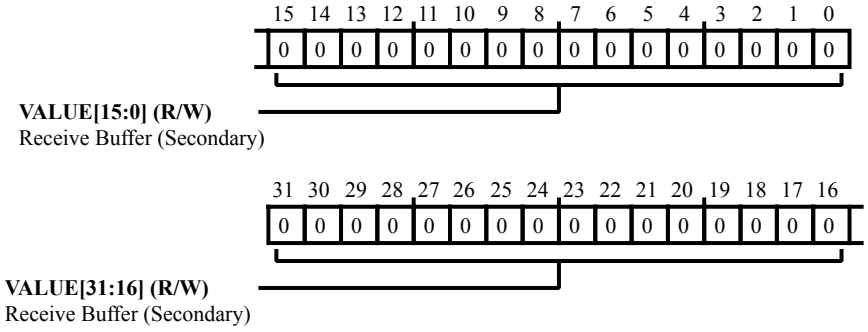

Figure 25-37: SPORT_RXSEC_A Register Diagram

Table 25-34: SPORT_RXSEC_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Secondary). The SPORT_RXSEC_A.VALUE bits hold the half SPORT's secondary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_A.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_A and SPORT_MCTL_A register descriptions.

Half SPORT 'B' Rx Buffer (Secondary) Register

The SPORT_RXSEC_B register buffers the half SPORT's secondary channel receive data. This buffer becomes active when the half SPORT is configured to receive data on the secondary channel. After a complete word has been received in the receive shifter, it is placed into the SPORT_RXSEC_B register. This data can be read in core mode (in interrupt-based or polling-based mechanism) or directly DMA'd into processor memory using the DMA controller. With a data buffer and an input shift register, the SPORT_RXSEC_B register acts as a two-location buffer. So, the SPORT can keep a maximum of two 32-bit received words at any given time (independent of the SPORT_CTL_A.PACK bit setting).

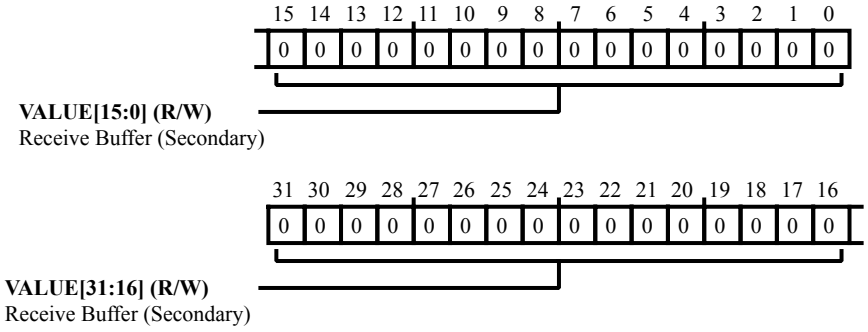

Figure 25-38: SPORT_RXSEC_B Register Diagram

Table 25-35: SPORT_RXSEC_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Secondary). The SPORT_RXSEC_B.VALUE bits hold the half SPORT's secondary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_B.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_B and SPORT_MCTL_B register descriptions.

Half SPORT 'A' Tx Buffer (Primary) Register

The SPORT_TXPRI_A register buffers the half SPORT's primary channel transmit data. This register must be loaded with the data to be transmitted if the half SPORT is configured to transmit on the primary channel. Either a program running on the processor core loads the data into the buffer (word-by-word process) or the DMA controller automatically loads the data into the buffer (DMA process).

The SPORT_TXPRI_A register acts as a three-location buffer if SPORT data packing is disabled (SPORT_CTL_A.PACK =0); while it acts as a two-location buffer when packing is enabled (SPORT_CTL_A.PACK =1). So, depending on the PACK bit setting, two 32-bit words or three 32-bit words can be stored in the transmit queue at any time. When the transmit register is loaded and any previous word has been transmitted, the SPORT_TXPRI_A register contents are automatically loaded into the output shifter. The half SPORT can issue an interrupt (transmit buffer is not full) when it has loaded the output transmit shifter, signifying that the transmit buffer is ready to accept the next word. This interrupt does not occur when the half SPORT is executing a DMA-based transfer.

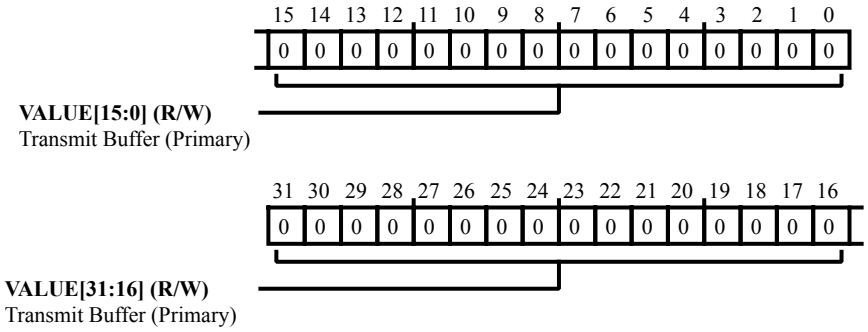

Figure 25-39: SPORT_TXPRI_A Register Diagram

Table 25-36: SPORT_TXPRI_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Primary). The SPORT_TXPRI_A.VALUE bits hold the half SPORT's primary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_A.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_A and SPORT_MCTL_A register descriptions.

Half SPORT 'B' Tx Buffer (Primary) Register

The SPORT_TXPRI_B register buffers the half SPORT's primary channel transmit data. This register must be loaded with the data to be transmitted if the half SPORT is configured to transmit on the primary channel. Either a program running on the processor core loads the data into the buffer (word-by-word process) or the DMA controller automatically loads the data into the buffer (DMA process).

The SPORT_TXPRI_B register acts as a three-location buffer if SPORT data packing is disabled (SPORT_CTL_B.PACK =0); while it acts as a two-location buffer when packing is enabled (SPORT_CTL_B.PACK =1). So, depending on the PACK bit setting, two 32-bit words or three 32-bit words can be stored in the transmit queue at any time. When the transmit register is loaded and any previous word has been transmitted, the SPORT_TXPRI_B register contents are automatically loaded into the output shifter. The half SPORT can issue an interrupt (transmit buffer is not full) when it has loaded the output transmit shifter, signifying that the transmit buffer is ready to accept the next word. This interrupt does not occur when the half SPORT is executing a DMA-based transfer.

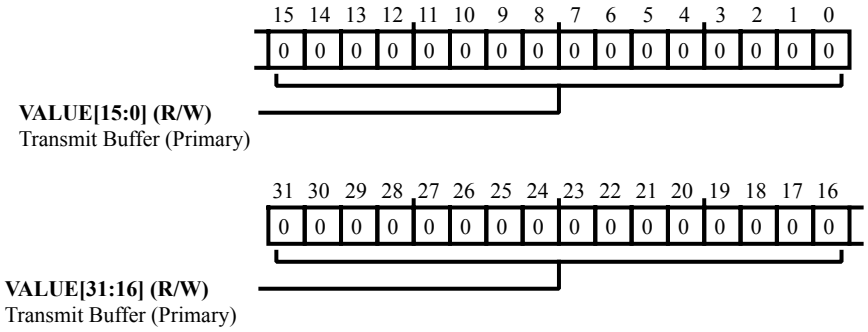

Figure 25-40: SPORT_TXPRI_B Register Diagram

Table 25-37: SPORT_TXPRI_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Primary). The SPORT_TXPRI_B.VALUE bits hold the half SPORT's primary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_B.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_B and SPORT_MCTL_B register descriptions.

Half SPORT 'A' Tx Buffer (Secondary) Register

The SPORT_TXSEC_A register buffers the half SPORT's secondary channel transmit data. This register must be loaded with the data to be transmitted if the half SPORT is configured to transmit on the secondary channel. Either a program running on the processor core loads the data into the buffer (word-by-word process) or the DMA controller automatically loads the data into the buffer (DMA process).

The SPORT_TXSEC_A register acts as a three-location buffer if SPORT data packing is disabled (SPORT_CTL_A.PACK =0); while it acts as a two-location buffer when packing is enabled (SPORT_CTL_A.PACK =1). So, depending on the PACK bit setting, two 32-bit words or three 32-bit words can be stored in the transmit queue at any time. When the transmit register is loaded and any previous word has been transmitted, the SPORT_TXSEC_A register contents are automatically loaded into the output shifter. The half SPORT can issue an interrupt (transmit buffer is not full) when it has loaded the output transmit shifter, signifying that the transmit buffer is ready to accept the next word. This interrupt does not occur when the half SPORT is executing a DMA-based transfer.

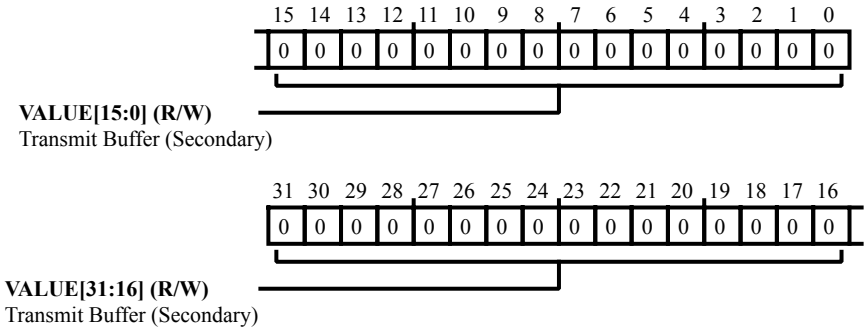

Figure 25-41: SPORT_TXSEC_A Register Diagram

Table 25-38: SPORT_TXSEC_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Secondary). The SPORT_TXSEC_A.VALUE bits hold the half SPORT's secondary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_A.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_A and SPORT_MCTL_A register descriptions.

Half SPORT 'B' Tx Buffer (Secondary) Register

The SPORT_TXSEC_B register buffers the half SPORT's secondary channel transmit data. This register must be loaded with the data to be transmitted if the half SPORT is configured to transmit on the secondary channel. Either a program running on the processor core loads the data into the buffer (word-by-word process) or the DMA controller automatically loads the data into the buffer (DMA process).

The SPORT_TXSEC_B register acts as a three-location buffer if SPORT data packing is disabled (SPORT_CTL_B.PACK =0); while it acts as two-location buffer when packing is enabled (SPORT_CTL_B.PACK =1). So, depending on the PACK bit setting, two 32-bit words or three 32-bit words can be stored in the transmit queue at any time. When the transmit register is loaded and any previous word has been transmitted, the SPORT_TXSEC_B register contents are automatically loaded into the output shifter. The half SPORT can issue an interrupt (transmit buffer is not full) when it has loaded the output transmit shifter, signifying that the transmit buffer is ready to accept the next word. This interrupt does not occur when the half SPORT is executing a DMA-based transfer.

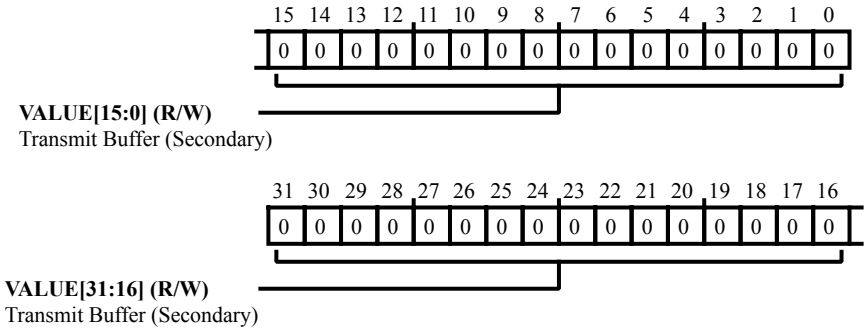

Figure 25-42: SPORT_TXSEC_B Register Diagram

Table 25-39: SPORT_TXSEC_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Secondary). The SPORT_TXSEC_B.VALUE bits hold the half SPORT's secondary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_B.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_B and SPORT_MCTL_B register descriptions.

26 Analog-to-Digital Converter Controller (ADCC)

The analog front end (AFE) includes a powerful ADC controller (ADCC), which automates the ADC sampling process and simplifies ADC accesses. The ADCC provides an interface that synchronizes the controls between the processor and an analog-to-digital converter (ADC). The processor or its peripheral infrastructure initiate the analog-to-digital conversions, based on either external or internal events, by giving the triggers to ADCC module.

NOTE: The ADCC and DACC chapters describe the control and data interface to the AFE. For information about the analog portion (I/O pins and electrical specifications) of the AFE, see the product data sheet.

Some processors do *not* include a dedicated ADC controller. For these processors, ADC sampling uses processor-interrupts (initiated by the events) and interrupt service routine programming of the appropriate peripheral to initiate the ADC conversion process. (The interrupt service routine programming typically includes a SPORT or an SPI). This approach has some limiting factors:

- ADC sampling instances are not precisely controlled due to interrupt latencies (which can vary) due to variable instruction execution cycles or multiple interrupts running in system.
- Consumption of processor MIPS can be prohibitive, especially for high frequency of conversion-related events.
- The ADC can require a sequence of frames for taking a sample from the ADC with precise timing between the frames. In this case, it can be difficult to implement the sequence with delay routines in the application. (For example, a sequence could include sending control bits, then a conversion pulse and then reading converted data).

The ADCC provides dedicated hardware to address some of these limitations. The hardware initiates the ADC sampling (based on some processor or its peripheral events) by providing sampling signals with required timings in real time. Using the ADCC permits flexible scheduling of sampling instants and provides precise control execution of timing and analog sampling events on the ADCs. The ADCC saves both processor MIPS and provides precise controllability for ADC sampling time.

The converted ADC data from the ADCs is directly available in ADCC registers. The data can be read in core mode to store data into processor memory or can be directly routed through a dedicated DMA.

On the ADSP-CM40x microcontroller, the ADCC is designed to interface with the on-chip internal ADCs, which require minimal core intervention.

ADCC Features

The ADCC provides many architecture-based features (basic to the design) and mode-selectable features (usage is optional or configurable).

Architecture-based features of the ADCC include:

- Two ADC interfaces to control two 16-bit ADCs independently
- Automated ADC sampling with ADC control hardware for sending the control word, executing a conversion cycles, and reading the converted data with programmable timing
- Up to six trigger inputs which permit the ADCC to initiate the ADC sampling events precisely

  Either the processor core or its peripheral infrastructure generates the trigger inputs internally.
- Up to 24 ADC sampling events per valid trigger received

  Each event can be assigned to either of the ADCC timers and assigned to one of the ADCs (and its channel). Each event is independently programmable to specify when to initiate ADC sampling based on the trigger input.
- 24 event status registers (one per each event), which indicate the amount of delay before the start of event handling after the event match occurs
- Two independent 32-bit ADCC timers for controlling ADCC operation from two different sources

  Any of the 24 events can be assigned to any ADCC timer. The ADCC automatically stops these timers after completion of associated events to save power. The trigger input to ADCC timer can be enabled or masked on-the-fly to disable all the events related to a trigger source temporarily.
- Separate eight-deep pending FIFO for each ADC interface to queue the active events when the corresponding ADC is busy
- Serial clock, chip select, control signals, and data signals to control the ADC operations during control, conversion, and data read phases.
- Internally generated ADC clock from processor system clock

  This clock is gated (for example, it is active only when controlling ADC) to provide excellent noise immunity during conversion process. The clock polarity (for example, the first edge after chip select signal assertion) is configurable.
- Automated ADC sampling process, placing the converted data from ADC in a directly available in ADCC event data register

  This data can be read in core mode to store the data in required memory space, or the data can be directly DMA transferred.
- A built-in DMA unit

There is a DMA unit with a channel for each ADCC timer for DMA transferring the converted data of the associated events. The DMA unit supports an optional mechanism for circular buffering.

- Error detection capabilities, including support for the detection of an event miss, event collision, ADCC timer trigger overrun, bandwidth monitor error when using DMA, and memory write response error conditions
- Trigger master capability to provide two trigger signals to TRU unit on completion of each ADCC timer frame

Mode-selectable features of the ADCC include:

- Simultaneous sampling of both ADCs
- Chip select signal with configurable width and polarity

  The ADC chip select signal has a programmable width in terms of the ADC clocks. The polarity of this signal is configurable as active high or active low signal. The ADC uses this signal to accept control word, to start the conversion, and to output the converted data.
- Single-bit mode or dual-bit mode (with extra options) for data transfers to and from the ADCs

  Two lines are available for sending the ADC control word, but the control word to ADC can be sent on single line or on both lines. The control lines can be driven to idle state (for example, either high or low) when a valid control word is not driven to ADC. The control word can be sent LSB-bit first or MSB-bit first. When using two lines for sending the control word, the first bit can optionally start on either line.

  Two lines are available for reading data from ADC, but either a single line or both lines can be used for reading ADC data. The data can be read LSB-bit first or MSB-bit first. When using two lines for reading data, the first bit can optionally start on either line.
- Bandwidth monitoring option (when using DMA mode of data read operation), which identifies whether the timer count has gone beyond an expected limit when receiving an ADC data frame

  This option also provides a status bit to indicate whether DMA is pending reception of data from ADC.
- Status indication (status bits) and optional interrupt generation (when using core mode of data read operation) to core on completion of each event
- Interrupt can be optionally generated at the end of each ADCC timer frame completion or on ADCC error detection

ADCC Functional Description

The ADCC controller provides a means to automate the ADC sampling sequence precisely. It reduces the core overhead required (compared to sampling methods where no dedicated ADC controller is present) and simplifies the ADC accesses. Two independent ADC interfaces in the ADCC synchronize the controls between the processor and the two on-chip ADCs. These features permit flexible scheduling of sampling instants and provide precise control execution of timing and analog sampling events on the ADCs.

NOTE: Refer to the following sections for supplementary information about ADCC functionality:

- ADCC Signal Descriptions
- ADCC Block Diagram
- ADCC Architectural Concepts

The processor or its peripheral infrastructure (based on either external or internal conditions) provides trigger signals to the ADCC module to initiate the ADC conversions. The ADCC can simultaneously accept up to six trigger inputs from different sources (for example, from trigger masters of the TRU of the processor).

Two 32-bit counters, known as ADCC timers, are central to ADCC operation. These timers each independently accept one of the trigger inputs and start counting at the system clock rate of the processor. The counting begins when a valid edge is detected on the selected trigger input.

NOTE: Because the two ADCC timers can each accept a different trigger input, the counts in the ADCC timers cannot always be the same.

The ADCC provides 24 events that can be independently programmed to occur at the specified time after the trigger signal. This time is configured in the time bit field, `ADCC_EVT[nn].TIME` of the corresponding event time register. The event can be assigned to either to ADCC TMR0 or ADCC TMR1, by configuring the timer select bit field, `ADCC_EVCTL[nn].TMRSEL` of the corresponding event control register. As the ADCC timers count at SYSCLK rate, the event comparator unit compares the event time of enabled events with the current count of the assigned ADCC timer. If the count matches, the particular event is active.

The event control register of each event also specifies which ADC interface executes the sampling sequence of the event. The `ADCC_EVCTL[nn].ADCSEL` bit field selects either ADC0 interface or ADC1 interface. Also, the event control register stores a control word, which (when sent to an ADC) selects the analog input channel that the ADC uses for conversion. When an event becomes active, the comparators signal the activity to the timing and control unit of the corresponding ADC interface.

If the ADC interface is ready to initiate an ADC sampling sequence, the interface starts the sequence for the current active event. When the ADC interface is busy, the new active event is stored in an 8-deep pending event FIFO.(For example, the ADC interface is busy when an event becomes active while it is already executing the sampling sequence of another event). The interface executes events from the pending FIFO (in chronologically order) when the ADC interface is free.

The following example illustrates the trigger, event, sample, and data transfer (channel) process. The ***ADCC Operation Example - Relating All Signals*** figure provides an overview of ADCC operations.

This example assumes that an application wants to sample two analog inputs. One input is from each ADC interface, at every PWM period. Another input is two samples from different analog inputs, one from each ADC interface, whenever there is some activity on a GP input pin of the processor.

The application needs a total of four ADCC events. Two are triggered every PWM period, and the other two are triggered whenever there is an edge on the general-purpose input pin. This example also assumes that events 0 through 3 are selected for this purpose.

For simplicity, assume that the pulse-width modulator (PWM) triggers event 0 (EVT-0) and event 1 (EVT-1). Event 0 initiates ADC0 sampling for the ADC0_CHN0 channel, while event 1 initiates ADC1 sampling for the ADC1_CHN1 channel. Also, a general-purpose pin input triggers event 2 (EVT-2) and event 3 (EVT-3). Event 2 initiates ADC0 sampling for the ADC0_CHN2 channel, while event 3 initiates ADC1 sampling for the ADC1_CHN3 channel.

Because there are two trigger sources in the application, both ADCC timers (TMR0 and TMR1) are enabled. One of the timers (in the example, TMR0) accepts trigger from the PWM master, while other timer (in the example, TMR1) accepts trigger from the general-purpose pin input. The PWM_SYNC signal is used to generate trigger signal in each cycle. The PINT block is used to provide trigger signal from the edges appearing on the general-purpose input pin. The TRU of the processor routes the trigger signals from the trigger masters (in the example, PWM and PINT) to the trigger slave (in the example, TMR0 and TMR1).

The activation of events after the trigger input is configurable in the event time register. The ***ADCC Operation Example - Event Information*** table provides a list of event information.

Table 26-1: ADCC Operation Example - Event Information

	ADC0_interface	ADC channel	ADCC Timer	Triggered by	Event Time
Event-0	0	ADC0_CHN0	0	PWM Sync	EVT0 < EVT1
Event-1	1	ADC1_CHN1	0	PWM Sync	EVT0 < EVT1
Event-2	0	ADC0_CHN2	1	GP Input	EVT3 < EVT2
Event-3	1	ADC1_CHN3	1	GP Input	EVT3 < EVT2

In this case, the ADCC signals can look as shown in ***ADCC Operation Example - Relating All Signals***.

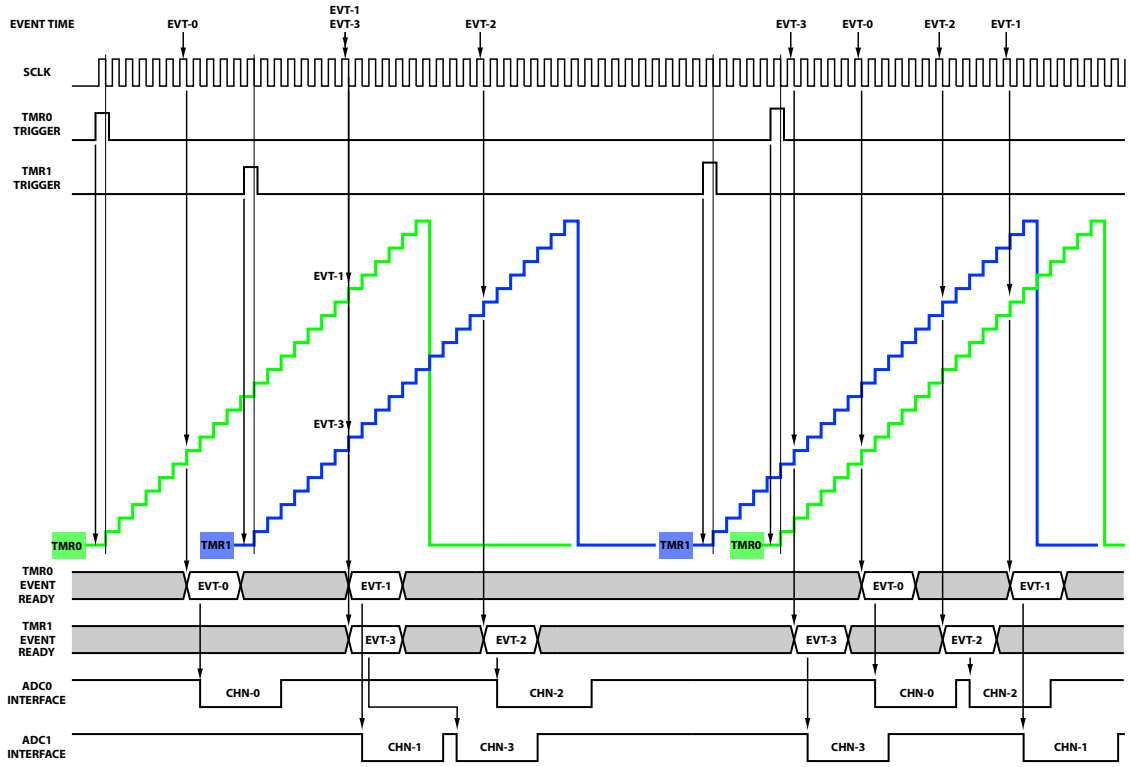

Figure 26-1: ADCC Operation Example - Relating All Signals

Note the following key points from the ***ADCC Operation Example - Relating All Signals*** figure:

- The ADCC timers start counting at the SYSCLK rate when a valid edge on a selected trigger is detected. The event comparators compare the timer count with the event time register of associated events. The comparators signal that the event is “Ready” on a count match.
- Each timer continues to run until all the associated events complete. The exact time can depend on usage of core mode or DMA mode of operation for reading the ADC data. After all the events complete, the ADCC timer count is reset to zero.
- Because the trigger inputs of both of the timers are asynchronous to each other, there is possibility that events associated with them can overlap.

  In the example, note that event 1 and event 3 have occurred at the same time. (For example, event comparators have signaled event 1 of TMR0 and event 3 of TMR1 as ready in the same SYSCLK cycle). The priority in this case goes to the event associated with TMR0, and event 1 is serviced first. Event 3 is placed in the pending event FIFO, and the event collision error bit is set.
- For simplicity, the ***ADCC Operation Example - Relating All Signals*** figure shows the collided events as separately serviced, one after the other. The ADC interface can decide during real-world operation to pipeline the ADC sampling. The ***ADCC Operation Example - Event Tightly Pipelined*** figure describes this operation.
- The pulse on the ADC interface line shows ADC sampling sequence symbolically. This presentation does not detail the exact sampling sequence and does not show the ADCC signals used to control ADCs.

- There is no restriction for assigning events to ADCC timer 0 or timer 1. There is also no restriction for assigning a particular event to the ADC0 interface or ADC1 interface. All the events can be independently programmed.

When an event becomes active, the ADC sampling sequence related to it starts when the ADC interface is ready to initiate the sampling process. The ADC sampling sequence is divided into phases:

- Control phase - the ADCC sends the control word to select the ADC channel for conversion. Typically, one ADC provides multiple input channels and one of them can be selected for conversion by specifying the channel-ID while sending the control word. The control word for each event is stored in the `ADCC_EVCTL[nn].CTLWD` field. For more information about the ADC control word, see ADCC Programming Guidelines.
- Conversion phase - the ADCC sends a conversion pulse to initiate the conversion for selected ADC channel.
- Data phase - the ADCC reads the converted data from ADC. ADCC provides chip select or start of conversion signal for each of these phases.

The ADCC also provides ADC clock in gated format, which is active only during these phases. Two control lines and two data lines are provided through which each control word is sent and converted data is read. A typical timing in this case, appears in the ***ADCC Operation Example - Event Spaced*** figure:

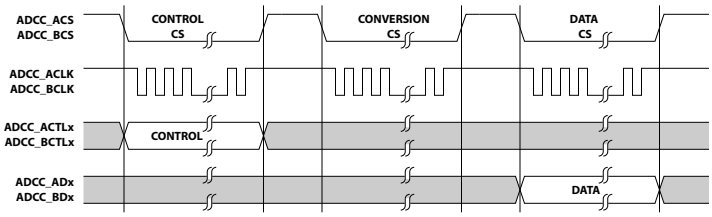

Figure 26-2: ADCC Operation Example - Event Spaced

The ADCC hardware automates this sampling process, and the converted data is directly available to read in the data register of that event. When the ADCC starts to execute an event, the conversion data for event is only available during the third frame being sent. If another event becomes active during this time, service for the newly arrive frame is delayed. To reduce the latency of service for an event in collision events, the ADCC can decide to pipeline the ADC sampling sequence of different events. (An event is *in collision* if it becomes active while the ADCC is busy executing a previously received event).

Refer to the ***ADCC Operation Example - Event Overlapped*** figure. Consider a case in which event 1 becomes active while the ADC interface is providing the conversion pulse for the event 0 ADC sampling sequence. The ADCC starts the sampling sequence for event 1 in the next phase. (For example, the ADCC sends the control word for event 1 while receiving data for the event 0 sample).

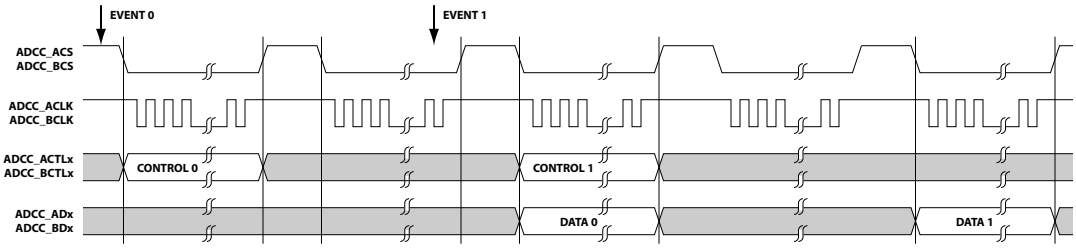

Figure 26-3: ADCC Operation Example - Event Overlapped

The ***ADCC Operation Example - Event Tightly Pipelined*** figure shows another example in which the ADC interface tightly packs the ADC sampling sequence. When the ADC interface sends the control word for an event, the interface provides a conversion pulse to the previous event and reads the converted data of the previous event. This example is the *maximum ADC throughput*" case, because the ADC continuously samples its channels.

To achieve as close to maximum throughput as possible, the ADCC tries to pipeline operations whenever feasible. A single chip select can accomplish these operations simultaneously:

- Data reception of the n^{th} control word
- Act as the conversion chip select of the (n^{th} + 1) control word
- Send out the (n^{th} + 2) control word

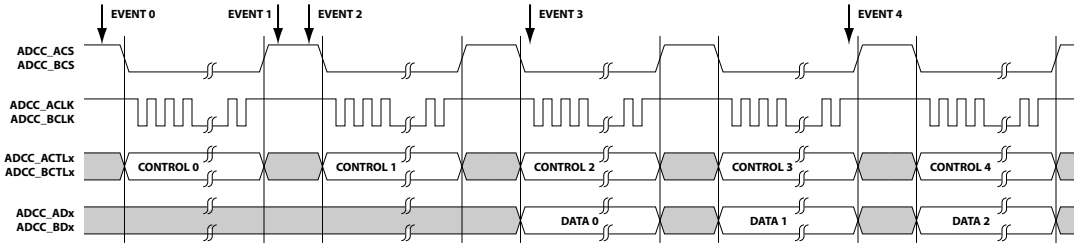

Figure 26-4: ADCC Operation Example - Event Tightly Pipelined

ADSP-CM40x ADCC Register List

The ADC controller (ADCC) automates the ADC sampling process and simplifies ADC accesses. The ADCC provides an interface that synchronizes the controls between the processor and an analog-to-digital converter (ADC). A set of registers govern ADCC operations. For more information on ADCC functionality, see the ADCC register descriptions.

Table 26-2: ADSP-CM40x ADCC Register List

Name	Description
ADCC_BPTR0	Base Pointer 0 Register
ADCC_BPTR1	DMA Base Pointer 1 Register
ADCC_BWMON0	Bandwidth Monitor 0 Register

Table 26-2: ADSP-CM40x ADCC Register List (Continued)

Name	Description
ADCC_BWMON1	Bandwidth Monitor 1 Register
ADCC_CBSIZ0	Circular Buffer Size 0 Register
ADCC_CBSIZ1	Circular Buffer Size 1 Register
ADCC_CFG	ADC Configuration Register
ADCC_CTL	Control Register
ADCC_ECOL	Event Collision Status Register
ADCC_EIMSK	Event Interrupt Mask Register
ADCC_EIMSK_CLR	Event Interrupt Mask Clear Register
ADCC_EIMSK_SET	Event Interrupt Mask Set Register
ADCC_EISTAT	Event Interrupt Status Register
ADCC_EMISS	Event Miss Status Register
ADCC_EPND	Pending Events Status Register
ADCC_ERRMSK	Error Mask Register
ADCC_ERRMSK_CLR	Error Mask Clear Register
ADCC_ERRMSK_SET	Error Mask Set Register
ADCC_ERRSTAT	Error Status Register
ADCC_EVCTL[nn]	Event n Control Register
ADCC_EVDAT[nn]	Event n Data Register
ADCC_EVSTAT[nn]	Event n Status Register
ADCC_EVTEN	Event Enable Register
ADCC_EVTEN_CLR	Event Enable Clear Register
ADCC_EVTEN_SET	Event Enable Set Register
ADCC_EVT[nn]	Event n Time Register
ADCC_FIMSK	Frame Interrupt Mask Register
ADCC_FIMSK_CLR	Frame Interrupt Mask Clear Register
ADCC_FIMSK_SET	Frame Interrupt Mask Set Register
ADCC_FISTAT	Frame Interrupt Status Register
ADCC_FRINC0	Frame Increment 0 Register
ADCC_FRINC1	Frame Increment 1 Register
ADCC_T0STAT	Timer 0 Status Register
ADCC_T1STAT	Timer 1 Status Register

Table 26-2: ADSP-CM40x ADCC Register List (Continued)

Name	Description
ADCC_TCA0	Timing Control A (ADC0) Register
ADCC_TCA1	Timing Control A (ADC1) Register
ADCC_TCB0	Timing Control B (ADC0) Register
ADCC_TCB1	Timing Control B (ADC1) Register
ADCC_TMR0	Timer 0 Current Count Register
ADCC_TMR1	Timer 1 Current Count Register

ADSP-CM40x ADCC Interrupt List

Table 26-3: ADSP-CM40x ADCC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
44	ADCC0_ERR	ADCC0 Error	Level	
94	ADCC0_TMR0_EVT	ADCC0 Timer 0 Event Complete	Level	
95	ADCC0_TMR1_EVT	ADCC0 Timer 1 Event Complete	Level	

ADSP-CM40x ADCC Trigger List

Table 26-4: ADSP-CM40x ADCC Trigger List Masters

Trigger ID	Name	Description	Sensitivity
22	ADCC0_TMR0_EVT	ADCC0 Timer 0 Event Complete	Level
23	ADCC0_TMR1_EVT	ADCC0 Timer 1 Event Complete	Level

Table 26-5: ADSP-CM40x ADCC Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
24	ADCC0_SLV0	ADCC0 Trigger Slave 0	None
25	ADCC0_SLV1	ADCC0 Trigger Slave 1	None
26	ADCC0_SLV2	ADCC0 Trigger Slave 2	None
27	ADCC0_SLV3	ADCC0 Trigger Slave 3	None
28	ADCC0_SLV4	ADCC0 Trigger Slave 4	None
29	ADCC0_SLV5	ADCC0 Trigger Slave 5	None

ADCC Signal Descriptions

The ADCC controls the operation of internal ADCs, based on the settings for enabled events and trigger input. Because the ADCs are internal, none of the ADCC signals are available as external package pins.

NOTE: There are two instances of the ADC interface on the ADSP-CM40x. Each offers separate clock, chip select, control, and data signals for controlling the two ADCs independently.

A number of signals connect the ADCC to the ADCs for providing ADC clock, chip select, and control. Using these signals, the ADCC regulates the ADC sampling sequence and reads the converted data. These signals appear in the ***ADSP-CM40x ADCC-to-ADC0/1 Signal Descriptions*** table.

Table 26-6: ADSP-CM40x ADCC-to-ADC0/1 Signal Descriptions

Name	I/O	Description
$\overline{\texttt{ADCC\_ACS}}$	O	ADCC ADC0 (A) chip select (or start-of-conversion) for the ADC
ADCC_ACLK	O	ADCC ADC0 (A) clock
ADCC_ACTL0	O	ADCC ADC0 (A) control 0 for sending control word
ADCC_ACTL1	O	ADCC ADC0 (A) control 1 for sending control word
ADCC_AD0	I	ADCC ADC0 (A) data 0 for reading converted data from the ADC
ADCC_AD1	I	ADCC ADC0 (A) data 1 for reading converted data from the ADC
$\overline{\texttt{ADCC\_BCS}}$	O	ADCC ADC1 (B) chip select (or start-of-conversion) for the ADC
ADCC_BCLK	O	ADCC ADC1 (B) clock
ADCC_BCTL0	O	ADCC ADC1 (B) control 0 for sending control word
ADCC_BCTL1	O	ADCC ADC1 (B) control 1 for sending control word
ADCC_BD0	I	ADCC ADC1 (B) data 0 for reading converted data from the ADC
ADCC_BD1	I	ADCC ADC1 (B) data 1 for reading converted data from the ADC

The ADCC also has a number of signals to connect the ADCC to the microcontroller core. Using these signals, the ADCC sends trigger and error information to the TRU and SEC. These signals appear in the ***ADSP-CM40x ADCC-to-Core Signal Descriptions*** table.

Table 26-7: ADSP-CM40x ADCC-to-Core Signal Descriptions

Signal Name	I/O	Signal Description
ADCC_SLV[n]	I	ADCC Trigger Inputs [5:0] for starting a new ADCC frame.
ADCC_TMR0_EVT	O	TMR0 Event Trigger for signaling the TRU when an ADC0 frame is completed
ADCC_TMR1_EVT	O	TMR1 Event Trigger for signaling the TRU when an ADC0 frame is completed
ADCC_ERR	O	ADC Error Interrupt.

The following are more detailed descriptions of each ADCC signal type.

ADCC_ACLK, ADCC_BCLK

The ADCC clock provides the clock source for communicating with ADCs, which some of the ADCs can also use for the conversion process. The ADCC provides this clock in gated format (for example, active only when controlling the ADCs) to ensure excellent noise immunity during the conversion process.

The ADC clock is internally generated from system clock of processor. For example, the ADCC_TCA0.CKDIV bit field specifies the divider to generate the ADC0 clock signal from SYSCLK. This divisor is a 16-bit field, allowing a wide range of clock rates for ADC operation.

Use the following equation to calculate the ADC0 clock frequency:

```
ADCC_ACLK = (SYSCLK) / ( ADCC_TCA0.CKDIV + 1)
```

Alternatively, calculate the clock divisor required for the ADC0 clock frequency as:

```
ADCC_TCA0.CKDIV = (SYSCLK ÷ ADCC_ACLK ) – 1
```

The minimum and default value of ADCC_TCA0.CKDIV field is 0. Do not set the divisor field to zero when SYSCLK is greater than the maximum clock frequency the ADC supports.

Both the ADC and the ADCC use this clock to drive the output signals and sample the incoming signals. The clock polarity (selection of reference edges or first edge after chip select signal assertion) is configurable (for example, using the ADCC_CTL.CKPOL0 bit).

$\overline{\text{ADCC\_ACS}}$, $\overline{\text{ADCC\_BCS}}$

The ADCC provides a chip select signal to select the ADC for communication. It asserts the signal while sending the control word, during conversion period, or while reading converted data. These three phases form the ADC sampling sequence, which starts based on a trigger input and the settings of one or more enabled events.

The width and period of the chip select signal is configurable based on ADCC timing register settings. For example on ADC0, the width of the chip select signal is configurable based on the ADCC_TCA0.NCK, ADCC_TCB0.TCSCK, and ADCC_TCB0.TCKCS fields. The active duration of ADC0 clock select is calculated as (in terms of ADC0 clock cycles):

$\overline{\text{ADCC\_ACS}}$ Active Duration = (ADCC_TCB0.TCSCK + ADCC_TCA0.NCK + ADCC_TCB0.TCKCS)

The ADCC uses the ADCC_TCB0.TCSCS field to control the period of the ADC0 chip select signal, as

Chip Select Period = $\overline{\text{ADCC\_ACS}}$ Active Duration + ADCC_TCB0.TCSCS)

The ***ADCC Clock Signal and Chip Select Signal Description*** figure shows the timing relationships provided through the timing register settings. The polarity of the chip select signal can be configured as active-high or active-low (for example, using theADCC_CTL.CSPOL0 bit).

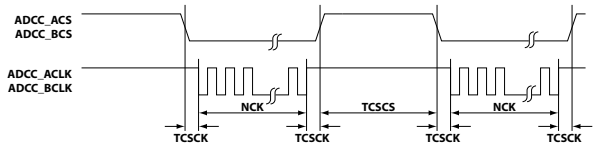

Figure 26-5: ADCC Clock Signal and Chip Select Signal Description

ADCC_ACTL0, ADCC_ACTL1, ADCC_BCTL0, ADCC_BCTL1

The ADCC provides two control signal lines for sending the control word to the ADC serially. But, depending on the ADC interface settings, the ADCC can send the control word on a single line or use both lines, based on the control word size selection (`ADCC_CTL.CSIZE` bit).

When using two lines, the first bit of the control word can start on either of the control signal lines, based on the data swap setting (for example, `ADCC_CTL.DSWP0` bit). The control word for each event is stored in the control word field of the corresponding event control register, `ADCC_EVCTL[nn].CTLWD`. The control word indicates the ADC channel for the next conversion process and selects whether the ADC uses async mode or sync mode of conversion. The ADCC sends the control word at the start of the ADC sampling sequence. It transmits the bits of the word on active edges of the ADC clock (as selected with the clock polarity selection).

When a valid control word is not driven to ADC, the control lines can be driven to a high-idle or low-idle state. The state is based on the transmit idle setting (for example, `ADCC_CTL.TIDLE0` bit). The ADCC can send the control word in LSB-first format or MSB-first format, depending on the LSB-first mode setting (for example, `ADCC_CTL.LSBF0` bit).

ADCC_AD0, ADCC_AD1, ADCC_BD0, ADCC_BD1

The ADCC provides two lines for reading converted data from the ADC data serially. But, based on the ADC interface setting, the ADCC can read ADC data from a single line or use both lines depending on the data word size selection (`ADCC_CTL.DSIZE` bit).

When using two lines, the first bit of the data word can start on either of the data signal lines, based on the data swap setting (for example, `ADCC_CTL.DSWP0`bit). The ADCC reads the converted data during the data phase of ADC sampling. It samples the data on active edges of the ADC clock (as selected with the clock polarity selection). The data for each event is stored in the data register of the corresponding event register. The data register can be read in core mode or can be directly DMA transferred to a selected memory location.

The ADCC can read the data in LSB-first format or MSB-first format, depending on the LSB-first mode setting (for example, `ADCC_CTL.LSBF0` bit).

ADCC_SLV[n]

The ADCC can accept up to six trigger inputs, which the trigger routing unit (TRU) of the processor routes internally. Only two of these triggers can be accepted at a time. The ADCC trigger selection field (for example, `ADCC_CTL.TRGSEL0`) assigns one trigger for each ADCC timer.

Two 32-bit internal timers (TMR0/TMR1) in the ADC can be independently configured as the trigger inputs. If both timers are enabled, the ADCC *either*

- accepts the two trigger inputs (if both are selected for different triggers) *or*
- accepts a single trigger input (if both are selected for same trigger)

The ADCC ignores non-selected trigger inputs.

When the ADCC detects a valid trigger, the corresponding ADCC timer starts counting at the rate of the processor system clock (SYSCLK). The ADCC treats trigger inputs as edge-sensitive, and makes the selection of rising- versus falling-active with the trigger polarity selection (for example, using the `ADCC_CTL.TRGPOL0` bit).

`ADCC_TMR0_EVT, ADCC_TMR1_EVT`

The ADCC can act as a trigger master, providing one of two trigger signals to the TRU on completion of each of the ADCC timer frames. The triggers can be selectively enabled (for example, using the `ADCC_CTL.TRGOE0` bit).

`ADCC_ERR`

If the interrupt is enabled, the ADCC generates an error interrupt output when any error status is:

- set in the `ADCC_ERRSTAT` register, and
- unmasked (enabled) with the corresponding bit in the `ADCC_ERRMSK` register

The ADCC passes this interrupt to the SEC of the processor for handling.

ADCC Block Diagram

The ADCC controller consists of a trigger input selection multiplexer, two independent 32-bit timers, 24 event register bank, 24 event comparators, a pending event FIFO, and a timing generation unit for two ADC interfaces.

Also, the ADCC incorporates two in-built DMA units, one for each ADCC timer, to store ADC samples directly in required memory space.

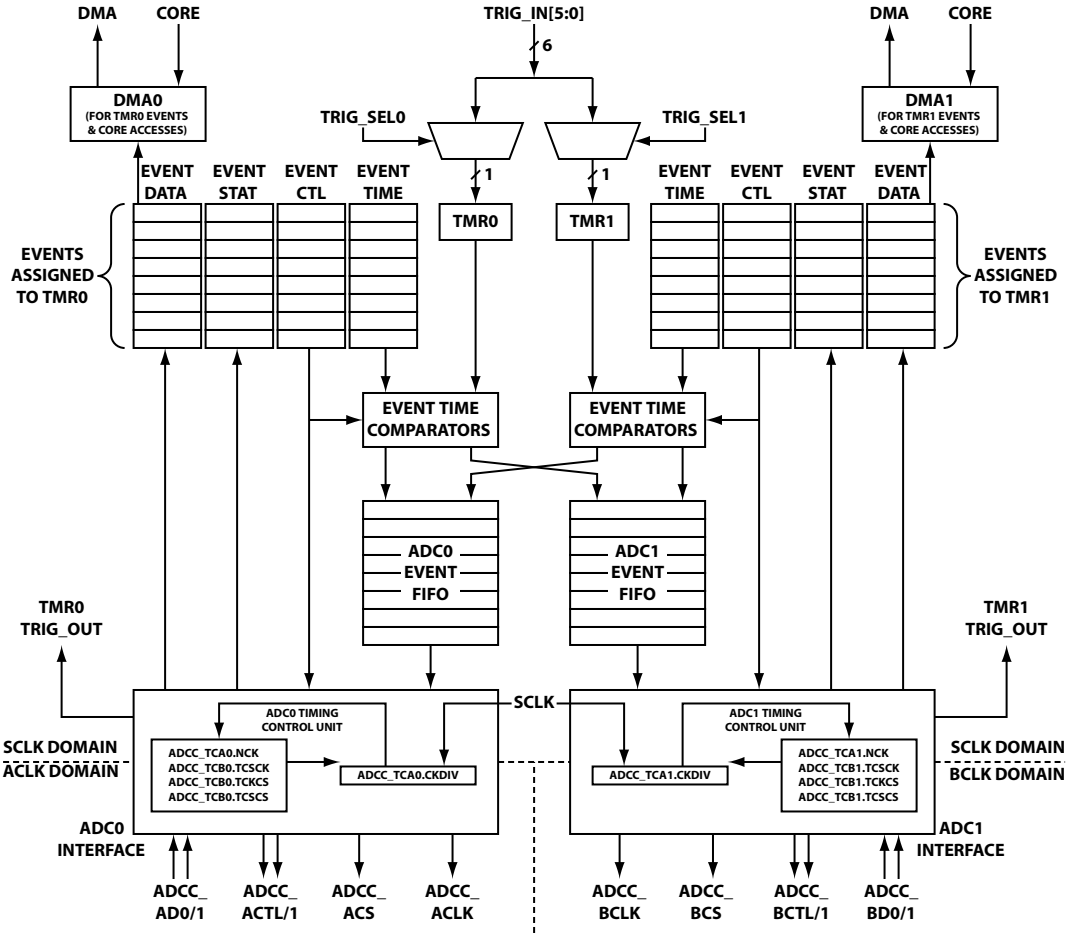

Figure 26-6: ADCC Block Diagram

NOTE: For more information about each part of the ADCC, see ADCC Architectural Concepts.

ADCC Architectural Concepts

The following sections describe the blocks appearing in the ADCC Block Diagram.

- Core and DMA Interfaces
- Trigger Inputs
- Timers
- Event Register Banks
- Event Comparators
- Pending Event FIFO
- Timing and Control Unit

Core and DMA Interfaces

The ADCC has a 32-bit core interface through which the core programs the ADCC control registers and reads the ADCC status registers. The ADCC can also use this interface for reading the 16-bit converted ADC data stored in event data registers in core mode.

To minimize the core overhead, the ADCC provides a 16-bit DMA interface for directly DMA transferring converted ADC data from the event data registers to the required memory space. This interface consists of two built-in DMA units, one for each ADCC timer.

Trigger Inputs

The ADCC can accept up to six trigger inputs, based on which ADCC timers start running at processor system clock rate (SYSCLK). Also, the ADCC contains two 32-bit internal timers (TMR0 and TMR1). Each timer can be independently configured to use one of the trigger inputs. The `ADCC_CTL.TRGSEL0` bit field selects the trigger input for TMR0, and the `ADCC_CTL.TRGSEL1` bit field selects the trigger input for TMR1. When both ADCC timers are enabled for different trigger inputs, the ADCC uses two trigger inputs. The ADCC uses one trigger input when both ADCC timers are enabled for same trigger input or if only one ADCC timer is enabled. The ADCC ignores non-selected trigger inputs.

The ***ADCC Trigger Sources for the ADSP-CM40x*** figure shows the detailed ADCC trigger generation logic. The trigger routing unit (TRU) provides all these triggers of the processor. The TRU provides system-level sequence control without core intervention. Configure the slave-select (SSR) field of the selected trigger input to the receive triggers from a specific trigger master. In this way, the ADCC slave trigger can accept triggers asserted by that particular trigger master or asserted through software. Software writes the ID of that trigger master to one of the fields in the TRU_MTR register. The trigger-out response from the selected master is internally routed to the ADCC trigger input. For more information about the TRU and trigger slaves or masters, see the Trigger Routing Unit (TRU) chapter.

For trigger input signals, the active edge of the trigger is programmable as either rising edge or falling edge trigger, using the `ADCC_CTL.TRGPOL0` or `ADCC_CTL.TRGPOL1` bit. For internal trigger inputs (provided by TRU), the falling edge appears after one system clock from the corresponding rising edge.

If the trigger input appears before completion of ADCC frame, the trigger overrun status bit is set and the ADCC (optionally) generates an error interrupt. Because each ADCC timer can initiate the ADCC frame, the ADCC provides two trigger overrun bits, one for each timer.

ADCC provides an option that uses the `ADCC_CTL.TRGOE0` or `ADCC_CTL.TRGOE1` bit to enable or disable the trigger input to the ADCC timer. When cleared, the trigger inputs are not considered valid to initiate ADCC frames.

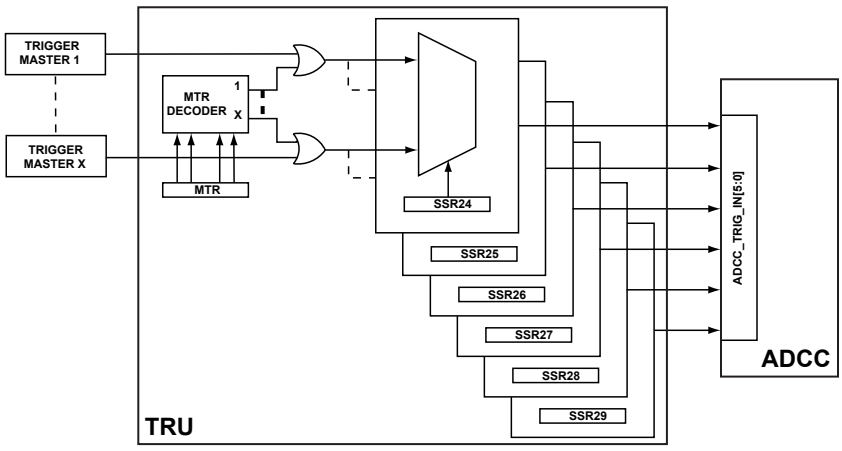

Figure 26-7: ADCC Trigger Sources for the ADSP-CM40x

Timers

The ADCC provides two independent 32-bit timers (TMR0 and TMR1). TMR0 is enabled by default when the controller is enabled. TMR1 is enabled optionally using the `ADCC_CTL.TMR1EN` bit.

Any of the six trigger inputs can be assigned to each ADCC timer using the `ADCC_CTL.TRGSEL0`, and `ADCC_CTL.TRGSEL1` bit fields.

These timers start counting at system clock rate (SYSCLK) when a valid edge is detected on the selected trigger input. The timer only stops counting under one of the following conditions:

- A timer rollover occurs.
- All the events associated with the trigger have completed.

The *timer-rollover case* can never happen unless the event time register of an event is programmed at some point after the trigger occurs. This programming is a practice that is contrary to the guidelines provided in the ADCC Programming Model.

In the *all-events-completed case*, the exact time at which the timer stops counting depends on the core mode or DMA mode of the ADC data read operation. In DMA mode, the timer stops counting only when all ADC data related to ADCC frame are written into processor memory and the memory write response returns successfully.

When an ADCC timer is disabled or the ADCC controller is disabled, the timer resets to zero.

Event Register Banks

A sample to be taken from an ADC is referred to as an event. The events can be programmed to occur at specified times after a trigger input to the ADCC. The sample can be from any ADC channel, either in async mode or sync mode. An event is complete when the data reception (for the event) completes.

The ADCC can handle a total of 24 events which can be independently configured and enabled. Each event can be assigned to an ADCC timer and can instruct the ADCC to start a sampling sequence for either ADC0 or ADC1.

Each event consists of a four register bank, including an event control register (ADCC_EVCTL[nn]), an event time register (ADCC_EVT[nn]), an event status register (ADCC_EVSTAT[nn]), and an event data register (ADCC_EVDAT[nn]). The ADCC_EVTEN register contains event enable bits.

The event control register determines the following:

- Event assignment to TMR0 or TMR1
- Event execution on the ADC0 interface or the ADC1 interface
- ADC channel selection for sampling
- Requirement for simultaneous sampling with another ADC
- Memory offset for event data storage for reading ADC data in DMA mode

The event time register specifies the time offset from the corresponding ADCC timer trigger input to the start of that particular event. (For example, the time offset is when the event must happen based on the trigger input). This time offset is specified in terms of system clock of the processor.

The event status register provides the delay in number of system clock cycles after the event match occurred. This information is useful for debugging purpose.

The result of ADC sampling (for example, converted data from the ADC) is directly stored into the event data register. The data can be accessed in core mode or can be directly DMA transferred into processor memory.

An ADCC timer frame completes when all the events associated with that ADCC timer complete after detecting valid trigger. At least one event per ADCC timer must be enabled for the ADCC to execute an ADC sampling sequence.

Event Comparators

The event comparator block consists of 24 event time comparators, which determine when an enabled event is ready for handling. After detecting a valid edge on a selected trigger input, the ADCC timer starts running at system clock rate (SYSCLK). The comparators compare the ADCC timer count with the event time specified in the ADCC_EVT[nn].TIME bit field of the enabled event. If the time value matches, the comparators signal the timing and control unit, indicating that an event has become active and is ready to handle. In response, the timing and control unit starts the ADC sampling sequence (if the ADC interface is available).

If more than one event is active during the same SYSCLK cycle, the ADCC processes only the highest priority event. (The event in this case is associated with an ADCC timer and is assigned to the same ADC interface). All other events are missed (even if there was space available in the pending event FIFO). When an event gets missed, the corresponding event miss bit is set in the ADCC_EMISS register.

The priority of events is fixed, with the event with the lowest event ID having higher priority than other compared events. The priority of the ADCC_EVCTL[nn] registers from highest to lowest is ADCC_EVT0>ADCC_EVT1>…..>ADCC_EVT22>ADCC_EVT23. The events assigned to TMR0 have higher priority than events assigned to TMR1. If the events from both ADCC timers to same ADC interface become active in the same SYSCLK cycles, the event triggered by TMR0 gets higher priority.

Pending Event FIFO

The ADCC provides a separate, 8-deep FIFO for each ADC interface. If an event becomes ready when another event is ongoing on the same ADC interface, the occurred event is stored in the pending event FIFO. The stored event is an event *in collision*, and the corresponding event collision bit is set in the ADCC_ECOL register.

If the pending event FIFO is full when an event becomes active, the event is missed, and the corresponding event miss bit is set in the ADCC_EMISS register.

After the ADC interface unit is free to start the new ADC sampling sequence, the event from the FIFO is shifted to the timing and control unit.

On disabling the ADCC, all pending events in the pending event FIFO are flushed.

Timing and Control Unit

The timing and control unit of ADCC provides two ADC interfaces. Each interface can independently control one ADC with required timing and interface protocol.

When an event becomes ready to handle, the ADC interface unit initiates the ADC sampling sequence, according to the ADCC settings for that particular event. The ADC sampling sequence is divided into phases:

- Control phase - the control word is sent to ADC
- Conversion phase - the conversion pulse is provided
- Data phase - the converted data from ADC is read

The ADC interface provides an ADC clock, an ADC chip select, control lines for sending the control word, and data lines for reading converted data.

The ADCC timing registers determine the timing of the ADCC_ACLK, $\overline{\text{ADCC\_ACS}}$, ADCC_BCLK, and $\overline{\text{ADCC\_BCS}}$ signals. The ADC clock signals are gated and are provided only while controlling the ADC. The chip select signal has programmable timing in terms of the ADC clock. The ADCC uses the bits of the ADCC_CTL register to determine the timings and protocol of ADC control lines and data lines.

ADCC Operating Modes

The operating modes of the ADCC include:

- Data Transfer Modes
- Dual-Bit (Two Signal Line) Interface Mode
- Dual-Bit Interface Data Swap Mode
- Clock Modes
- Chip Select Modes
- Simultaneous Sampling Mode

Data Transfer Modes

The ADC interface of ADCC has a data acquisition capability in which the converted data from ADC is directly stored into the data register of the corresponding event. The event setup instructs the ADCC as to the ADC to use and the channel to sample. When the ADC interface handles the events, it initiates the ADC sampling sequence by providing the control word and conversion pulse to ADC. Then, the ADCC reads the converted ADC data and collects the data into the data register of that event, `ADCC_EVDAT[nn]`.

The ADCC can read the newly received data in the `ADCC_EVSTAT[nn]` register either in core mode or in DMA mode, which permits storing the data in the required memory space. The ADCC supports the following methods for reading ADC samples:

- Core-driven single data reception for each ADC sample
- DMA-driven data read for multiple words transfers with minimal intervention of core.

The `ADCC_CTL.DMAEN` enables the DMA-driven mode of ADCC operation. When disabled, the ADCC uses core-driven mode. Core-driven transfers use ADCC data interrupts to signal the processor core to perform single word read from the event data register, which is ready. DMA transfers can be set up to transfer a configurable number of ADC samples to internal or external memory of processor without core intervention.

For more information, see Core-Driven Data Read Mode and DMA-Driven Data Read Mode.

Core-Driven Data Read Mode

The ADCC provides 24 event data registers (one for each ADCC event) for storing sampled ADC data associated with that event. These registers, `ADCC_EVDAT[nn]`, are accessible in core mode. Typically, when an ADC sampling sequence completes, the newly available read data is stored in the data register of that event. The ADCC signals the data ready interrupt to the core. In the interrupt service routine, the core checks the ADCC event interrupt status register, `ADCC_EISTAT`, to determine which event is complete and reads the ADC sample from its data register.

The ADCC event interrupt mask register, `ADCC_EIMSK`, provides bits to unmask (enable) or mask (disable) the data read interrupt requests for individual events. If there are multiple events in an ADCC frame, to reduce the interrupt requests for every event in that frame, the ADCC provides a frame completion interrupt. The interrupt is triggered once in a frame after completion of all the events. The individual data requests from events can be masked, then their data can be read based on the ADCC timer frame completion interrupt.

In core mode, the “end of the ADCC timer frame” is when all the event data related to that ADCC timer has been received from the ADC. The ADCC timer stops at the end of frame and is reset to zero.

DMA-Driven Data Read Mode

The ADCC includes two built-in DMA units for reading the ADC samples received in event data registers. The controller provides a 16-bit DMA interface to store these ADC samples directly in processor memory without core intervention. One DMA unit handles data read requests for the events associated with TMR0, and the other DMA unit handles data requests for the events associated with TMR1.

Each DMA unit provides a set of registers to configure the DMA work-unit:

- Base pointer register (for example, ADCC_BPTR0) - this register contains the base address of memory space for storing ADC samples.
- Frame increment register (for example, ADCC_FRINC0) - this register contains the address increment applicable to the DMA base pointer register between the ADCC timer frames.
- Circular buffer size register (for example,ADCC_CBSIZ0) - this register holds number of ADCC timer frames to be received.

Apart from these registers, the DMA unit uses the event offset field of control register of the respective event to store the ADC samples related to it. The ADCC_EVCTL[nn].EVTOFS field specifies the memory offset in the frame block defined by the base pointer and frame increment registers for each ADCC frame.

Each ADC timer frame can consist of many ADCC events. The ADCC_BPTR0 or ADCC_BPTR1 registers hold the base-address used for placing the event data of the first frame into memory. The event data are each placed in a location calculated from the base pointer and the event offset (for example, ADCC_BPTR0 + ADCC_EVCTL[nn].EVTOFS). The base address of the second frame is calculated from the base pointer and the frame increment (for example, ADCC_BPTR0 + ADCC_FRINC0). The base address of the third frame includes a multiple of the frame increment (for example, ADCC_BPTR0 + 2 *ADCC_FRINC0). This *linear buffering mode* calculation pattern continues through the series of frames. The calculation of the address for placing the received data of an event in the n^{th} frame applies all of these factors. For example:

Location = ADCC_BPTR0 + (n-1) *ADCC_FRINC0 + ADCC_EVCTL[nn].EVTOFS

When the circular buffer size register (for example, ADCC_CBSIZ0) is programmed to zero, the DMA operates (as previously described) in linear buffering mode.

When the circular buffer size register is programmed to a non-zero value, the DMA operates in *circular buffering mode*. In this mode, the DMA initially follows the same pattern as given for the linear buffering. But, after a specified number of frames, the frame base pointer is reset to the base pointer address (for example, ADCC_BPTR0). It is reset to the address instead of calculating the value from the base pointer plus frame increment. The ADCC uses the circular buffer size register (for example, ADCC_CBSIZ0) to set the number of frames after which circular buffer wraparound (resetting to the base pointer) occurs. After receiving the selected number of frames, the circular buffer wraparound occurs, and the DMA begins overwriting the data from previous frames.

NOTE: Circular buffer wraparound is sometimes referred to as *circular buffer loopback*.

Considering operation in both DMA modes, the base pointer register holds the address for the following frames:

- The first frame after the ADCC is enabled (both DMA modes)
- The first frame after the core writes to the base pointer register (in linear buffering mode)
- The first frame after a circular buffer wraparound occurs (in circular buffering mode)

NOTE: There is no DMA burst feature, and each event data is written separately after completion of all the events in that frame.

In DMA mode, the end of an ADCC timer frame occurs when all the event data related to that ADCC timer is written into memory and responses have been received. The ADCC timer stops at the end of frame and is reset to zero. When there are multiple pending events to write into memory, the priority is from the highest priority for event 0 to the lowest for event 23.

Because the ADC data is 16 bits wide, bit 0 of the base pointer register and the frame increment register is non-writable and reads as 0. This read-no-write access for bit 0 is to avoid address alignment error, by ensuring 16-bit alignment. Do not to access the event data registers directly while in DMA mode.

If the ADCC is disabled while a DMA transfer is in progress, the DMA of ADCC completes (gracefully). It writes all data received (up to the point the ADCC was disabled) into memory. The DMA pending status for the corresponding timer (for example, `ADCC_T0STAT.DPND`) remains high until all such transactions complete on the DMA bus, including the write response signaling.

DMA Bandwidth Monitoring

In DMA mode, ADCC provides a DMA bandwidth monitoring feature, which permits identifying whether the ADCC timer count has gone beyond an expected limit while completing a frame. The ADCC provides a bandwidth monitor register (for example, `ADCC_BWMON0`) for each ADCC timer to use this feature for their frames.

For example, consider a case where each frame of an ADCC timer can complete much earlier than *Q* number of SYSCLK cycles after receiving the trigger. The program can use the DMA bandwidth monitoring feature to check whether all the events and their DMA transfers in each ADCC timer frame complete within that time. Program the register of the related ADCC timer with a count of *Q* in this case.

The ADCC raises the error interrupt when the corresponding ADCC timer crosses this count while completing a frame. The ADCC timers count until the DMA transfer of all events completes.

If bandwidth monitoring feature is not needed, program the bandwidth monitor register to zero, disabling this feature.

Dual-Bit (Two Signal Line) Interface Mode

The ADCC offers dual-bit interface functionality. It provides two lines (for example, `ADCC_AD0` and `ADCC_AD1`) for sending the control word serially to the ADC and two lines for reading serially the converted data from the ADC. These two lines do not operate separately, but the bit stream is serially interleaved on them.

Although the ADCC provides two lines for sending the control word and receiving the data bits, the ADC interface unit can be configured to use a single line (single-bit interface). Or, the ADC interface unit can be configured to use both lines (dual-bit interface), based on the interfaced protocol the ADC supports. The number of control lines is selected using the `ADCC_CTL.CSIZE` bit, and the number of data lines is selected using the `ADCC_CTL.DSIZE` bit.

Dual-Bit Interface Data Swap Mode

When using dual-bit interface mode (two signal lines), the first bit starts on either data signal line (for example, `ADCC_AD0` and `ADCC_AD1`) based on the setting of the data swap bit (for example, `ADCC_CTL.DSWP0`). Also, the ADCC can read or send data in either LSB-first or MSB-first format depending on the LSB first bit (for example,

ADCC_CTL.LSBF0). Refer to the ***Dual Bit Swap Disable*** table and the ***Dual Bit Swap Enable*** table. The tables show how the ADCC reads the ADC data bits on two data signal lines for different combination of the data swap bit and LSB first bit. The same bit pattern applies for the control signal lines (for example, ADCC_ACTL0ADCC_AD0when sending control bits.

NOTE: The ADCC control register provides separate versions of these bits for both ADC interfaces. The ADC interfaces can be independently configured for the protocol (data swap or choice of first bit) supported by the ADC connected to it.

Table 26-8: Dual Bit Swap Disable

Clock Cycle	ADCC_CTLx.DSWP = 0			
	ADCC_CTLx.LSBF= 0		ADCC_CTLx.LSBF = 1	
	MSB First Format		LSB First Format	
	Bits on AD1	Bits on AD0	Bits on AD1	Bits on AD0
1	Data 15	Data 14	Data 1	Data 0
2	Data 13	Data 12	Data 3	Data 2
3	Data 11	Data 10	Data 5	Data 4
4	Data 9	Data 8	Data 7	Data 6
5	Data 7	Data 6	Data 9	Data 8
6	Data 5	Data 4	Data 11	Data 10
7	Data 3	Data 2	Data 13	Data 12
8	Data 1	Data 0	Data 15	Data 14

Table 26-9: Dual Bit Swap Enable

Clock Cycle	ADCC_CTLx.DSWP = 1			
	ADCC_CTLx.LSBF= 0		ADCC_CTLx.LSBF = 1	
	MSB First Format		LSB First Format	
	Bits on AD1	Bits on AD0	Bits on AD1	Bits on AD0
1	Data 14	Data 15	Data 0	Data 1
2	Data 12	Data 13	Data 2	Data 3
3	Data 10	Data 11	Data 4	Data 5
4	Data 8	Data 9	Data 6	Data 7
5	Data 6	Data 7	Data 8	Data 9
6	Data 4	Data 5	Data 10	Data 11
7	Data 2	Data 3	Data 12	Data 13

Table 26-9: Dual Bit Swap Enable (Continued)

Clock Cycle	ADCC_CTLx.DSWP = 1			
	ADCC_CTLx.LSBF= 0		ADCC_CTLx.LSBF = 1	
	MSB First Format		LSB First Format	
	Bits on AD1	Bits on AD0	Bits on AD1	Bits on AD0
8	Data 0	Data 1	Data 14	Data 15

Clock Modes

The ADCC provides a clock signal to communicate with the interfaced ADC. Some of the ADCs can use this signal for the conversion process. The ADC clock is in gated format (for example, it is active only during the phases of ADC sampling sequence) to ensure excellent noise immunity during the conversion process.

Clock Frequency programming

The ADC clock is internally generated from system clock of processor. The `ADCC_TCA0.CKDIV` bit field specifies the divider to generate ADC0 clock signal from SYSCLK.

```
ADCC_ACLK = (SYSCLK) / (ADCC_TCA0.CKDIV + 1)
```

Alternatively, the clock divisor value for the required ADC0 clock frequency is calculated as:

```
ADCC_TCA0.CKDIV = (SYSCLK ÷ ADCC_ACLK) – 1
```

Falling Edge Clock Edge Programming (for example, `ADCC_CTL.CKPOL0` =0)

The ADC interface drives the control word bits and samples the data bits based on active edges of clock. The active edge of the ADC clock edge is selected with the clock polarity bit.

In this case, the first ADC clock edge driven, after chip-select becomes active, is a falling edge. The clock signal is at the high level when the ADC interface is not busy.

The ADC interface drives the control word bits on the ADC control lines on the rising edges of the ADC clock. But, it drives the first bit prior to the first falling edge given to ADC. The ADC samples on the falling edges of the ADC clock.

The ADC drives the bits of the converted data on the falling edge of the ADC clock. The ADC interface of ADCC samples these data bits on the next falling edge, allowing a full cycle of operation for high-speed read operations. The ADC drives the first bit before the chip select becomes active.

The ***Clock Edge Programmed for CKPOL =0*** figure shows how the ADC signal transitions occur for this case.

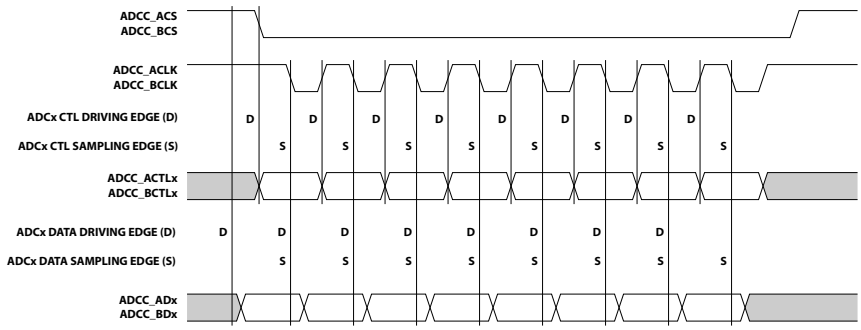

Figure 26-8: Clock Edge Programmed for CKPOL =0

Rising Edge Clock Edge Programming (for example, `ADCC_CTL.CKPOL0` =1)

The ADC drives the control word bits and samples the data bits based on active edges of clock. The active edge of the ADC clock edge is selected with the clock polarity bit.

In this case, the first ADC clock edge driven after clock select becomes active is a rising edge. The clock signal is at the low level when the ADC interface is not busy.

The ADC interface drives the control word bits on the ADC control lines on the falling edges of the ADC clock. But, it drives the first bit prior to the first rising edge given to ADC. The ADC samples on the rising edges of the ADC clock.

The ADC drives the bits of the converted data on the rising edge of the ADC clock. The ADC interface of ADCC samples these data bits on the next rising edge, allowing a full cycle of operation for high-speed read operations. The ADC drives the first bit before the chip select becomes active.

The ***Clock Edge Programmed for CKPOL =1*** figure shows how the ADC signal transitions occur for this case.

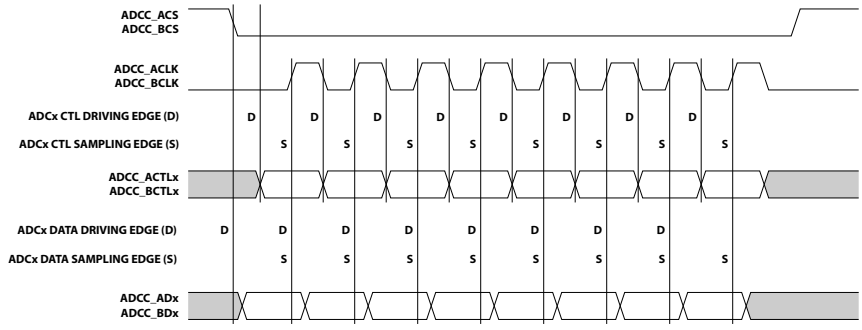

Figure 26-9: Clock Edge Programmed for CKPOL =1

Chip Select Modes

The ADCC provides a chip select signal (for example,) to select the ADC for communication and to signal (optionally with its edges) the start of conversion for the ADC. The chip select is asserted during the phases of the ADC sampling sequence (control, conversion, and data).

The chip select signal provided to ADC can be configured as an active-high signal or an active-low signal, based on the protocol the interfaced ADC supports. This sensitivity is configurable using the chip select polarity bit (for example, ADCC_CTL.CSPOL0).

During the conversion phase or data-read phase of the ADC sampling sequence, a valid control word is not always driven to the ADC. The ADC control lines sometimes must be driven to an inactive level (either a high or a low level) to meet an ADC requirement.

The ADC interface of the ADCC provides the option to select the state to drive control lines while idle based on the transmit idle bit (for example, ADCC_CTL.TIDLE0).

The ***Chip Select Idle Programming*** figure shows an example in which the chip select polarity is selected as an active-low signal, and the transmit idle bit is cleared. This example shows the chip select as held low when not transmitting a valid control word to the ADC.

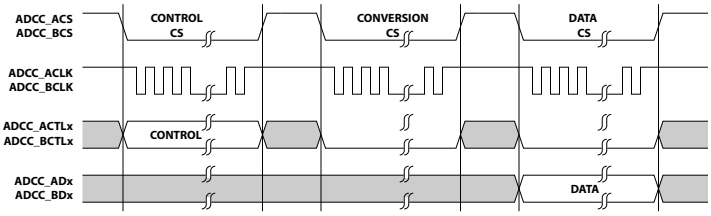

Figure 26-10: Chip Select Idle Programming

The width and period of the chip select signal is configurable based on settings of timing control registers (for example, ADCC_TCA0 and ADCC_TCB0). The width of the chip select signal is configurable based on the number-of-clocks bit field, the time-CS-to-CK-setup bit field, and the time-CK-to-CS-hold bit field, where:

- Time-CS-to-CK-setup (for example,ADCC_TCB0.TCSCK) is the minimum delay between the assertion edge of the chip select and the first edge of the ADC clock
- Number-of-clocks (for example, ADCC_TCA0.NCK) is the number of ADC clock cycles to be given in each chip select pulse.
- Time-CK-to-CS-hold (for example, ADCC_TCB0.TCKCS) is the minimum delay between last edge of the ADC clock and the de-assertion edge of the chip select signal
- Time-CS-to-CS-delay (for example, ADCC_TCB0.TCSCS) is the minimum delay between the completion of one phase and starting of another phase of the ADC sampling sequence. (For example, the delay is the time between the de-assertion edge of one chip select signal to the assertion edge of the next chip select signal).

All of these parameters are specified in terms of ADC clock cycles. The ***Chip Select Signal Description*** figure illustrates the timing relationships and parameters.

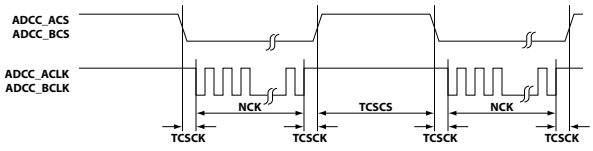

Figure 26-11: Chip Select Signal Description

From these parameters, the active duration of the chip select signal can be calculated (for ADC0) as:

Active Duration CS = (ADCC_TCB0.TCSCK + ADCC_TCA0.NCK + ADCC_TCB0.TCKCS)

And, the period of the chip select signal can be calculated (for ADC0) as:

Period CS = (*Active Duration CS* + ADCC_TCB0.TCSCS)

Simultaneous Sampling Mode

The simultaneous sampling feature permits taking samples from two ADC channels, one from ADC0 and another from ADC1, in a synchronous fashion.

This feature is useful in situations where the application must analyze the two different analog signals, and both signals must be sampled at the same time. An ADC interface handles events sequentially. If the two signals connect to two different channels of same ADC, it is not possible to sample them simultaneously. If these two signals connect to different ADCs, the ADC interfaces (which are independent) cannot be synchronized while taking samples from the two signals. For example, the two ADC interfaces can initiate the sampling sequence based on their own activities, independent of each other.

To address this common requirement, the simultaneous sampling feature ensures that the samples on two ADC channels are taken simultaneously, using a shared chip select. (For example, one sample is from ADC0 and another from ADC1). The simultaneous sampling sequence on both ADCs is initiated only after all the pending events are handled on the respective ADC interface. This functionality ensures that the two ADC interfaces are synchronized.

To take simultaneous samples from the ADC0 channel and the ADC1 channel, configure the two events with following settings in their event control register:

- Set the simultaneous sampling enable bit

  (ADCC_EVCTL[nn].SIMSAMP =1 for both)

- Set the share chip select bit in the ADC control word

  (ADCC_EVCTL[nn].CTLWD bit 1 =1 for both)

  For more information about the ADC control word, see ADCC Programming Guidelines.

- Assign both events to the same ADCC timer

  (ADCC_EVCTL[nn].TMRSEL bit is the same for both)

- Select the same event time for both events

  (ADCC_EVT[nn] value is the same for both).

- Assign each event to a different ADC interface

  (ADCC_EVCTL[nn].ADCSEL bit must be different for each)

- Use identical settings in the timing control registers for both ADC interfaces

  (ADCC_TCA0 == ADCC_TCA1, ADCC_TCB0 ==ADCC_TCB1)

Program simultaneous events in pairs. When there are pending events, and a simultaneous sampling pair is encountered, the pair is executed synchronously only after both ADCs become available for accepting the new control word.

The ***Simultaneous Sampling with Two Events*** figure shows a case of simultaneous sampling. Event 0 is configured to initiate sampling of channel-m of ADC0, and event 1 is configured to initiate sampling of channel-n of ADC1.

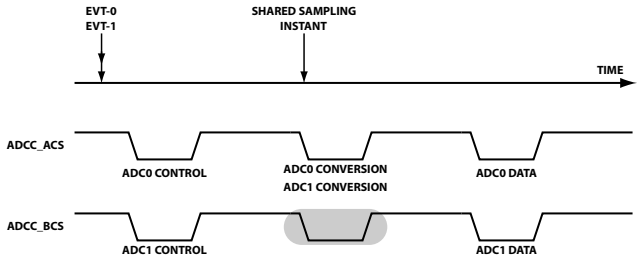

Figure 26-12: Simultaneous Sampling with Two Events

Event 0 and event 1 are triggered at the same time, because both events are assigned to same ADCC timer and the event time registers of both events are same. As there are no pending events on both ADC interfaces, the ADCC immediately chooses to start the ADC sampling sequence on both ADC interfaces synchronously.

If pending events are present on either of the ADCs, simultaneous sampling happens only after both ADCs are available. The ***Simultaneous Sample with Many Events*** figure shows a case in which events 0 and 1 sample on the ADC0 channels. Events 2, 3, 4, 5, and 10 sample on the ADC1 channels such that event 1 and event 10 are a simultaneous sampling event pair. Assume that the ADC0 interface has only event 0 before simultaneous sampling; while the ADC1 interface has events 2, 3, 4, and 5 before simultaneous sampling. This case demonstrates the wait implemented by the ADC interface to manage pending events until it processes the simultaneous sampling pair.

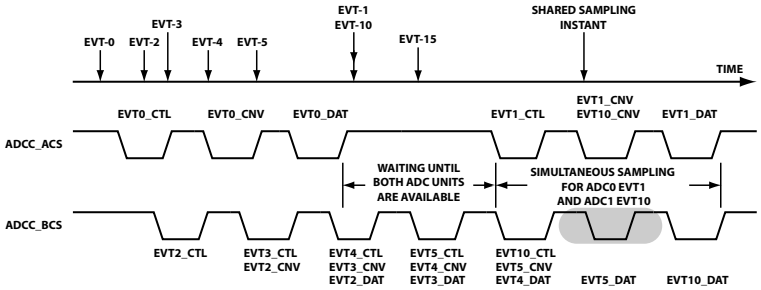

Figure 26-13: Simultaneous Sample with Many Events

From the figure, it is important to observe that when the simultaneous sampling event pair becomes active, the ADC0 interface is idle. The interface does not initiate the sampling sequence for event 1 associated with it. The ADC1 interface is busy servicing pending events activated before the paired event (event-10) is ready. The ADC0 interface waits until the ADC1 interface becomes available to handle the paired event. Also, note that the ADC interface treats simultaneous sampling event pairs as individual (non-paired or normal) events based on prioritization. There is no priority assigned to paired events over normal events. Due to the inherent nature of the ADC interface to handle the events in serial, the event triggered first is handled first. The ADC interface handles subsequent triggered events only after all other triggered pending events. If another event (for example, event 15) is triggered on

ADC0 interface, the event is stored in pending event FIFO, even if ADC0 interface is not handling any event. Event 15 is handled only after execution of the pending events triggered before it (for example, after executing event 1).

Refer to the ***Simultaneous Sampling with Two Events*** figure and the ***Simultaneous Sample with Many Events*** figure. While executing simultaneous sampling events, the conversion pulse to ADC1 in the ADC sampling sequence is not given from ADC1 interface, but is shared from the ADC0 interface.

ADCC Event Control (SEC and TRU Related)

For the ADCC, the term "event" typically refers to ADC-related sampling and data conversion. The processor considers "events" as interrupts (related to the SEC) and triggers (related to the TRU).

The ADCC provides status and error bits through different registers to signal the core about its state and various error conditions that occurred during its operation.

The ADCC provides a primary error status register (`ADCC_ERRSTAT`) and two supplementary error status registers (`ADCC_ECOL` and `ADCC_EMISS`). The supplementary registers provide more information about the errors flagged in `ADCC_ERRSTAT` register. Optionally, the ADCC can generate an error interrupt based on these conditions.

Also, the ADCC provides an event completion status register (`ADCC_EISTAT`), which is used in core mode to service the ADC data read requests. A frame interrupt status register (`ADCC_FISTAT`) provides timer frame completion status, which indicates whether all the events related to a frame have completed successfully. Optionally, the ADCC data interrupt can be generated based on these conditions.

NOTE: The frame completion status bits must be acknowledged after completion of the ADCC frame. Otherwise, the incoming trigger is ignored, and a trigger overrun error is flagged.

The ADCC provides a timer status register (for example, `ADCC_T0STAT`) for each ADCC timer. This status register indicates the current frame number the ADCC is handling. Also, this register indicates whether DMA is pending reception of data from the ADC. An event pending register (`ADCC_EPND`) indicates which events are yet to start after the trigger.

More information about ADCC features relating to processor events (SEC interrupts and TRU triggers) is available in the following sections:

- Interrupt Status
- Error Status
- Pending, Frame, and Delay Status
- Event Handling Latency

Interrupt Status

The ADCC provides a data interrupt channel for each ADCC timer. `ADCC_TMR0_EVT` carries the TMR0 related interrupts, and `ADCC_TMR1_EVT` carries the TMR1 related interrupts. The ADCC uses the same interrupt channel whether executing data reads in core mode or in DMA mode.

The frame completion status bits for each ADCC timer (ADCC_FISTAT.FINT0 and ADCC_FISTAT.FINT1) are applicable when using either core or DMA mode. In core mode, this bit is set when data corresponding to all events (which were not missed) in a frame are received from the ADC. In DMA mode, this bit is set when:

- the data corresponding to all the events of a frame are received
- the data is DMA-transferred into processor memory, and
- the memory write responses are successfully received from the SCB

Clear the frame completion status bits to acknowledge before the next trigger appears. Otherwise, the trigger is ignored, and the trigger overrun error condition is flagged. Clear these sticky status bits with a W1C operation.

The ADCC data interrupt can be generated on completion of each frame. The frame interrupt mask register (ADCC_FIMSK) unmasks (enables) or masks (disables) the frame completion interrupt from each ADCC timer.

When operating in core mode, the ADCC uses the same data interrupt to service the data read requests from each event. The event completion bit (ADCC_EISTAT.EVT[nn]) indicates that data for the corresponding event is pending and ready for the core to read. Based on the selection in the timers select bit (ADCC_EVCTL[nn].TMRSEL) of the event, the corresponding event interrupts are sent on either the ADCC_TMR0_EVT or on the ADCC_TMR1_EVT channel. Optionally, this interrupt from each event can be individually masked off from being signaled, using the bits in ADCC_EIMSK. The event data ready bits in ADCC_EISTAT register are sticky. Clear these bits with a W1C operation.

Error Status

The ADCC can signal error conditions during its operation. It reports these conditions in an error status register (ADCC_ERRSTAT), which holds the error status for the following:

- Trigger overrun error
- Event miss in frame error
- Event collision in frame error
- DMA bandwidth monitoring error
- Memory write response error

The trigger overrun error, DMA bandwidth monitoring error, and memory write response errors for ADCC timers are individually flagged through separate bits.

The event miss bit is flagged when any one of the events are missed during the ADCC frame. The ADCC provides a separate event miss register (ADCC_EMISS), which indicates which event was missed.

Similarly, the event collision bit is flagged when any one of the events collided with an active event. For example, this flagging occurs when an event becomes active while the ADC interface is already handling another event. The collided event is placed in pending event FIFO, and its handling is delayed. The ADCC provides a separate event collision register (ADCC_ECOL), which indicates which event collided.

The ADCC provides an error interrupt channel (ADCC_ERR) to signal these error conditions to the core. This interrupt carries error-related interrupts for both ADCC timers and for both ADC interfaces. By default, all the error conditions generate interrupts. But, the error conditions can be individually masked from appearing on the ADCC_ERR signal by programming the error mask register (ADCC_ERRMSK).

The following sections provide additional information about the most common ADCC error conditions.

Trigger Overrun Error Status

Trigger overrun is a condition in which the ADCC timer detects a valid trigger edge on the selected trigger input before a previously triggered frame completes.

The ADCC timer starts counting when it detects a valid edge on its trigger input. Then, the ADCC handles the events associated with the trigger by initiating ADC sampling sequences at the required time offsets. After all the events complete, the frame is completed, and the ADCC sets the frame completion status bit (for example, ADCC_FISTAT.FINT0). In DMA mode, the frame completion status bit is set when:

- the data corresponding to all the events of a frame are received
- then DMA is transferred into processor memory, and
- the successful memory write responses is received from the SCB.

The frame completion conditions must be acknowledged by clearing this bit in the software. After the bit is cleared, the ADCC timer is ready to accept new trigger, initiating a new frame. If the new trigger pulse is received before this time, it is treated as premature trigger, and the trigger overrun condition is flagged.

When this condition occurs, the ADCC ignores this trigger, sets the respective trigger overrun status bit (for example, ADCC_ERRSTAT.TRGOV0), generates the error interrupt (if enabled), and continues its operation normally.

A trigger overrun condition also occurs when a new trigger is detected while ADC programming is in-progress (indicated by the ADCC_CFG.PND bit). This case can be avoided if the timer trigger enable (for example, the ADCC_CTL.TRGIE0 bit) is disabled before ADC programming then re-enabled at the end of ADC programming.

Event Miss Error Status

An ADCC event is considered missed when the ADC sampling sequence corresponding to that event is not executing on the required ADC interface. When an event is missed, the ADCC_ERRSTAT.EMIS bit is flagged and in the ADCC_EMISS.EVT[nn] bit corresponding to that event is set to provide more information to the application.

In the following scenarios, a programmed event can be missed:

- *FIFO overrun*

  An event is missed if the event pending FIFO is full when the event match occurs.

- *Event time conflict*

  One or more events is missed if:

  - two or more events requiring sampling at the same ADC interface are assigned to same ADCC timer, and
  - the events have identical event time values in their event time registers.

  Only the higher priority event is forwarded, and the lower priority events are missed.

- *Non-paired simultaneous sampling*

  An event is missed if it is enabled for simultaneous sampling (`ADCC_EVCTL[nn].SIMSAMP` bit) on one ADC interface, but there is no paired simultaneous sampling event programmed for the other ADC interface.

- *ADC interface conflict for simultaneous sampling pair*

  An event is missed if both events in a simultaneous sampling event pair are configured for same the ADC interface. The event pair in this case is considered as two normal events (not a simultaneously sampled pair), and the lower priority event is missed.

- *Missed half of simultaneous sampling pair*

  Both events are missed if one event in a simultaneous sampling pair is missed at one ADC interface due to any event miss reasons previously mentioned.

The priority of event is based on its event number ID. For example, the event with lowest event ID has higher priority compared to other events, such that EVT0>EVT1>…..>EVT22>EVT23. The priority does not depend on whether it is assigned to the ADC0 interface or to the ADC1 interface. The priority does depend on whether the event is assigned to TMR0 or assigned to TMR1. The events assigned to TMR0 have higher priority than events assigned to TMR1.

This priority is checked only at the event comparator block, when more than two events become active at the same SYSCLK cycle. After the events are queued, the ADC interface does not check the priority, and events are handled in sequence as they become active.

As mentioned in *event time conflict*, low priority events are missed when two or more events are assigned to same ADCC timer with identical event time register value. (Events are assigned to same ADC interface). Consider a case in which two events (EVT1 and EVT5) occur simultaneously.

- If both events are assigned to same ADCC timer and route to the same ADC interface (ADC0 or ADC1):
  - the EVT5 event (being the lower priority event) is missed, and
  - the EVT1 event is forwarded to the ADC interface for handling.

Do not assign the same event time for multiple normal events of same ADCC timer and send them to the same ADC interface. It is the responsibility of the programmer to ensure that the values in the event time registers do not lead to event misses.

- If one of the events is assigned to TMR0 and the other to TMR1 (both route to same ADC interface), the event assigned to TMR0 is forwarded first to the ADC interface. Then, the event assigned to TMR1 is forwarded. Typically, the event is stored in the event pending FIFO when the FIFO is not full. This case appears when sources that are not synchronized trigger TMR0 and TMR1. And, the event time register is not always the same for both events. It is important to consider the possibility of events occurring either simultaneously or being missed when enabling events on two asynchronously triggered timers. Sufficient time must space the events, so the pending event FIFO does not operate at its maximum capacity.
- If both events are assigned to the same ADCC timer, both events are forwarded to the respective ADC interfaces for handling, assuming there is no pending event. (In this case, one event routes to the ADC0 interface and other to ADC1 interface). A simultaneous event pair is assigned to same ADCC timer with same event time but going to different ADC interfaces. If one of the events is missed due to some reason, the paired event also is missed. (For example, the pending event FIFO on the particular ADC interface is full or if another high priority event from same timer becomes active in the same SYSCLK cycle)

NOTE: To reduce the event miss condition resulting from pending event FIFO full conditions, the ADCC provides an 8-deep event FIFO for each ADC interface. But, the programmer must ensure that the events are sufficiently spaced apart from each other (event handling in pipelined manner can also be considered). The programmer must ensure that event miss conditions do not lead to FIFO overrun.

Event Collision Error Status

If event times are not sufficiently spaced apart, an event could occur while a previous event is underway. An ADCC event is considered *in collision* if:

- that event becomes active (the event comparator unit signaled it as ready-to-handle) when the ADC interface is busy handling a previous event, or
- there are already some events in the pending event FIFO waiting to be handled

In such cases, the newly active event cannot be issued to the ADC at the earliest possible time because of some previous events are not complete yet.

The ADCC provides an 8-deep FIFO for queuing collided events. The `ADCC_ERRSTAT.ECOL` bit flags the collision condition, and the `ADCC_ECOL.EVT[nn]` bit corresponding to that event is set to provide more information to the application. The ADCC also provides a means for the application to know (by how many SYSCLK cycles) the event was delayed. It indicates this time in the `ADCC_EVSTAT[nn].DLYCNT` field of that event, which is updated only when the control word for the event is sent to the ADC.

If one event of a simultaneous sampling pair undergoes collision, the other event (in the simultaneous sampling pair) also is indicated as undergoing collision.

Due to inherent nature of ADCC to queue the ready events, the collided event can be handled before completion of all the pending events. For example, this previous event (for instance, $(n-1)^{th}$) is completed only after the reception of its data. When a collision occurs, it indicates that the n^{th} event is sent to the ADC only after some delay. But, it is not after the completion of $(n-1)^{th}$ event. When a chip select pulse of the conversion phase or data phase of $(n-1)^{th}$ word must be sent to the ADC, the ADCC state-machine sends the control word for n^{th} event with the earliest chip select pulse.

NOTE: The event collision is an error condition in which the expected sampling time of the ADC for the collided event can be delayed. The event is not missed.

DMA Bandwidth Monitoring Error Status

The ADCC can be configured to store the ADC samples related to enabled events directly in the processor memory through the built-in DMA unit. When using this mode, the ADCC provides a DMA bandwidth monitoring feature, which identifies whether the ADCC timer count has gone beyond an expected limit while completing its frame. The bandwidth monitor register (for example, `ADCC_BWMON0`) specifies this count.

When the ADCC timer count crosses the count specified in bandwidth monitor register while completing a frame, the bandwidth error bit flags the DMA bandwidth error for the respective DMA unit. (An example of the bandwidth error bit is `ADCC_ERRSTAT.BWERR0`).Clear this sticky error bit with a W1C operation. Optionally, the ADCC can signal to the core through an error interrupt this error condition.

Memory Write Response Error Status

If the ADCC detects a memory write error response that corresponds to a data write issued for an ADCC timer event, the memory response error bit flags the condition. (An example of the memory response error bit is `ADCC_ERRSTAT.MERR0`). Clear this sticky error bit with a W1C operation. Optionally, the ADCC can signal to the core through an error interrupt this error conditiont.

Pending, Frame, and Delay Status

In addition to data request status and error bits, the ADCC provides bits for DMA pending status, event pending status, current frame number, and event delay status. These conditions operate as follows:

DMA Pending Status

If the ADCC is disabled while a DMA transaction is in-process, the DMA finishes (gracefully). It does not shut down until the data corresponding to all completed events is sent to memory and responses are received. The DMA pending bit (for example, `ADCC_T0STAT.DPND`) indicates the DMA activity associated with an ADCC timer. This status bit is useful for cases in which the ADCC is later re-enabled. The application typically polls this bit before re-enabling the ADC controller.

Event Pending Status

The ADCC provides an event pending register (ADCC_EPND) to indicate the events within the current frames that are pending execution. This register contains a dedicated bit corresponding to each ADCC event. At the trigger pulse, when a frame corresponding to an ADCC timer is initiated, all the bits corresponding to enabled events in that frame are set. The bits are cleared upon receiving the data from the ADC for the corresponding events. Because the bits in this register indicate whether the event is pending, if an event is missed, the respective bit also is cleared.

Current Frame Status (Number)

In DMA mode, the application can require knowledge of the number of frames the ADCC handled. The current frame count bit field (for example, ADCC_T0STAT.CURFR) of the associated ADCC timer provides this information. The field indicates the current frame number in chronological order after the ADCC was enabled (in linear buffer mode). Or, the field indicates the frame number after the last wraparound (in circular buffer mode). This field wraps over after 0xFFFF frames.

Event Delay Status

When an event is in collision, the application can need knowledge of the amount of time the event was delayed (in SYSCLK cycles). The ADCC uses the delay count bit field (ADCC_EVSTAT[nn].DLYCNT) of the event to indicate this information. The ADCC updates the field only when it sends the control word of the event to the ADC.

Event Handling Latency

The event handling latency of the ADC interface is the time between the internal occurrence of the event and start of the ADC sampling sequence for the event. The latency is the time between:

- the event time value matching the ADCC timer count value, and
- the assertion of chip-select for the control phase of the ADC sampling sequence

If an event becomes active when the ADC interface of ADCC is idle, the timing and control unit immediately starts handling that event. (The event has a predictable latency of 4–5 SYSCLK cycles).

If the event becomes active when the ADC interface is busy handling a previous sampling event, the new event is held in the pending event FIFO. The latency increases by the duration that the new event is held in the pending event FIFO. To reduce the latency in servicing this new event, the ADC interface can decide to queue the sampling sequence for this event with the sampling sequence of an ongoing event. For more information about this operation, see the ADCC Functional Description.

For a simultaneous sampling event pair, the latency can depend on the activities on other the ADC interface. This event pair is executed only when both ADC interfaces are ready.

Sometimes there can be few cycles of latency at the trigger routing unit (TRU) of processor when providing trigger inputs to the ADCC from different trigger master sources. This situation is especially prone to occur when an

asynchronous external trigger input is selected as the trigger input of ADCC. The time needed for synchronization to this external signal can lead to few SYSCLK cycles of delay.

The ADCC provides an event delay count status field (ADCC_EVSTAT[nn].DLYCNT) for each event to convey the latency that occurred in handling an event. This count indicates the number of system clock cycles the event was delayed, after the event match occurred. This field gets updated in every ADCC timer frame, when the ADC chip select signal goes active to transmit the control word to the ADC corresponding to that event.

ADCC Programming Concepts

There are general programming concepts for using the ADCC when it is present on any processor.

NOTE: There can also be processor-specific guidelines for programming the ADCC.

Fully configure the ADCC including trigger setup, ADCC events, and DMA programming (if necessary) before enabling the controller by setting ADCC_CTL.EN bit. Set the trigger input bits (for example, ADCC_CTL.TRGIE0) just before enabling the ADCC.

Do not configure two or more events assigned to the same timer and going to the same ADC interface with the same event time. If the ADCC has this configuration, other event miss error conditions can occur.

If different sources that are not synchronized trigger the ADCC timers, it is important to consider the possibility of events occurring either simultaneously or being missed. Space apart the events with sufficient time, so the pending event FIFO does not operate at (or beyond) its maximum capacity.

The frame completion status bit must be acknowledged for every frame. If they are not acknowledged, trigger overrun conditions can occur.

There is no restriction on dividing the events among ADCC timers and among ADC interfaces.

When configure simultaneous sampling event pair, certain restrictions apply in terms of configuring event attributes. See Simultaneous Sampling Mode for more details.

The event registers can be modified after the ADC controller is enabled; these registers include the event time (ADCC_EVT[nn]), event control (ADCC_EVCTL[nn]), and event enable (ADCC_EVTEN). Do not modify these registers when the frame is in-progress. If necessary, modify these registers only after completion of the frame associated with the ADCC timer to which the event is assigned. Modify them before clearing the frame completion interrupt bit. These registers must be stable when the trigger arrives.

The next trigger is accepted only after clearing the frame completion interrupt bit (for example, ADCC_FISTAT.FINT0).

In linear DMA mode, the base pointer register (for example, ADCC_BPTR0) and frame increment register (for example,ADCC_FRINC0) can be modified after enabling the ADC controller. Do not modify the base pointer register in circular buffer DMA mode. The changes in DMA registers are considered only in the next frame.

Do not disable the ADC controller while a frame is in-progress (for example, disable the ADCC only after the frame interrupt). This protocol prevents unfinished transfers on ADC interface. Before re-enabling the ADCC, poll the

DMA pending bit (for example, ADCC_T0STAT.DPND) to confirm that there is no DMA activity. Before reconfiguring the ADCC_CTL register, disable the ADCC. The only write allowed into an enabled ADCC control register is to disable it.

The ADCC status registers and counter-values are retained upon disabling ADCC (for debug purposes) and are cleared on re-enabling the controller (a 0-to-1 transition on the ADCC_CTL.EN bit). Error status bits are not cleared with this transition. Clear errors with a W1C operation.

ADCC Programming Model

The discussion in following sections provides the detailed programming model of the ADCC. If the application requires changing the ADC voltage reference (from the default) that step must be performed prior to any other ADCC programming.

ADC Voltage Reference (VREF) Selection

The internal ADCs in the ADSP-CM40x processors can be configured to select internally generated reference voltage or external reference voltage. By default ADCs select internally generated 2.5 V for voltage reference (VREF). In this case, these VREF signals are brought out on the VREF0 and VREF1 pins of processors to connect the required reference coupling capacitors. Do not load these pins externally by connecting to an external circuit or components. For characteristics of internal reference voltage, refer to the device data sheet.

The internal reference can be turned off and an external reference can be selected during analog subsystem initialization by programming the ADCC_CFG.REFSEL bit =1. The analog subsystem can be initiated by setting the ADCC_CTL.ADCFG bit =1.

The following pseudo code shows how to initialize analog subsystem to select external reference voltage:

NOTE: The Analog Subsystem Initialization process must be initiated at the start of application (before configuring ADCC module for actual ADC sampling process).

1. Initialize ADCC timing control registers.

```
ADCC_TCA0 = CKDIV_1 | NCK_8 ;

ADCC_TCB0 = TCSCK_1 | TCKCS_0 | TCSCS_10 ;

ADCC_TCA1 = CKDIV_1 | NCK_8 ;

ADCC_TCB1 = TCSCK_1 | TCKCS_0 | TCSCS_10 ;
```

2. Configure the ADCC_CFG register to select external reference selection.

```
ADCC0_CFG = REFSEL_1;
```

3. Enable ADCC using the ADCC_CTL.ADCFG bit set to start ADCC initialization.

```
ADCC0_CTL = ADCC_CTL_EN | ADCC_CTL_ADCFG | ADCC_CTL_DSIZE;
```

4. The ADCC configures the ADCs, setting the ADCC_CFG.PND bit while the configuration is in process. The ADCC clears this bit when the configuration is complete. Wait until ADC configuration is pending.

```
while((ADCC0_CFG & ADCC_CFG_PND) == 0); // wait for PND bit set then wait for
PND bit clear

while((ADCC0_CFG & ADCC_CFG_PND) == 1); // wait until ADCC operation is pending
```

5. Disable the ADCC.

```
ADCC0_CTL = 0x0000; // disable ADCC first

while((ADCC0_CFG & BITM_ADCC_CFG_PND) == 1); // wait until ADCC operation is
pending
```

Program Flow

The following steps provide and overview of the program flow for using the ADCC.

1. Write configuration data (including event control data) to the ADCC memory mapped registers.
2. Enable the ADCC (ADCC_CTL.EN =1).
3. The ADCC waits for trigger input.
4. A trigger input starts the ADCC timer at the time of an enabled event.

   Result: The ADCC transfers control words the ADC and receives data from the ADC.

   Info: When the data is ready, DMA transfers and the core can read data in parallel from the ADCC.
5. The ADCC generates a frame complete interrupt (for example, ADCC_FISTAT.FINT0 =1) when the data corresponding to all events in a frame are received.

   Info: With the frame complete, the ADCC can be re-configuring; if necessary, changing the base pointer (for example, ADCC_BPTR0).
6. Clear the frame complete interrupt (for example, ADCC_FISTAT.FINT0 =0).
   - If more frames are expected, the ADCC resumes waiting for trigger input.
   - If no more frames are expected, the ADCC can be disabled (ADCC_CTL.EN =0).

ADCC Programming Guidelines

There are general programming concepts for using the ADCC when it is present on any processor.

The processor includes two 16-bit on-chip internal ADCs. For control, the ADCC provides two ADC interfaces, which can be independently configured for the ADC sampling sequence timing.

The ADC expects an 8-bit control word on a single control line (for example,), while the controller drives the converted 16-bit data on two data lines (for example, and).

The ADC timing can appear as shown in the ***ADCC Timing Guide for Programming*** figure.

- Program the width of the ADC control interface to a 1-bit (single bit) interface (for example,`ADCC_CTL.CSIZE` =0).
- Program the width of ADC data interface to 2-bit (dual bit) interface (for example, `ADCC_CTL.DSIZE` =1).
- With these settings, the number of ADC clocks cycles required in a chip select pulse is eight (for example, `ADCC_TCA0.NCK` =8).
- The chip select polarity is active low (for example, `ADCC_CTL.CSPOL0` =0).
- The communication is in MSB bit first format (for example, `ADCC_CTL.LSBF0` =0).
- The control and data bits are in normal (not data swapped) format (for example, `ADCC_CTL.DSWP0` =0).

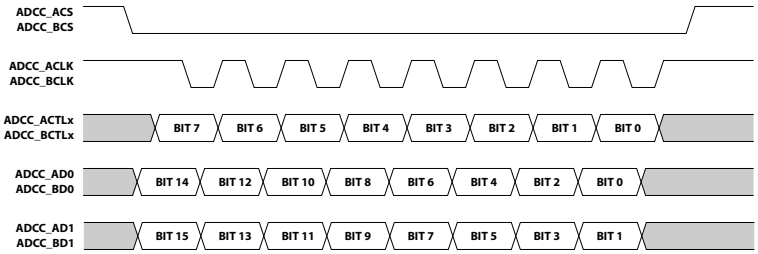

Figure 26-14: ADCC Timing Guide for Programming

In the ***ADCC Clock Edge Programming (Clock on Falling Edge)*** figure, the ADC expects the first edge as falling edge after assertion of the chip select signal. The ADC interface of ADCC drives the control word bits on the ADC control lines on the rising edges of the ADC clock. The ADC drives the bits of converted data on the falling edge of the ADC clock (for example, `ADCC_CTL.CKPOL0` =0).

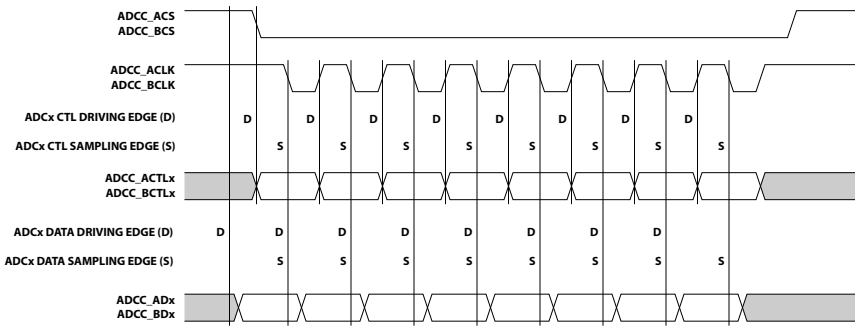

Figure 26-15: ADCC Clock Edge Programming (Clock on Falling Edge)

In the ***ADCC Chip Select Idle Programming*** figure, the ADC expects the control lines to be held low when not transmitting a valid control word during any of the phases of the ADC sampling sequence (for example, `ADCC_CTL.TIDLE0` =0).

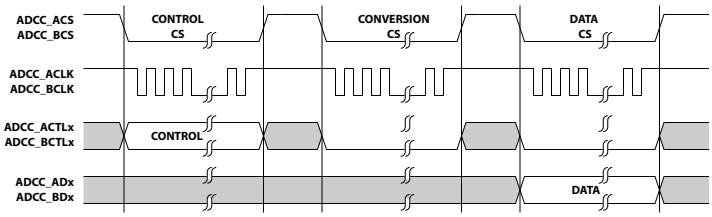

Figure 26-16: ADCC Chip Select Idle Programming

The ***ADC 8-Bit Control Word Format*** table describes the fields in the 8-bit ADC control word. The control register of each event provides a 16-bit control word field (`ADCC_EVCTL[nn].CTLWD`) for storing an event-specific control word. Use only the lower 8 bits. Only the channel ID is sent through the control word. The ADCC selects the ADC interface through the `ADCC_EVCTL[nn].ADCSEL` field.

Table 26-10: ADC 8-Bit Control Word Format

Bit Position	Field	Program as ...
[7:4]	`ADC_CHAN` Each ADC interface has 12 channels	ADC Channel number
3	Reserved	Program as 1
2	Reserved	Program as 1
1	`SHARE_CS`	Program 1 if simultaneous sampling is necessary
0	Reserved	Program as 1

ADSP-CM40x ADCC Register Descriptions

ADC Controller (ADCC) contains the following registers.

Table 26-11: ADSP-CM40x ADCC Register List

Name	Description
`ADCC_BPTR0`	Base Pointer 0 Register
`ADCC_BPTR1`	DMA Base Pointer 1 Register
`ADCC_BWMON0`	Bandwidth Monitor 0 Register
`ADCC_BWMON1`	Bandwidth Monitor 1 Register
`ADCC_CBSIZ0`	Circular Buffer Size 0 Register
`ADCC_CBSIZ1`	Circular Buffer Size 1 Register
`ADCC_CFG`	ADC Configuration Register
`ADCC_CTL`	Control Register
`ADCC_ECOL`	Event Collision Status Register
`ADCC_EIMSK`	Event Interrupt Mask Register

Table 26-11: ADSP-CM40x ADCC Register List (Continued)

Name	Description
ADCC_EIMSK_CLR	Event Interrupt Mask Clear Register
ADCC_EIMSK_SET	Event Interrupt Mask Set Register
ADCC_EISTAT	Event Interrupt Status Register
ADCC_EMISS	Event Miss Status Register
ADCC_EPND	Pending Events Status Register
ADCC_ERRMSK	Error Mask Register
ADCC_ERRMSK_CLR	Error Mask Clear Register
ADCC_ERRMSK_SET	Error Mask Set Register
ADCC_ERRSTAT	Error Status Register
ADCC_EVCTL[nn]	Event n Control Register
ADCC_EVDAT[nn]	Event n Data Register
ADCC_EVSTAT[nn]	Event n Status Register
ADCC_EVTEN	Event Enable Register
ADCC_EVTEN_CLR	Event Enable Clear Register
ADCC_EVTEN_SET	Event Enable Set Register
ADCC_EVT[nn]	Event n Time Register
ADCC_FIMSK	Frame Interrupt Mask Register
ADCC_FIMSK_CLR	Frame Interrupt Mask Clear Register
ADCC_FIMSK_SET	Frame Interrupt Mask Set Register
ADCC_FISTAT	Frame Interrupt Status Register
ADCC_FRINC0	Frame Increment 0 Register
ADCC_FRINC1	Frame Increment 1 Register
ADCC_T0STAT	Timer 0 Status Register
ADCC_T1STAT	Timer 1 Status Register
ADCC_TCA0	Timing Control A (ADC0) Register
ADCC_TCA1	Timing Control A (ADC1) Register
ADCC_TCB0	Timing Control B (ADC0) Register
ADCC_TCB1	Timing Control B (ADC1) Register
ADCC_TMR0	Timer 0 Current Count Register
ADCC_TMR1	Timer 1 Current Count Register

Base Pointer 0 Register

The ADCC_BPTR0 register provides the base pointer (address) used for placing the first Timer 0 frame's event-data into memory through DMA. The value of base address (pointer) in the ADCC_BPTR0 register at the time of the trigger pulse (start of frame) corresponds to one of the following Timer 0 related frames:

- The first frame after ADCC is enabled
- The first frame after the ADCC_BPTR0 register is written by core (in linear DMA mode)
- The first frame after a loop back occurs in circular buffering mode

The data from the first frame's events is placed in memory, starting at the address indicated with the value of the frame base address. Each of the event data are placed in a location (ADCC_BPTR0 + ADCC_EVCTL[nn].EVTOFS), where ADCC_EVCTL[nn].EVTOFS is a programmable offset for each event.

The second frame's base address (pointer) is calculated as (ADCC_BPTR0 + ADCC_FRINC0).

The third frame's base address (pointer) is calculated as (ADCC_BPTR0 + (2 x ADCC_FRINC0)).

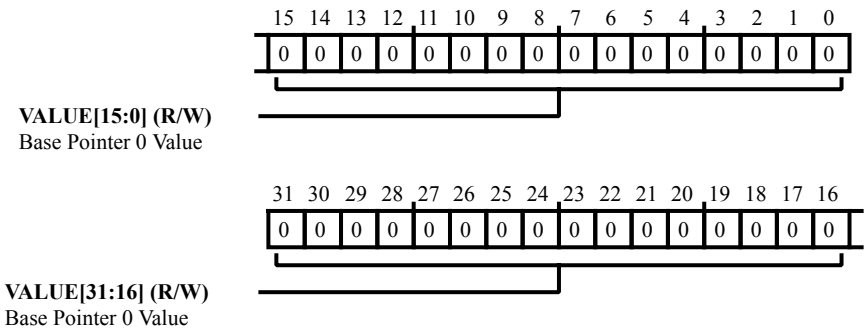

Figure 26-17: ADCC_BPTR0 Register Diagram

Table 26-12: ADCC_BPTR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Base Pointer 0 Value. The ADCC_BPTR0.VALUE bits hold the base pointer (address) for the first frame of the DMA. Note that bit 0 must be written as 0 to ensure correct address alignment.

DMA Base Pointer 1 Register

The ADCC_BPTR1 register provides the base pointer (address) used for placing the first Timer 1 frame's event-data into memory through DMA. The value of base address (pointer) in the ADCC_BPTR1 register at the time of the trigger pulse (start of frame) corresponds to one of the following Timer 1 related frames:

- The first frame after ADCC is enabled
- The first frame after the ADCC_BPTR1 register is written by core (in linear DMA mode)
- The first frame after a loop back occurs in circular buffering mode

The data from the first frame's events is placed in memory, starting at the address indicated with the value of the frame base address. Each of the event data are placed in a location (ADCC_BPTR1 + ADCC_EVCTL[nn].EVTOFS), where ADCC_EVCTL[nn].EVTOFS is a programmable offset for each event.

The second frame's base address (pointer) is calculated as (ADCC_BPTR1 + ADCC_FRINC0).

The third frame's base address (pointer) is calculated as (ADCC_BPTR1 + (2 x ADCC_FRINC0)).

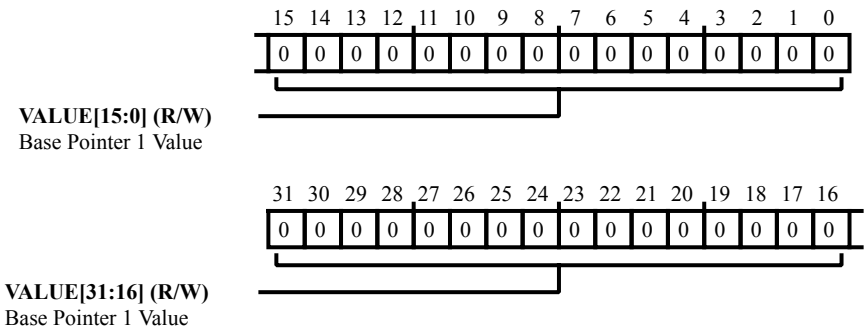

Figure 26-18: ADCC_BPTR1 Register Diagram

Table 26-13: ADCC_BPTR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Base Pointer 1 Value. The ADCC_BPTR1.VALUE bits hold the base pointer (address) for the first frame of the DMA. Note that bit 0 must be written as 0 to ensure correct address align-ment.The ADCC_BPTR1.VALUE bits hold the base pointer (address) for the first frame of the DMA. Note that bit 0 must be written as 0 to ensure correct address alignment.

Bandwidth Monitor 0 Register

The ADCC_BWMON0 register monitors the Timer 0 count and can be used to signal an error if the count exceeds the ADCC_BWMON0.CNT value in DMA mode. This checks whether the DMA transfer of all events in the frame are completed before a certain count of the timer. To monitor for completion of all events and their DMA before n SYSCLK cycles after the trigger, program the value n in the ADCC_BWMON0.CNT field. When the timer count exceeds n, the ADCC issues an error interrupt and indicates the status with the ADCC_ERRSTAT.BWERR0 bit. The Timer stops only when all DMA transfers are complete and responses received, in DMA mode.

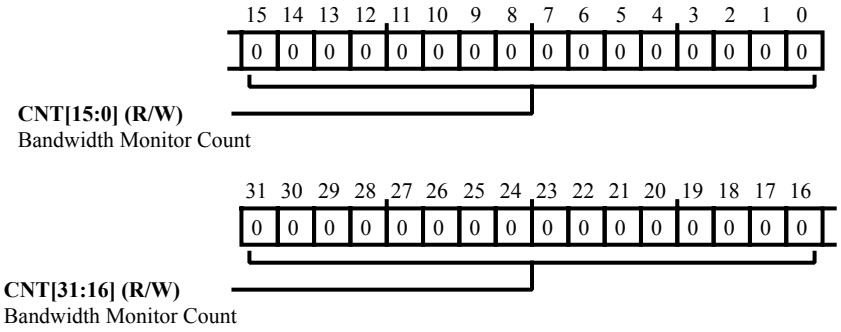

Figure 26-19: ADCC_BWMON0 Register Diagram

Table 26-14: ADCC_BWMON0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	CNT	Bandwidth Monitor Count. The ADCC_BWMON0.CNT bits hold the maximum expected Timer 0 count for a frame. If programmed as 0, bandwidth monitoring capability is disabled.

Bandwidth Monitor 1 Register

The ADCC_BWMON1 register monitors the Timer 1 count and can be used to signal an error if the count exceeds the ADCC_BWMON1.CNT value in DMA mode. This checks whether the DMA transfer of all events in the frame are completed before a certain count of the timer. To monitor for completion of all events and their DMA before n SYSCLK cycles after the trigger, program the value n in the ADCC_BWMON1.CNT field. When the timer count exceeds n, the ADCC issues an error interrupt and indicates the status with the ADCC_ERRSTAT.BWERR1 bit. The Timer stops only when all DMA transfers are complete and responses received, in DMA mode.

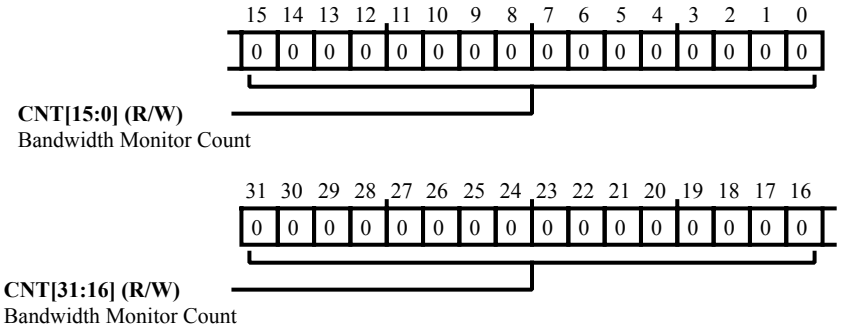

Figure 26-20: ADCC_BWMON1 Register Diagram

Table 26-15: ADCC_BWMON1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	CNT	Bandwidth Monitor Count. The ADCC_BWMON1.CNT bits hold the maximum expected Timer 1 count for a frame. If programmed as 0, bandwidth monitoring capability is disabled.

Circular Buffer Size 0 Register

The ADCC_CBSIZ0 register holds the circular buffer size to be used for the Timer 0 frames of DMA. If the ADCC_CBSIZ0.VALUE bit field is programmed as 0, the ADCC performs only linear buffering is for the Timer 0 frames. For more information about ADCC linear and circular buffering, see the ADCC functional description.

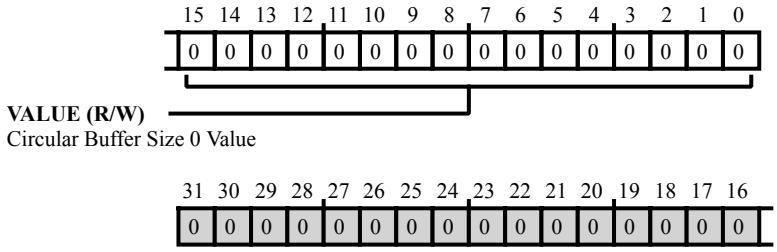

Figure 26-21: ADCC_CBSIZ0 Register Diagram

Table 26-16: ADCC_CBSIZ0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Circular Buffer Size 0 Value. The ADCC_CBSIZ0.VALUE bits select the number of Timer 0 frames in a DMA circular buffer (number of frames after which to wrap back); if =0, linear mode is used instead of circular DMA mode.

Circular Buffer Size 1 Register

The ADCC_CBSIZ1 register holds the circular buffer size to be used for the Timer 1 frames of DMA. If the ADCC_CBSIZ1.VALUE bit field is programmed as 0, the ADCC performs only linear buffering is for the Timer 1 frames. For more information about ADCC linear and circular buffering, see the ADCC functional description.

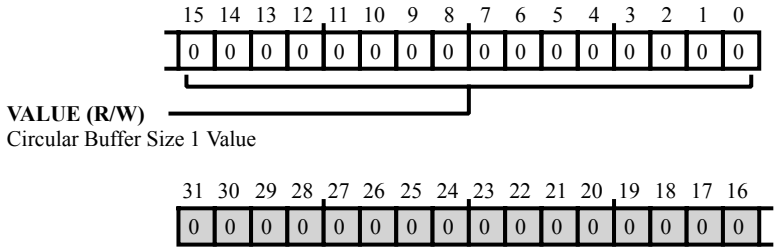

Figure 26-22: ADCC_CBSIZ1 Register Diagram

Table 26-17: ADCC_CBSIZ1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Circular Buffer Size 1 Value. The ADCC_CBSIZ1.VALUE bits select the number of Timer 1 frames in a DMA circular buffer (number of frames after which to wrap back); if =0, linear mode is used instead of circular DMA mode.

ADC Configuration Register

The ADCC_CFG register configures ADC specific features and indicates status of pending ADC configuration operations.

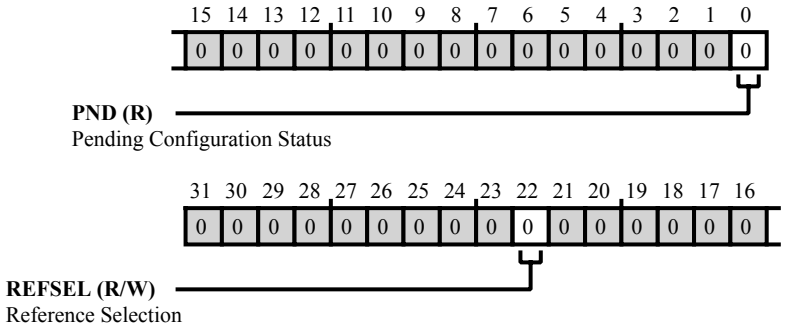

Figure 26-23: ADCC_CFG Register Diagram

Table 26-18: ADCC_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
22 (R/W)	REFSEL	Reference Selection. The ADCC_CFG.REFSEL bit selects whether the ADC uses and internal or external voltage reference.	
		0	Internal Reference
		1	External Reference
0 (R/NW)	PND	Pending Configuration Status. The ADCC_CFG.PND bit indicates the pending status of ADC configuration. While configuration is pending completion, the ADCC ignores triggers.	
		0	No Pending Configure (Honors Triggers)
		1	Pending Configure (Ignores Triggers)

Control Register

The ADCC_CTL register enables ADCC operation and configures a number of ADC0 and ADC1 interface features.

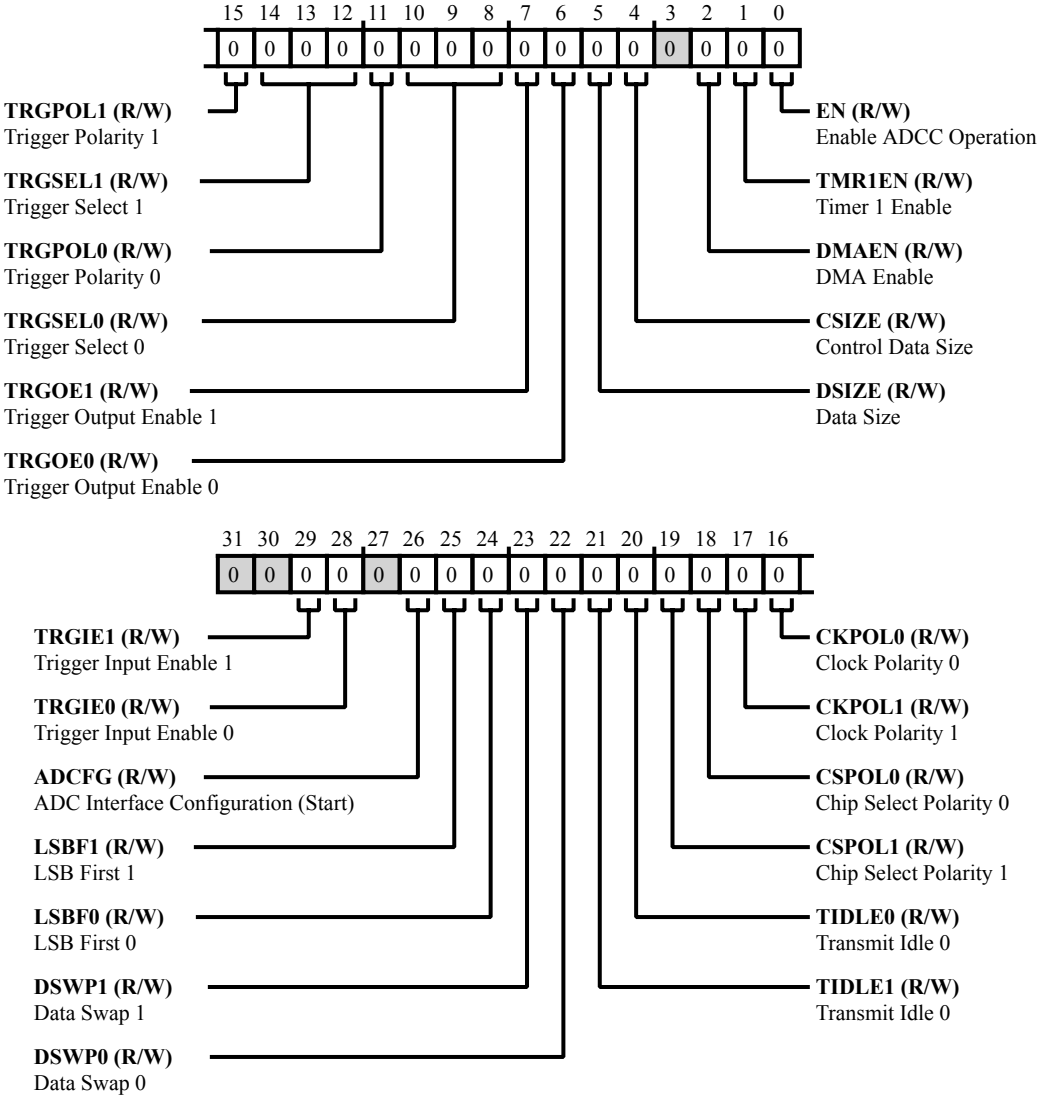

Figure 26-24: ADCC_CTL Register Diagram

Table 26-19: ADCC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/W)	TRGIE1	Trigger Input Enable 1. The ADCC_CTL.TRGIE1 bit enables recognition of trigger input as valid to initiate Timer 1 frames.	
		0	Disable Recognition of Trigger Input
		1	Enable Recognition of Trigger Input
28 (R/W)	TRGIE0	Trigger Input Enable 0. The ADCC_CTL.TRGIE0 bit enables recognition of trigger input as valid to initiate Timer 0 frames.	
		0	Disable Recognition of Trigger Input
		1	Enable Recognition of Trigger Input
26 (R/W)	ADCFG	ADC Interface Configuration (Start). The ADCC_CTL.ADCFG bit initiates configuring the ADC interface according to the selections in the ADCC_CFG register. The ADCC auto clears this bit after completing ADC configuration.	
		0	No Action
		1	Start ADC Configuration Operation
25 (R/W)	LSBF1	LSB First 1. The ADCC_CTL.LSBF1 bit selects LSB or MSB first mode for ADC1 interface operations. For more information about LSB/MSB first mode operations, see the ADCC functional description.	
		0	MSB First Mode
		1	LSB First Mode
24 (R/W)	LSBF0	LSB First 0. The ADCC_CTL.LSBF0 bit selects LSB or MSB first mode for ADC0 interface operations. For more information about LSB/MSB first mode operations, see the ADCC functional description.	
		0	MSB First Mode
		1	LSB First Mode
23 (R/W)	DSWP1	Data Swap 1. The ADCC_CTL.DSWP1 bit enables data swap operations on ADC1 for dual-bit (2-bit wide) control and data. For more information about data swap operations, see the ADCC functional description.	
		0	Disable Data Swap
		1	Enable Data Swap

Table 26-19: ADCC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
22 (R/W)	DSWP0	Data Swap 0. The ADCC_CTL.DSWP0 bit enables data swap operations on ADC0 for dual-bit (2-bit wide) control and data. For more information about data swap operations, see the ADCC functional description.	
		0	Disable Data Swap
		1	Enable Data Swap
21 (R/W)	TIDLE1	Transmit Idle 0. The ADCC_CTL.TIDLE1 bit selects the value to hold on the ADCC_BCTL0 pin for ADC1 when idle (not transmitting a valid control register). The first chip select after enabling the ADCC is a control chip select. The ADCC_CTL.TIDLE1 value is held on the control pins after this chip select, unless a valid control transmission takes place.	
		0	Hold Idle Control Pin Low
		1	Hold Idle Control Pin High
20 (R/W)	TIDLE0	Transmit Idle 0. The ADCC_CTL.TIDLE0 bit selects the value to hold on the ADCC_ACTL0 pin for ADC0 when idle (not transmitting a valid control register). The first chip select after enabling the ADCC is a control chip select. The ADCC_CTL.TIDLE0 value is held on the control pins after this chip select, unless a valid control transmission takes place.	
		0	Hold Idle Control Pin Low
		1	Hold Idle Control Pin High
19 (R/W)	CSPOL1	Chip Select Polarity 1. The ADCC_CTL.CSPOL1 bit selects the polarity (active high or low) for $\overline{\text{ADCC\_BCS}}$ pin (ADC1 interface).	
		0	Active Low CS
		1	Active High CS
18 (R/W)	CSPOL0	Chip Select Polarity 0. The ADCC_CTL.CSPOL0 bit selects the polarity (active high or low) for $\overline{\text{ADCC\_ACS}}$ pin (ADC0 interface).	
		0	Active Low CS
		1	Active High CS

Table 26-19: ADCC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	CKPOL1	Clock Polarity 1. The ADCC_CTL.CKPOL1 bit selects the clock polarity for the ADC1 interface. This selection chooses the clock edge (rising or falling) driven after the ADCC_BCS pin is asserted for sampling the first bit of control and data.	
		0	Sample on Falling Edge
		1	Sample on Rising Edge
16 (R/W)	CKPOL0	Clock Polarity 0. The ADCC_CTL.CKPOL0 bit selects the clock polarity for the ADC0 interface. This selection chooses the clock edge (rising or falling) driven after the ADCC_ACS pin is asserted for sampling the first bit of control and data.	
		0	Sample on Falling Edge
		1	Sample on Rising Edge
15 (R/W)	TRGPOL1	Trigger Polarity 1. The ADCC_CTL.TRGPOL1 bit select the polarity for trigger associated with Timer 1.	
		0	Trigger on Falling Edge
		1	Trigger on Rising Edge
14:12 (R/W)	TRGSEL1	Trigger Select 1. The ADCC_CTL.TRGSEL1 bits selects the Timer 1 initiate trigger from among the ADCC_SLV[n] triggers from =0 for trigger 0 to =n for trigger n.	
11 (R/W)	TRGPOL0	Trigger Polarity 0. The ADCC_CTL.TRGPOL0 bit select the polarity for trigger associated with Timer 0.	
		0	Trigger on Falling Edge
		1	Trigger on Rising Edge
10:8 (R/W)	TRGSEL0	Trigger Select 0. The ADCC_CTL.TRGSEL0 bits selects the Timer 0 initiate trigger from among the ADCC_SLV[n] triggers from =0 for trigger 0 to =n for trigger n.	
7 (R/W)	TRGOE1	Trigger Output Enable 1. The ADCC_CTL.TRGOE1 bit enables the trigger output generation for Timer 1 events.	
		0	Disable
		1	Enable

Table 26-19: ADCC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	TRGOE0	Trigger Output Enable 0. The ADCC_CTL.TRGOE0 bit enables the trigger output generation for Timer 0 events.	
		0	Disable
		1	Enable
5 (R/W)	DSIZE	Data Size. The ADCC_CTL.DSIZE bit selects the interface width (single- or dual-bit) for receiving sample data from the ADC.	
		0	1-Bit Wide Interface
		1	2-Bit Wide Interface
4 (R/W)	CSIZE	Control Data Size. The ADCC_CTL.CSIZE bit selects the interface width (single- or dual-bit) for transmitting control data to the ADC.	
		0	1-Bit Wide Interface
		1	2-Bit Wide Interface
2 (R/W)	DMAEN	DMA Enable. The ADCC_CTL.DMAEN bit enables ADCC DMA operation, selecting the mode for sampled data transfer from the ADC to memory. If DMA is disabled (=0), the core should read data from the ADCC_EVDAT[nn] registers. If DMA is enabled, the ADCC transfers data using DMA, and th linear or circular buffer mode is chosen by the ADCC_CBSIZ0 or ADCC_CBSIZ1 register.	
		0	Disable
		1	Enable
1 (R/W)	TMR1EN	Timer 1 Enable. The ADCC_CTL.TMR1EN bit enables Timer 1. Note that Timer 0 is always enabled when the ADCC is enabled.	
		0	Disable
		1	Enable
0 (R/W)	EN	Enable ADCC Operation. The ADCC_CTL.EN bit enables ADCC operation (global enable for module).	
		0	Disable
		1	Enable

Event Collision Status Register

The ADCC_ECOL register indicates the collision status for ADCC events. When any bit in this register is set, the ADCC_ERRSTAT.ECOL bit is set, which may optionally cause the ADCC to generate output on the ADCC_ERR pin.

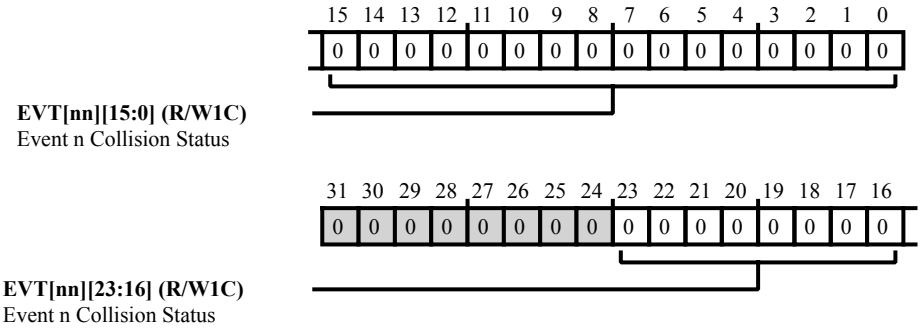

Figure 26-25: ADCC_ECOL Register Diagram

Table 26-20: ADCC_ECOL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:0 (R/W1C)	EVT[nn]	Event n Collision Status. The ADCC_ECOL.EVT[nn] bits each correspond to an ADCC event (n from 23 to 0) that underwent collision. When any bit in this register is set, the ADCC_ERRSTAT.ECOL bit is set, which may optionally cause the ADCC to generate output on the ADCC_ERR pin. The function of each bit is shown in the enumerations below.	
		0	No Status
		16777215	Event Collision Occurred

Event Interrupt Mask Register

The ADCC_EIMSK register masks (disables) generation of ADCC_TMR0_EVT or ADCC_TMR1_EVT event interrupts based on status in the ADCC_EISTAT register.

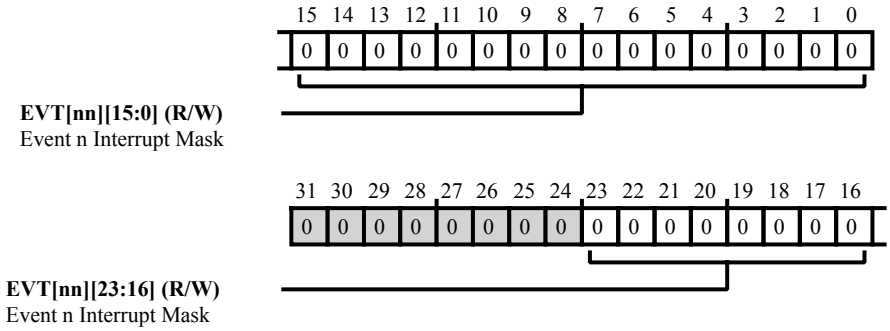

Figure 26-26: ADCC_EIMSK Register Diagram

Table 26-21: ADCC_EIMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:0 (R/W)	EVT[nn]	Event n Interrupt Mask. The ADCC_EIMSK.EVT[nn] bits mask (disable) each correspond to data interrupts from ADCC events (n from 23 to 0). The function of each bit is shown in the enumerations below.	
		0	Unmask (Enable) Event Interrupt
		16777215	Mask (Disable) Event Interrupt

Event Interrupt Mask Clear Register

The ADCC_EIMSK_CLR register can be used to selectively clear bits in the ADCC_EIMSK register without affecting other bits in the register. Writing a 1 to any bit position in ADCC_EIMSK_CLR clears the corresponding bit in ADCC_EIMSK. Reading the ADCC_EIMSK_CLR register returns the data present in the ADCC_EIMSK register.

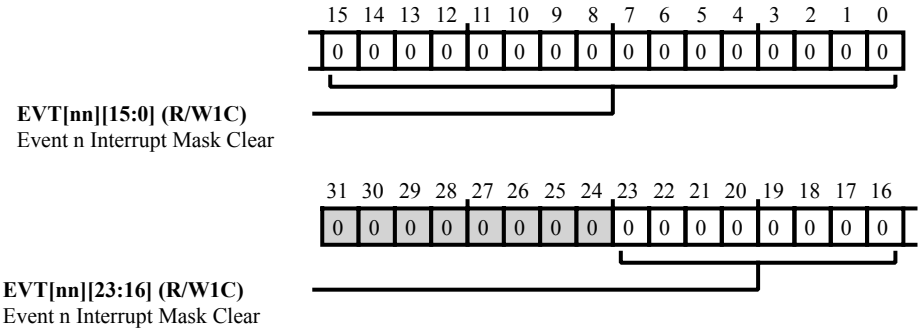

Figure 26-27: ADCC_EIMSK_CLR Register Diagram

Table 26-22: ADCC_EIMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/W1C)	EVT[nn]	Event n Interrupt Mask Clear. The ADCC_EIMSK_CLR.EVT[nn] bits permit clearing individual bits in the ADCC_EIMSK register without affecting other bits in the register. Write 1 to individual ADCC_EIMSK_CLR.EVT[nn] bits to clear the corresponding bit in ADCC_EIMSK.

Event Interrupt Mask Set Register

The ADCC_EIMSK_SET register can be used to selectively set bits in the ADCC_EIMSK register without affecting other bits in the register. Writing a 1 to any bit position in ADCC_EIMSK_SET sets the corresponding bit in ADCC_EIMSK. Reading the ADCC_EIMSK_SET register returns the data present in the ADCC_EIMSK register.

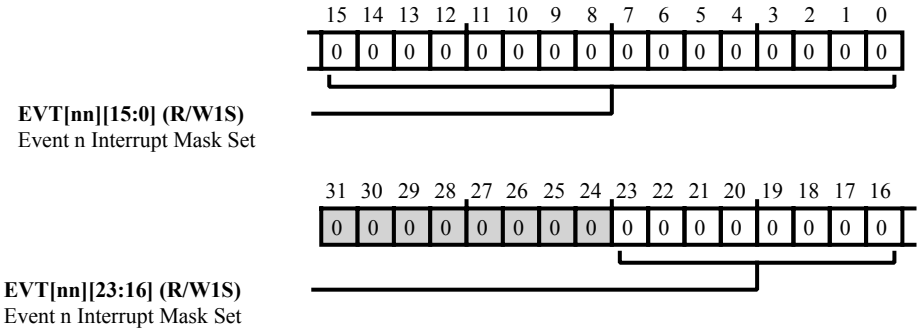

Figure 26-28: ADCC_EIMSK_SET Register Diagram

Table 26-23: ADCC_EIMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/W1S)	EVT[nn]	Event n Interrupt Mask Set. The ADCC_EIMSK_SET.EVT[nn] bits permit setting individual bits in the ADCC_EIMSK register without affecting other bits in the register. Write 1 to individual ADCC_EIMSK_SET.EVT[nn] bits to set the corresponding bit in ADCC_EIMSK.

Event Interrupt Status Register

The ADCC_EISTAT register indicates the interrupt status bits for ADCC events. Based on the ADCC_EVCTL[nn].TMRSEL bit selection, the ADCC generates the corresponding ADCC_TMR0_EVT or ADCC_TMR1_EVT event interrupts. Each of these interrupts can be optionally masked (disable interrupt output) using the bits in ADCC_EIMSK register.

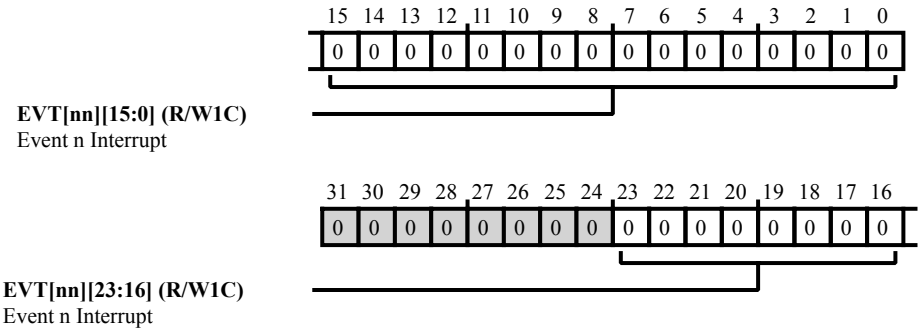

Figure 26-29: ADCC_EISTAT Register Diagram

Table 26-24: ADCC_EISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:0 (R/W1C)	EVT[nn]	Event n Interrupt. The ADCC_EISTAT.EVT[nn] bits each correspond to data interrupts from ADCC events (n from 23 to 0). The function of each bit is shown in the enumerations below. Note that data interrupt from events are applicable only in non-DMA mode.	
		0	No Data Pending for Core Read
		16777215	Event Data Pending for Core Read

Event Miss Status Register

The ADCC_EMISS register indicates the miss status for ADCC events. When any bit in this register is set, the ADCC_ERRSTAT.EMIS bit is set, which may optionally cause the ADCC to generate output on the ADCC_ERR pin.

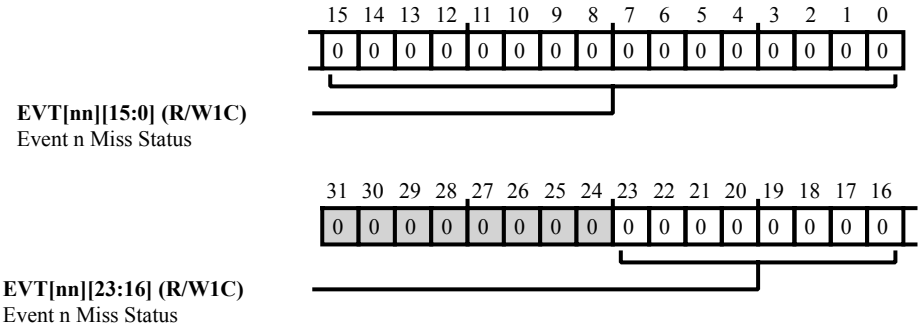

Figure 26-30: ADCC_EMISS Register Diagram

Table 26-25: ADCC_EMISS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:0 (R/W1C)	EVT[nn]	Event n Miss Status. The ADCC_EMISS.EVT[nn] bits each correspond to an ADCC event (n from 23 to 0) that were missed. When any bit in this register is set, the ADCC_ERRSTAT.EMIS bit is set, which may optionally cause the ADCC to generate output on the ADCC_ERR pin.The function of each bit is shown in the enumerations below.	
		0	No Status
		16777215	Event Miss Occurred

Pending Events Status Register

The ADCC_EPND register indicates which events within the current frames are pending (waiting for data transfer). Each of the ADCC_EPND.EVT[nn] bits from 0 to 23 corresponds to an events from 0 to 23. When the trigger pulse initiating a frame (corresponding to a Timer) is detected, all the status pending bits corresponding to enabled events in that frame are set. The ADCC clears each status bit on receiving the data from ADC for the corresponding event. If a pending event is missed, the ADCC clears the corresponding status bit.

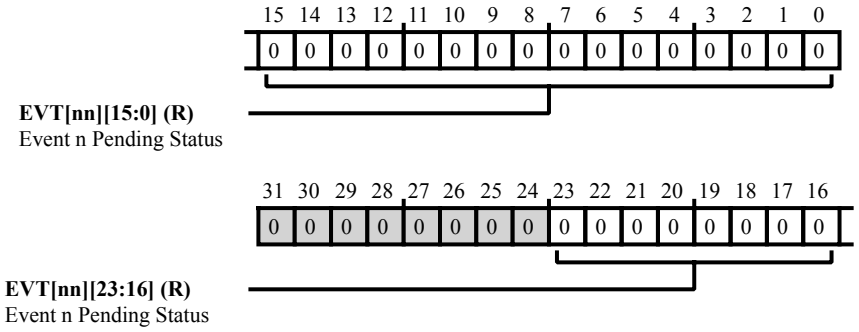

Figure 26-31: ADCC_EPND Register Diagram

Table 26-26: ADCC_EPND Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:0 (R/NW)	EVT[nn]	Event n Pending Status. The ADCC_EPND.EVT[nn] bits each correspond to an ADCC event (n from 23 to 0) that are pending (waiting for data transfer). The function of each bit is shown in the enumerations below.	
		0	No Status
		16777215	Event Pending

Error Mask Register

The ADCC_ERRMSK register masks (disables) or unmasks (enables) reporting of ADC related errors.

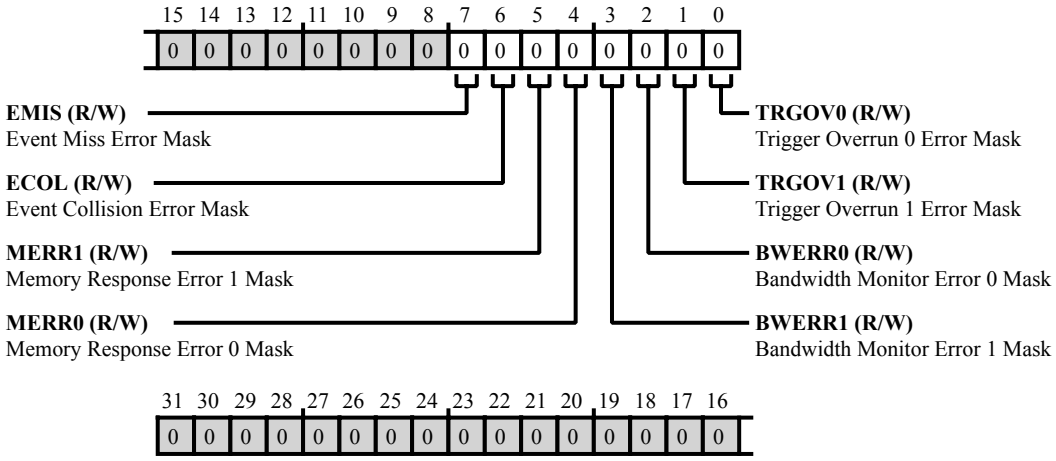

Figure 26-32: ADCC_ERRMSK Register Diagram

Table 26-27: ADCC_ERRMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	EMIS	Event Miss Error Mask. The ADCC_ERRMSK.EMIS bit masks (disables) generating an error interrupt on an event miss error.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)
6 (R/W)	ECOL	Event Collision Error Mask. The ADCC_ERRMSK.ECOL bit masks (disables) generating an error interrupt on an event collision error.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)
5 (R/W)	MERR1	Memory Response Error 1 Mask. The ADCC_ERRMSK.MERR1 bit masks (disables) generating an error interrupt on a memory response error related to Timer 1.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)

Table 26-27: ADCC_ERRMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	MERR0	Memory Response Error 0 Mask. The ADCC_ERRMSK.MERR0 bit masks (disables) generating an error interrupt on a memory response error related to Timer 0.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)
3 (R/W)	BWERR1	Bandwidth Monitor Error 1 Mask. The ADCC_ERRMSK.BWERR1 bit masks (disables) generating an error interrupt on a bandwidth monitor error related to Timer 1.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)
2 (R/W)	BWERR0	Bandwidth Monitor Error 0 Mask. The ADCC_ERRMSK.BWERR0 bit masks (disables) generating an error interrupt on a bandwidth monitor error related to Timer 0.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)
1 (R/W)	TRGOV1	Trigger Overrun 1 Error Mask. The ADCC_ERRMSK.TRGOV1 bit masks (disables) generating an error interrupt on a trigger overrun for the trigger associated with Timer 1.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)
0 (R/W)	TRGOV0	Trigger Overrun 0 Error Mask. The ADCC_ERRMSK.TRGOV0 bit masks (disables) generating an error interrupt on a trigger overrun for the trigger associated with Timer 0.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)

Error Mask Clear Register

The ADCC_ERRMSK_CLR register can be used to selectively clear bits in the ADCC_ERRMSK register without affecting other bits in the register. Writing a 1 to any bit position in ADCC_ERRMSK_CLR clears the corresponding bit in ADCC_ERRMSK. Reading the ADCC_ERRMSK_CLR register returns the data present in the ADCC_ERRMSK register.

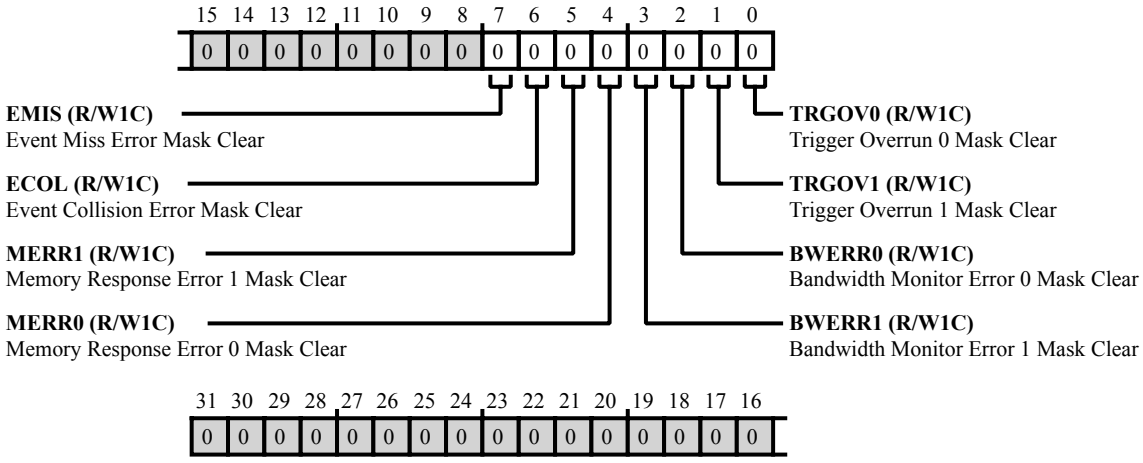

Figure 26-33: ADCC_ERRMSK_CLR Register Diagram

Table 26-28: ADCC_ERRMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W1C)	EMIS	Event Miss Error Mask Clear. Write 1 to ADCC_ERRMSK_CLR.EMIS to clear the corresponding bit in ADCC_ERRMSK.
6 (R/W1C)	ECOL	Event Collision Error Mask Clear. Write 1 to ADCC_ERRMSK_CLR.ECOL to clear the corresponding bit in ADCC_ERRMSK.
5 (R/W1C)	MERR1	Memory Response Error 1 Mask Clear. Write 1 to ADCC_ERRMSK_CLR.MERR1 to clear the corresponding bit in ADCC_ERRMSK.
4 (R/W1C)	MERR0	Memory Response Error 0 Mask Clear. Write 1 to ADCC_ERRMSK_CLR.MERR0 to clear the corresponding bit in ADCC_ERRMSK.
3 (R/W1C)	BWERR1	Bandwidth Monitor Error 1 Mask Clear. Write 1 to ADCC_ERRMSK_CLR.BWERR1 to clear the corresponding bit in ADCC_ERRMSK.
2 (R/W1C)	BWERR0	Bandwidth Monitor Error 0 Mask Clear.

Table 26-28: ADCC_ERRMSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		Write 1 to ADCC_ERRMSK_CLR.BWERR0 to clear the corresponding bit in ADCC_ERRMSK.
1 (R/W1C)	TRGOV1	Trigger Overrun 1 Mask Clear. Write 1 to ADCC_ERRMSK_CLR.TRGOV1 to clear the corresponding bit in ADCC_ERRMSK.
0 (R/W1C)	TRGOV0	Trigger Overrun 0 Mask Clear. Write 1 to ADCC_ERRMSK_CLR.TRGOV0 to clear the corresponding bit in ADCC_ERRMSK.

Error Mask Set Register

The ADCC_ERRMSK_SET register can be used to selectively set bits in the ADCC_ERRMSK register without affecting other bits in the register. Writing a 1 to any bit position in ADCC_ERRMSK_SET sets the corresponding bit in ADCC_ERRMSK. Reading the ADCC_ERRMSK_SET register returns the data present in the ADCC_ERRMSK register.

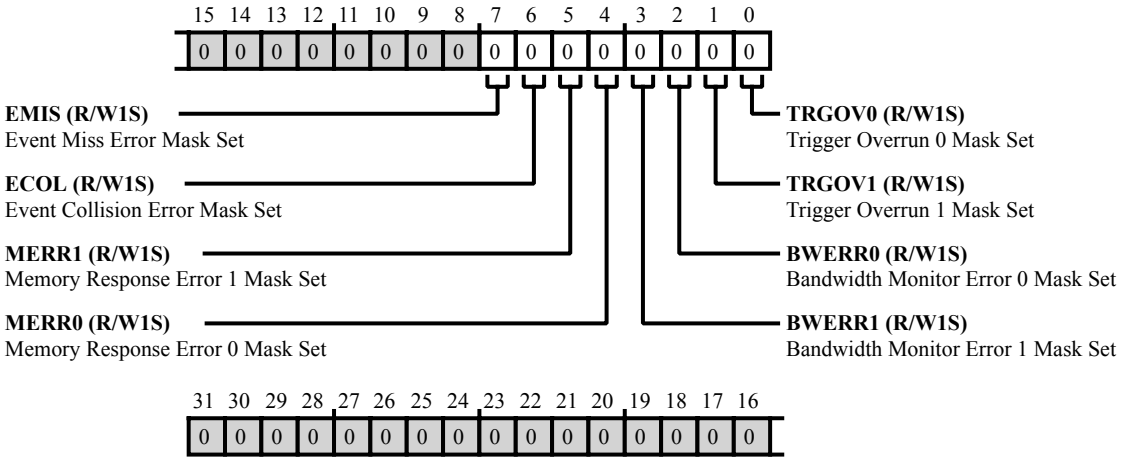

Figure 26-34: ADCC_ERRMSK_SET Register Diagram

Table 26-29: ADCC_ERRMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W1S)	EMIS	Event Miss Error Mask Set. Write 1 to ADCC_ERRMSK_SET.EMIS to set the corresponding bit in ADCC_ERRMSK.
6 (R/W1S)	ECOL	Event Collision Error Mask Set. Write 1 to ADCC_ERRMSK_SET.ECOL to set the corresponding bit in ADCC_ERRMSK.
5 (R/W1S)	MERR1	Memory Response Error 1 Mask Set. Write 1 to ADCC_ERRMSK_SET.MERR1 to set the corresponding bit in ADCC_ERRMSK.
4 (R/W1S)	MERR0	Memory Response Error 0 Mask Set. Write 1 to ADCC_ERRMSK_SET.MERR0 to set the corresponding bit in ADCC_ERRMSK.
3 (R/W1S)	BWERR1	Bandwidth Monitor Error 1 Mask Set. Write 1 to ADCC_ERRMSK_SET.BWERR1 to set the corresponding bit in ADCC_ERRMSK.
2 (R/W1S)	BWERR0	Bandwidth Monitor Error 0 Mask Set. Write 1 to ADCC_ERRMSK_SET.BWERR0 to set the corresponding bit in ADCC_ERRMSK.
1	TRGOV1	Trigger Overrun 1 Mask Set.

Table 26-29: ADCC_ERRMSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(R/W1S)		Write 1 to ADCC_ERRMSK_SET.TRGOV1 to set the corresponding bit in ADCC_ERRMSK.
0 (R/W1S)	TRGOV0	Trigger Overrun 0 Mask Set. Write 1 to ADCC_ERRMSK_SET.TRGOV0 to set the corresponding bit in ADCC_ERRMSK.

Error Status Register

The ADCC_ERRSTAT register indicates status for errors relating to trigger overruns, bandwidth monitoring, memory responses, and events.

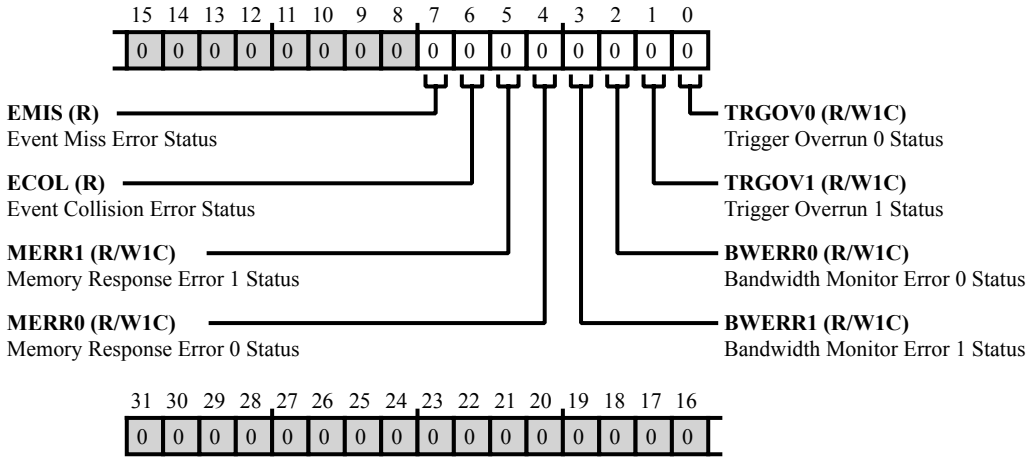

Figure 26-35: ADCC_ERRSTAT Register Diagram

Table 26-30: ADCC_ERRSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	EMIS	Event Miss Error Status. The ADCC_ERRSTAT.EMIS bit indicates whether an event was missed. To identify which the events were missed, read the value from the ADCC_EMISS register. For more information about missed events, see the ADCC functional description. To clear this error, clear the cause from the ADCC_EMISS register.	
		0	No Status
		1	Event Miss Error
6 (R/NW)	ECOL	Event Collision Error Status. The ADCC_ERRSTAT.ECOL bit indicates whether a collision has occurred for any event. To identify which events underwent collision, read the value from the ADCC_ECOL register. For more information about event collisions, see the ADCC functional description. To clear this error, clear the cause from the ADCC_ECOL register.	
		0	No Status
		1	Event Collision Error

Table 26-30: ADCC_ERRSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1C)	MERR1	Memory Response Error 1 Status. The ADCC_ERRSTAT.MERR1 bit indicates whether a memory write error response on data write issued for an event corresponding to Timer 1 has occurred.	
		0	No Status
		1	Memory Response Error
4 (R/W1C)	MERR0	Memory Response Error 0 Status. The ADCC_ERRSTAT.MERR0 bit indicates whether a memory write error response on data write issued for an event corresponding to Timer 0 has occurred.	
		0	No Status
		1	Memory Response Error
3 (R/W1C)	BWERR1	Bandwidth Monitor Error 1 Status. The ADCC_ERRSTAT.BWERR1 bit indicates whether a bandwidth monitor error (while monitoring is enabled, the timer count crossed the value written in ADCC_BWMON1) for Timer 1 has occurred.	
		0	No Status
		1	Bandwidth Monitor Error
2 (R/W1C)	BWERR0	Bandwidth Monitor Error 0 Status. The ADCC_ERRSTAT.BWERR0 bit indicates whether a bandwidth monitor error (while monitoring is enabled, the timer count crossed the value written in ADCC_BWMON0) for Timer 0 has occurred.	
		0	No Status
		1	Bandwidth Monitor Error
1 (R/W1C)	TRGOV1	Trigger Overrun 1 Status. The ADCC_ERRSTAT.TRGOV1 bit indicates whether a trigger overrun (a trigger input detected while a frame or ADC programming is in progress) for Timer 1 has occurred. The trigger input is ignored.	
		0	No Status
		1	Trigger Overrun
0 (R/W1C)	TRGOV0	Trigger Overrun 0 Status. The ADCC_ERRSTAT.TRGOV0 bit indicates whether a trigger overrun (a trigger input detected while a frame or ADC programming is in progress) for Timer 0 has occurred. The trigger input is ignored.	
		0	No Status
		1	Trigger Overrun

Event n Control Register

The ADCC_EVCTL[nn] register controls programmable features of the corresponding event.

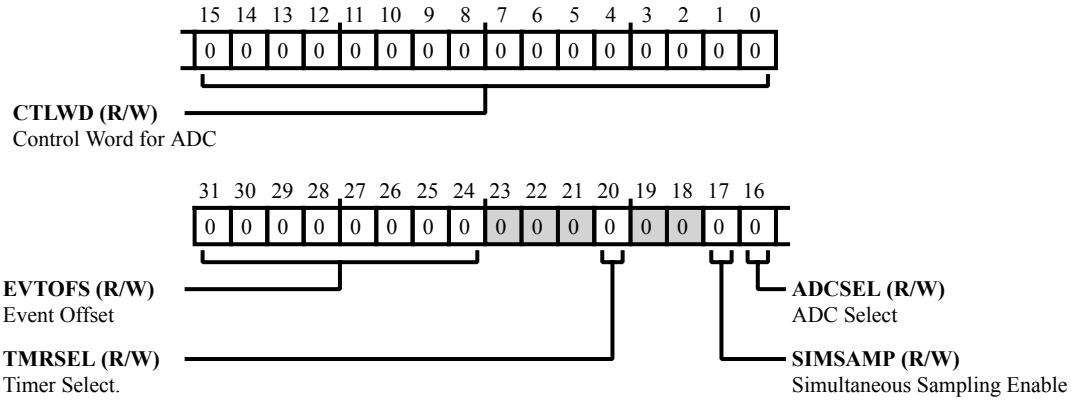

Figure 26-36: ADCC_EVCTL[nn] Register Diagram

Table 26-31: ADCC_EVCTL[nn] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:24 (R/W)	EVTOFS	Event Offset. The ADCC_EVCTL[nn].EVTOFS bits hold the memory address offset at which the event's data is stored. This offset is added to the frame's base pointer to produce the final memory address. ADCC_EVCTL[nn].EVTOFS's LSB must =0 for correct address alignment.	
20 (R/W)	TMRSEL	Timer Select.. The ADCC_EVCTL[nn].TMRSEL bit selects whether the event corresponds to Timer 0 or 1.	
		0	Event on Timer 0
		1	Event on Timer 1
17 (R/W)	SIMSAMP	Simultaneous Sampling Enable. The ADCC_EVCTL[nn].SIMSAMP bit enables simultaneous sampling operation.	
		0	Disable
		1	Enable
16 (R/W)	ADCSEL	ADC Select. The ADCC_EVCTL[nn].ADCSEL bit select whether the event executes on the ADC0 interface or the ADC1 interface.	
		0	Event on ADC0
		1	Event on ADC1
15:0	CTLWD	Control Word for ADC.	

Table 26-31: ADCC_EVCTL[nn] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(R/W)		The ADCC_EVCTL[nn].CTLWD bits hold the control word to be sent to ADC. These same 16 bits (or lesser, depending on the corresponding value of ADCC_TCA0.NCK or ADCC_TCA1.NCK) are sent to ADC. Program the control word LSB aligned in this field. For more information about control word content, see the ADCC programming guidelines.

Event n Data Register

The ADCC_EVDAT[nn] register holds the data sampled from the ADC channel. The data from this register can be read by the core or transferred through DMA. If the data sample received is less than 16 bits wide (depends on value of corresponding ADCC_TCA0.NCK or ADCC_TCA1.NCK), the data is MSB aligned in the ADCC_EVDAT[nn].VALUE field, and the lower bits are zero filled.

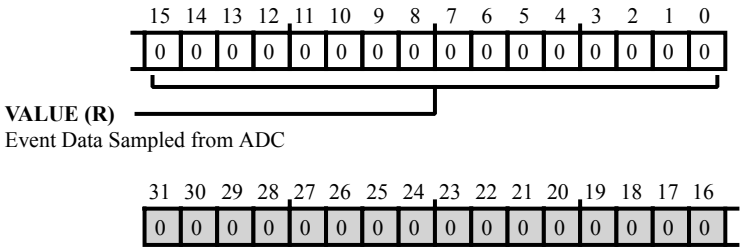

Figure 26-37: ADCC_EVDAT[nn] Register Diagram

Table 26-32: ADCC_EVDAT[nn] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Event Data Sampled from ADC. The ADCC_EVDAT[nn].VALUE bits hold the data of the event sampled from the ADC.

Event n Status Register

The ADCC_EVSTAT[nn] register indicates event status for event n.

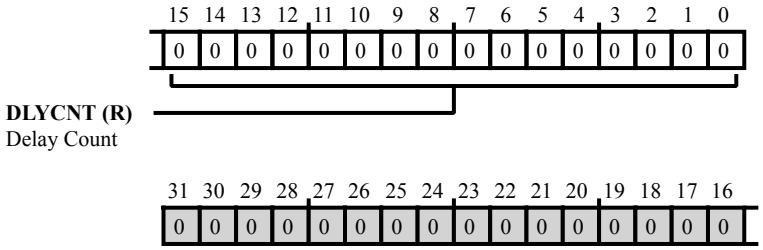

Figure 26-38: ADCC_EVSTAT[nn] Register Diagram

Table 26-33: ADCC_EVSTAT[nn] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	DLYCNT	Delay Count. The ADCC_EVSTAT[nn].DLYCNT bits indicate the number of SYSCLK cycles by which the event was delayed after the event match occurred. The ADCC updates this field when the corresponding ADCC_ACS or ADCC_BCS is asserted to transmit the control word corresponding to the event to the ADC.

Event Enable Register

The ADCC_EVTEN register enables detection of individual ADCC events. Based on the ADCC_EVCTL[nn].TMRSEL bit selection, the ADCC generates the corresponding ADCC_TMR0_EVT or ADCC_TMR0_EVT event interrupts. Note that the events corresponding to a frame must be enabled before the trigger corresponding to the frame arrives.

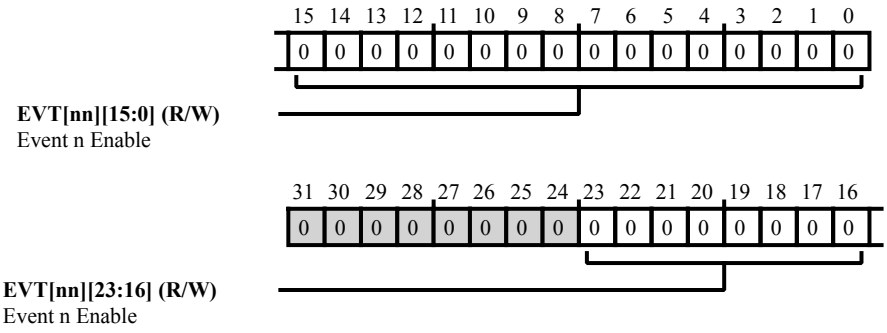

Figure 26-39: ADCC_EVTEN Register Diagram

Table 26-34: ADCC_EVTEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:0 (R/W)	EVT[nn]	Event n Enable. The ADCC_EVTEN.EVT[nn] bits each correspond to an ADCC event (n from 23 to 0). The function of each bit is shown in the enumerations below.	
		0	Disable Event Detection
		16777215	Enable Event Detection

Event Enable Clear Register

The ADCC_EVTEN_CLR register can be used to selectively clear bits in the ADCC_EVTEN register without affecting other bits in the register. Writing a 1 to any bit position in ADCC_EVTEN_CLR clears the corresponding bit in ADCC_EVTEN. Reading the ADCC_EVTEN_CLR register returns the data present in the ADCC_EVTEN register.

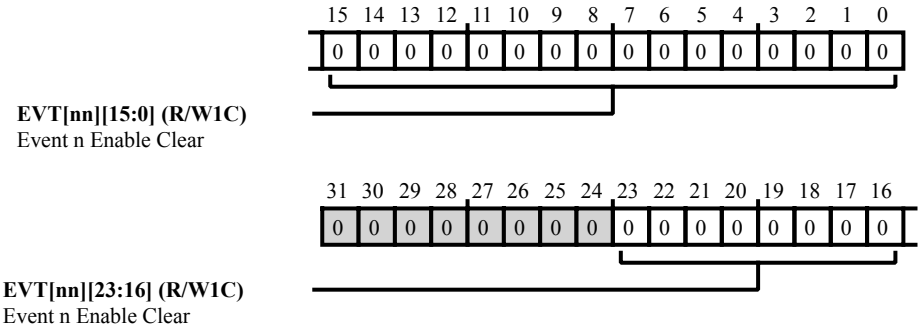

Figure 26-40: ADCC_EVTEN_CLR Register Diagram

Table 26-35: ADCC_EVTEN_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/W1C)	EVT[nn]	Event n Enable Clear. The ADCC_EVTEN_CLR.EVT[nn] bits permit clearing individual bits in the ADCC_EVTEN register without affecting other bits in the register. Write 1 to individual ADCC_EVTEN_CLR.EVT[nn] bits to clear the corresponding bit in ADCC_EVTEN.

Event Enable Set Register

The ADCC_EVTEN_SET register can be used to selectively set bits in the ADCC_EVTEN register without affecting other bits in the register. Writing a 1 to any bit position in ADCC_EVTEN_SET sets the corresponding bit in ADCC_EVTEN. Reading the ADCC_EVTEN_SET register returns the data present in the ADCC_EVTEN register.

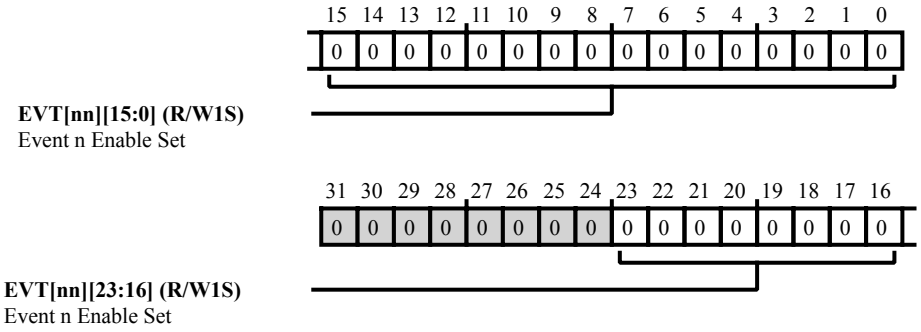

Figure 26-41: ADCC_EVTEN_SET Register Diagram

Table 26-36: ADCC_EVTEN_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/W1S)	EVT[nn]	Event n Enable Set. The ADCC_EVTEN_SET.EVT[nn] bits permit setting individual bits in the ADCC_EVTEN register without affecting other bits in the register. Write 1 to individual ADCC_EVTEN_SET.EVT[nn] bits to set the corresponding bit in ADCC_EVTEN.

Event n Time Register

The `ADCC_EVT[nn]` register holds the time (in terms SYSCLK cycles) at which to sample (the event) from ADC.

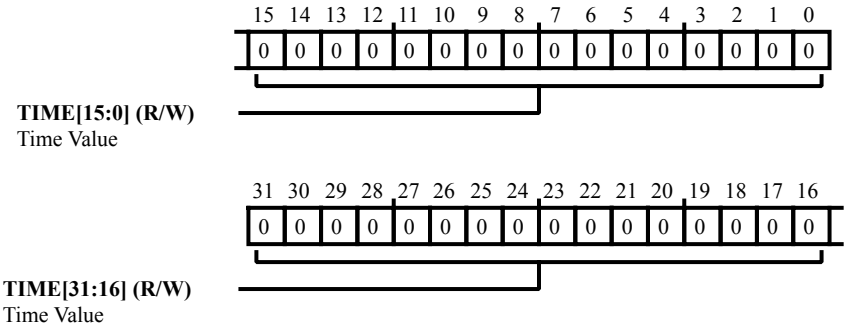

Figure 26-42: ADCC_EVT[nn] Register Diagram

Table 26-37: ADCC_EVT[nn] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TIME	Time Value. The `ADCC_EVT[nn].TIME` bits holds the time value (in terms SYSCLK cycles) at which to sample (the event) from ADC.

Frame Interrupt Mask Register

The ADCC_FIMSK register masks (disables) generation of ADCC_TMR0_EVT or ADCC_TMR1_EVT event interrupts based on status in the ADCC_FISTAT register.

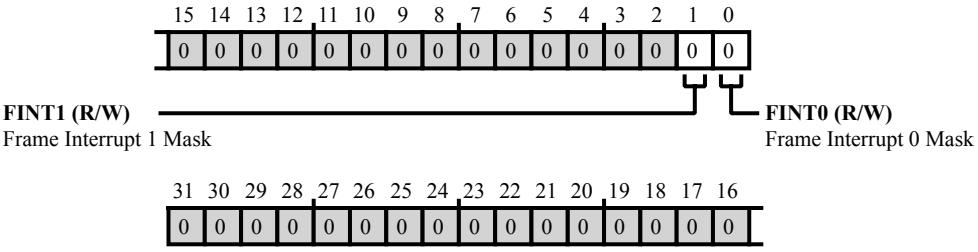

Figure 26-43: ADCC_FIMSK Register Diagram

Table 26-38: ADCC_FIMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	FINT1	Frame Interrupt 1 Mask. The ADCC_FIMSK.FINT1 bits mask (disable) the ADCC_FISTAT.FINT1 frame complete interrupts for data transfer related to Timer 1.	
		0	Unmask (Enable) Frame Interrupt
		1	Mask (Disable) Frame Interrupt
0 (R/W)	FINT0	Frame Interrupt 0 Mask. The ADCC_FIMSK.FINT0 bits mask (disable) the ADCC_FISTAT.FINT0 frame complete interrupts for data transfer related to Timer 0.	
		0	Unmask (Enable) Frame Interrupt
		1	Mask (Disable) Frame Interrupt

Frame Interrupt Mask Clear Register

The ADCC_FIMSK_CLR register can be used to selectively clear bits in the ADCC_FIMSK register without affecting other bits in the register. Writing a 1 to any bit position in ADCC_FIMSK_CLR clears the corresponding bit in ADCC_FIMSK. Reading the ADCC_FIMSK_CLR register returns the data present in the ADCC_FIMSK register.

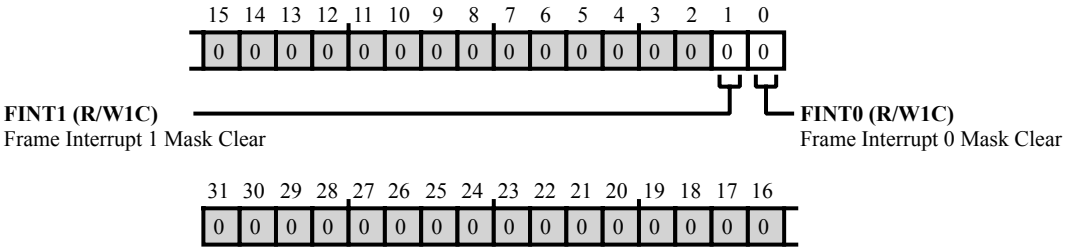

Figure 26-44: ADCC_FIMSK_CLR Register Diagram

Table 26-39: ADCC_FIMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	FINT1	Frame Interrupt 1 Mask Clear. The ADCC_FIMSK_CLR.FINT1 bits permit clearing the ADCC_FIMSK.FINT1 bit without affecting other bits in the register. Write 1 to the ADCC_FIMSK_CLR.FINT1 bit to clear the ADCC_FIMSK.FINT1 bit.
0 (R/W1C)	FINT0	Frame Interrupt 0 Mask Clear. The ADCC_FIMSK_CLR.FINT0 bits permit clearing the ADCC_FIMSK.FINT0 bit without affecting other bits in the register. Write 1 to the ADCC_FIMSK_CLR.FINT0 bit to clear the ADCC_FIMSK.FINT0 bit.

Frame Interrupt Mask Set Register

The ADCC_FIMSK_SET register can be used to selectively set bits in the ADCC_FIMSK register without affecting other bits in the register. Writing a 1 to any bit position in ADCC_FIMSK_SET sets the corresponding bit in ADCC_FIMSK. Reading the ADCC_FIMSK_SET register returns the data present in the ADCC_FIMSK register.

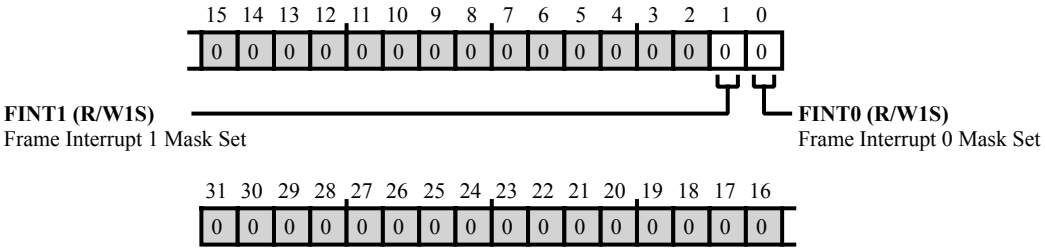

Figure 26-45: ADCC_FIMSK_SET Register Diagram

Table 26-40: ADCC_FIMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1S)	FINT1	Frame Interrupt 1 Mask Set. The ADCC_FIMSK_SET.FINT1 bits permit setting the ADCC_FIMSK.FINT1 bit without affecting other bits in the register. Write 1 to the ADCC_FIMSK_SET.FINT1 bit to set the ADCC_FIMSK.FINT1 bit.
0 (R/W1S)	FINT0	Frame Interrupt 0 Mask Set. The ADCC_FIMSK_SET.FINT0 bits permit setting the ADCC_FIMSK.FINT0 bit without affecting other bits in the register. Write 1 to the ADCC_FIMSK_SET.FINT0 bit to set the ADCC_FIMSK.FINT0 bit.

Frame Interrupt Status Register

The ADCC_FISTAT register indicates frame interrupt status. The ADCC generates interrupts corresponding to Timer 0 on the ADCC_TMR0_EVT output and generates the interrupts corresponding to Timer 1 on the ADCC_TMR1_EVT output.

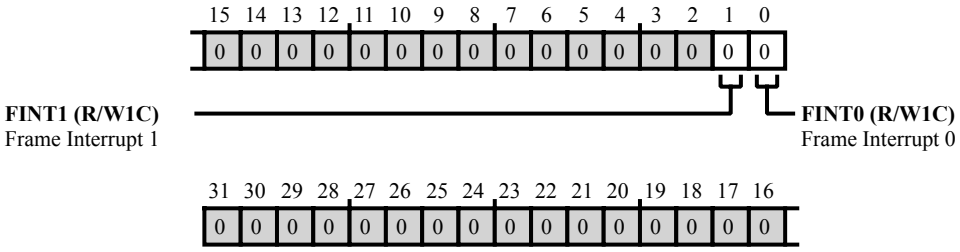

Figure 26-46: ADCC_FISTAT Register Diagram

Table 26-41: ADCC_FISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	FINT1	Frame Interrupt 1. The ADCC_FISTAT.FINT1 bit indicates core/DMA interrupt status for a Timer 1 frame. In non-DMA mode, this bit is set when the data corresponding to all events (which were not missed) in a frame are received by the ADCC (from the ADC). In DMA mode, this bit is set when all data corresponding to events of a frame are successfully DMA transferred by the ADCC (and responses received).	
		0	No Status
		1	Frame Complete
0 (R/W1C)	FINT0	Frame Interrupt 0. The ADCC_FISTAT.FINT0 bit indicates core/DMA interrupt status for a Timer 0 frame. In non-DMA mode, this bit is set when the data corresponding to all events (which were not missed) in a frame are received by the ADCC (from the ADC). In DMA mode, this bit is set when all data corresponding to events of a frame are successfully DMA transferred by the ADCC (and responses received).	
		0	No Status
		1	Frame Complete

Frame Increment 0 Register

The ADCC_FRINC0 register contains the address increment applied between the base-address of consecutive Timer 0 frames of DMA data (unless a fresh write of the ADCC_BPTR0 register occurred in between). The value is a signed, two's complement byte address increment.

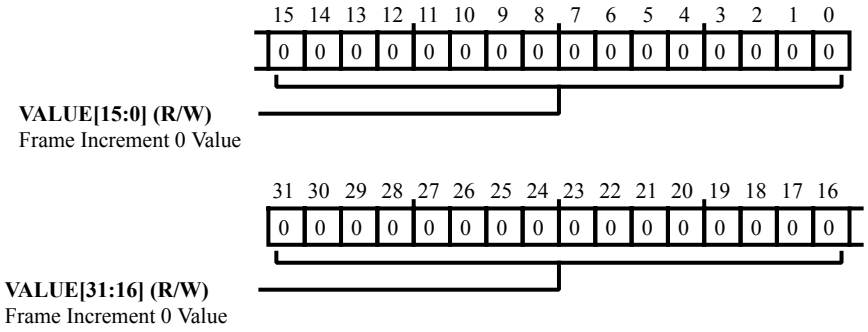

Figure 26-47: ADCC_FRINC0 Register Diagram

Table 26-42: ADCC_FRINC0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Frame Increment 0 Value. The ADCC_FRINC0.VALUE bits hold memory offset increment applied to the base-address of consecutive Timer 0 frames of DMA data. Note that bit 0 must be written as 0 to ensure correct address alignment.

Frame Increment 1 Register

The ADCC_FRINC1 register contains the address increment applied between the base-address of consecutive Timer 1 frames of DMA data (unless a fresh write of the ADCC_BPTR0 register occurred in between). The value is a signed, two's complement byte address increment.

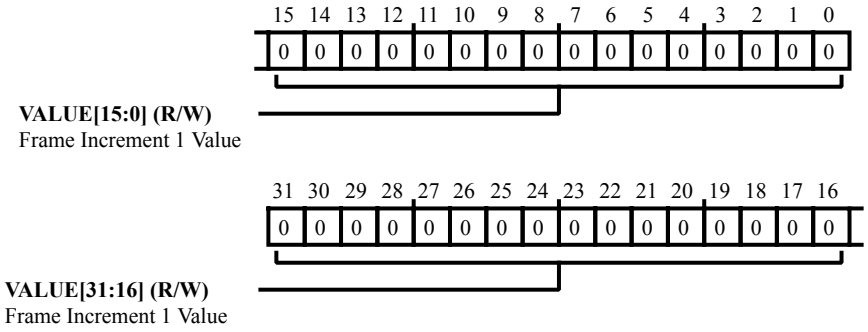

Figure 26-48: ADCC_FRINC1 Register Diagram

Table 26-43: ADCC_FRINC1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Frame Increment 1 Value. The ADCC_FRINC1.VALUE bits hold memory offset increment applied to the base-address of consecutive Timer 1 frames of DMA data. Note that bit 0 must be written as 0 to ensure correct address alignment.

Timer 0 Status Register

The ADCC_T0STAT register indicates Timer 0 related status for the ADCC operation.

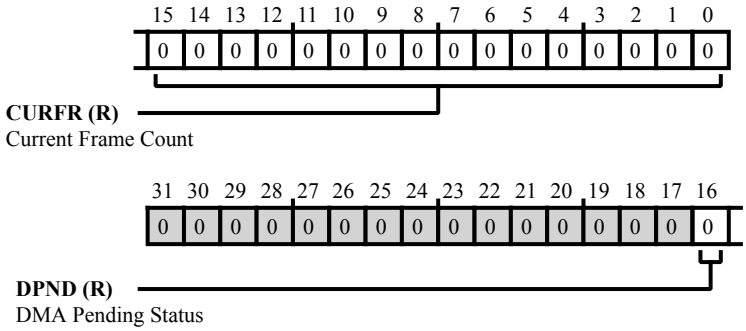

Figure 26-49: ADCC_T0STAT Register Diagram

Table 26-44: ADCC_T0STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/NW)	DPND	DMA Pending Status. The ADCC_T0STAT.DPND bit indicates pending status for data received from the ADC. When there are pending transfers, even if the ADCC is disabled while they are pending, the DMA finishes these pending data transfers.	
		0	No Status
		1	DMA Pending
15:0 (R/NW)	CURFR	Current Frame Count. The ADCC_T0STAT.CURFR bits hold the current frame number in the chronological order after ADCC enable occurred (linear buffer mode) or hold the frame number after the last wrap around (circular buffer mode). This field wraps over after 0xFFFF frames.	

Timer 1 Status Register

The ADCC_T1STAT register indicates Timer 1 related status for the ADCC operation.

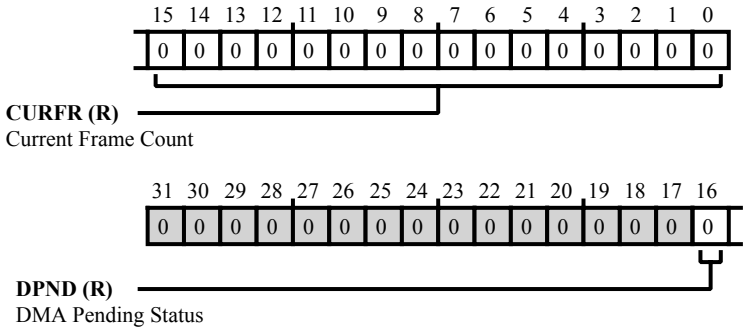

Figure 26-50: ADCC_T1STAT Register Diagram

Table 26-45: ADCC_T1STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/NW)	DPND	DMA Pending Status. The ADCC_T1STAT.DPND bit indicates pending status for data received from the ADC. When there are pending transfers, even if the ADCC is disabled while they are pending, the DMA finishes these pending data transfers.	
		0	No Status
		1	DMA Pending
15:0 (R/NW)	CURFR	Current Frame Count. The ADCC_T1STAT.CURFR bits hold the current frame number in the chronological order after ADCC enable occurred (linear buffer mode) or hold the frame number after the last wrap around (circular buffer mode). This field wraps over after 0xFFFF frames.	

Timing Control A (ADC0) Register

The ADCC_TCA0 register controls timing related to the ADC0 interface clock and chip select signals.

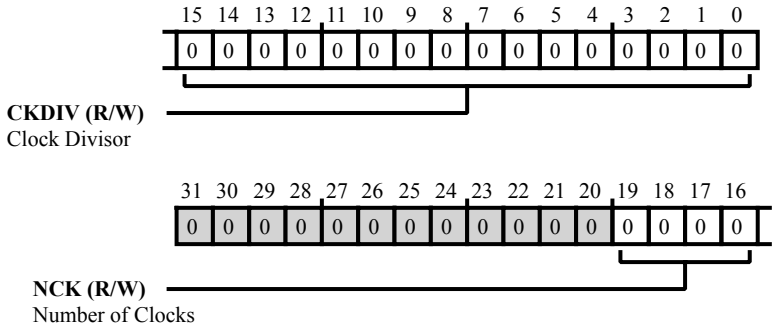

Figure 26-51: ADCC_TCA0 Register Diagram

Table 26-46: ADCC_TCA0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:16 (R/W)	NCK	Number of Clocks. The ADCC_TCA0.NCK bits select the number of ADCC_ACLK ADC clock cycles for each ADCC_ACS chip select pulse. A value of 0 implies 16 clock cycles. The ADCC_TCA0.NCK programming should ensure that total number of data/control bits is less than or equal to 16 bits. If either data or control are on dual bit lines (ADCC_CTL.DSIZE =1), the maximum number of clock permitted is 8, and NCK=0 is not permitted.
15:0 (R/W)	CKDIV	Clock Divisor. The ADCC_TCA0.CKDIV bits select the clock divisor ratio (SYSCLK:ACK) for clocks to be sent to ADC calculated as: ACK frequency = (SYSCLK frequency) / (CKDIV+1) Yielding, ADCC_TCA0.CKDIV =0 represents a ratio of 1:1, =1 represents a ratio of 1:2, ADCC_TCA0.CKDIV =2 represents a ratio of 1:3, and so on.

Timing Control A (ADC1) Register

The ADCC_TCA1 register controls timing related to the ADC1 interface clock and chip select signals.

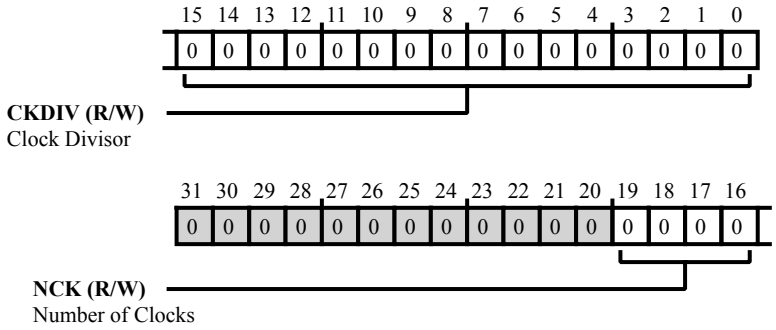

Figure 26-52: ADCC_TCA1 Register Diagram

Table 26-47: ADCC_TCA1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:16 (R/W)	NCK	Number of Clocks. The ADCC_TCA1.NCK bits select the number of ADCC_ACLK ADC clock cycles for each ADCC_ACS chip select pulse. A value of 0 implies 16 clock cycles. The ADCC_TCA1.NCK programming should ensure that total number of data/control bits is less than or equal to 16 bits. If either data or control are on dual bit lines (ADCC_CTL.DSIZE =1), the maximum number of clock permitted is 8, and NCK=0 is not permitted.
15:0 (R/W)	CKDIV	Clock Divisor. The ADCC_TCA1.CKDIV bits select the clock divisor ratio (SYSCLK:ACK) for clocks to be sent to ADC calculated as: ACK frequency = (SYSCLK frequency) / (CKDIV+1) Yielding, ADCC_TCA1.CKDIV =0 represents a ratio of 1:1, =1 represents a ratio of 1:2, ADCC_TCA1.CKDIV =2 represents a ratio of 1:3, and so on.

Timing Control B (ADC0) Register

The ADCC_TCB0 register controls parameter values for timing relationships between the ADCC_ACS and ADCC_ACLK pin signals in the ADC0 interface. For a timing diagram illustrating these timing relationships, see the ADCC functional description.

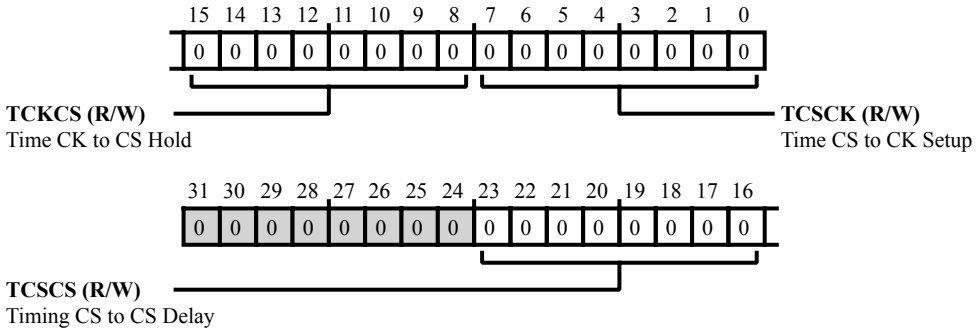

Figure 26-53: ADCC_TCB0 Register Diagram

Table 26-48: ADCC_TCB0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	TCSCS	Timing CS to CS Delay. The ADCC_TCB0.TCSCS value selects minimum delay time (in ADCC_ACLK cycles) required between on ADCC_ACS de-asserted edge and the next ADCC_ACS asserted edge. A value of 0 implies 256 cycles delay.
15:8 (R/W)	TCKCS	Time CK to CS Hold. The ADCC_TCB0.TCKCS value selects the minimum hold time (in ADCC_ACLK cycles) after (ADCC_TCB0.TCSCK + ADCC_TCA0.NCK) clock cycles have elapsed during ADCC_ACS asserted for which the chip select should be held asserted. A value of 0 implies 0 clock cycle delay.
7:0 (R/W)	TCSCK	Time CS to CK Setup. The ADCC_TCB0.TCSCK value selects the minimum setup time (in ADCC_ACLK cycles) required from the ADCC_ACS asserted to the first edge of the ADCC_ACLK clock. A value of 0 implies 256 clock cycles.

Timing Control B (ADC1) Register

The ADCC_TCB1 register controls parameter values for timing relationships between the ADCC_BCS and ADCC_BCLK pin signals in the BDC0 interface. For a timing diagram illustrating these timing relationships, see the ADCC functional description.

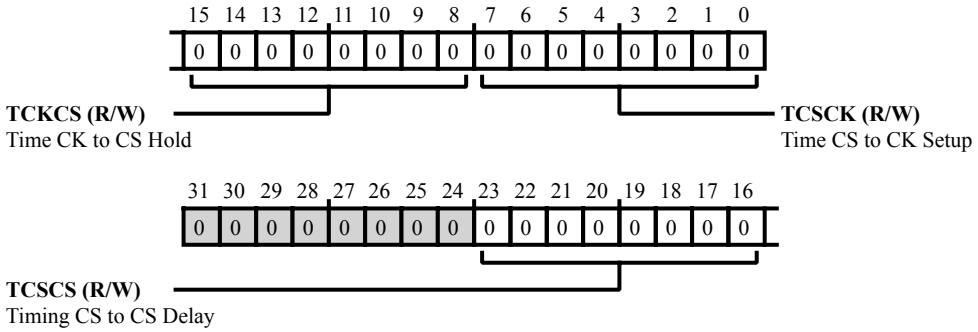

Figure 26-54: ADCC_TCB1 Register Diagram

Table 26-49: ADCC_TCB1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	TCSCS	Timing CS to CS Delay. The ADCC_TCB1.TCSCS value selects minimum delay time (in ADCC_BCLK cycles) required between on ADCC_BCS de-asserted edge and the next ADCC_BCS asserted edge. A value of 0 implies 256 cycles delay.
15:8 (R/W)	TCKCS	Time CK to CS Hold. The ADCC_TCB1.TCKCS value selects the minimum hold time (in ADCC_BCLK cycles) after (ADCC_TCB1.TCSCK + ADCC_TCA1.NCK) clock cycles have elapsed during ADCC_BCS asserted for which the chip select should be held asserted. A value of 0 implies 0 clock cycle delay.
7:0 (R/W)	TCSCK	Time CS to CK Setup. The ADCC_TCB1.TCSCK value selects the minimum setup time (in ADCC_BCLK cycles) required from the ADCC_BCS asserted to the first edge of the ADCC_BCLK clock. A value of 0 implies 256 clock cycles.

Timer 0 Current Count Register

The ADCC_TMR0 register holds the current count of the Timer 0 while a frame is in progress. This register may be used to determine the current timer count, which indicates approximately how much of the frame has been completed. The timer is initiated by the trigger and counts up starting from 0. The timer increments every SYSCLK cycle until the frame completion. In DMA mode, the frame completion happens when event all data have been placed in memory and responses are received. In non-DMA mode, the frame completion happens when all events' data have been received from the ADC. If a frame is not in progress, the ADCC_TMR0.CNT field is reset.

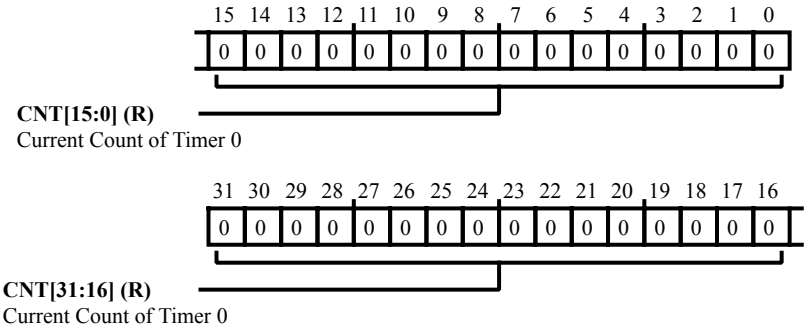

Figure 26-55: ADCC_TMR0 Register Diagram

Table 26-50: ADCC_TMR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Current Count of Timer 0. The ADCC_TMR0.CNT bits hold the current count of Timer 0.

Timer 1 Current Count Register

The ADCC_TMR1 register holds the current count of the Timer 1 while a frame is in progress. This register may be used to determine the current timer count, which indicates approximately how much of the frame has been completed. The timer is initiated by the trigger and counts up starting from 0. The timer increments every SYSCLK cycle until the frame completion. In DMA mode, the frame completion happens when event all data have been placed in memory and responses are received. In non-DMA mode, the frame completion happens when all events' data have been received from the ADC. If a frame is not in progress, the ADCC_TMR1.CNT field is reset.

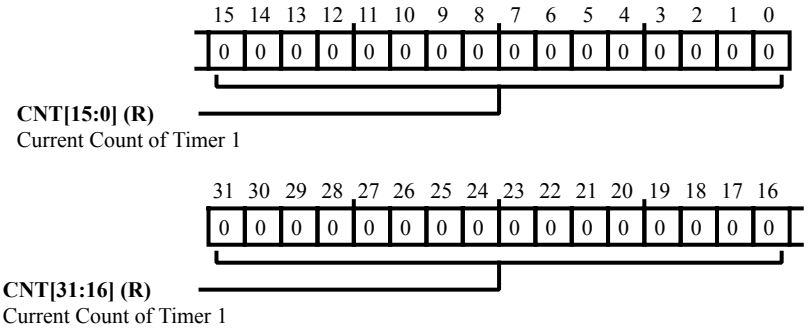

Figure 26-56: ADCC_TMR1 Register Diagram

Table 26-51: ADCC_TMR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Current Count of Timer 1. The ADCC_TMR1.CNT bits hold the current count of Timer 1.

27 Digital-to-Analog Converter Controller (DACC)

The analog front end (AFE) includes a powerful DAC controller (DACC), which automates DAC data conversion and simplifies DAC accesses. The DACC provides an interface that synchronizes the controls between the processor and a digital-to-analog converter (DAC).

NOTE: The DACC and ADCC chapters describe the control and data interface to the AFE. For information about the analog portion (I/O pins and electrical specifications) of the AFE, see the product data sheet.

Some processors do *not* include a dedicated DAC controller. In this situation, DAC conversion uses processor interrupts (initiated by the events) and interrupt service routine programming of the appropriate peripheral to initiate the conversion process. (The peripheral is usually a general-purpose timer). This approach has some limiting factors:

- DAC conversion processing is not precisely controlled due to interrupt latencies (which can vary) due to variable instruction execution cycles or multiple interrupts running in system.
- Consumption of processor MIPS can be prohibitive, especially for high frequency of conversion-related events.

The DACC addresses some of these limitations by providing dedicated hardware, which initiates the DAC output by providing control signals with required timings in real time. Using the DACC permits flexible scheduling of data transfer and provides precise control execution of timing and analog output on the DACs. The DACC saves both processor MIPS and provides precise controllability for DAC conversion or output time.

On the ADSP-CM40x microcontroller, the DACC is designed to interface with the on-chip internal DACs, which require minimal core intervention.

DACC Features

The DACC provides many architecture-based features (basic to the design) and mode-selectable features (usage is optional or configurable).

Architecture-based features of the DACC include:

- Two DAC interfaces to control two 12-bit ADCs independently
- Automated DAC conversion with programmable timing

- Support of both core mode and DMA mode for updating the DAC buffer. The DMA interface width can be programmable as a 16-bit or 32-bit interface.
- Separate four-deep FIFO for each DAC interface to queue the data for conversion, reducing the data request rate to the core or DMA
- Serial clock, frame sync select, and data signal to control the DAC operations
- Internally generated DAC clock from processor system clock

  This clock can be is gated (for example, it is active only when controlling the DAC) to provide excellent noise immunity during the conversion process. The clock polarity (for example, the first edge after assertion of the frame sync signal) is configurable.
- Two built-in DMA units

  There is one DMA unit for each DAC for DMA transferring the data for conversion. The DMA units support an optional mechanism for circular buffering.
- Error detection capabilities, including support for detection of memory access error, DAC data underflow, and address alignment error conditions

  The DACC does not perform a data transmission in case of underflow. An underflow interrupt can be signaled to the processor core.

Mode-selectable features of the DACC include:

- Gated clock signal with configurable polarity

  The DAC clock given to a DAC can be a continuous clock or can be gated when the DAC data is not driven. The clock edge polarity (that is, the driving edge of sync and data) can be configured to the rising edge or the falling edge.
- Frame sync signal with configurable width and polarity

  The DAC frame sync signal has a programmable width in terms of the DAC clocks. The polarity of this signal is configurable as active high or active low signal. The DAC uses this signal to start the conversion, and to output the converted data.
- Timing for the DAC update frequency is configurable
- Data length for the DAC interface is programmable for up to 16-bit data lengths
- Data transmission to the DAC in LSB-first or MSB-bit first format
- Circular buffering for DMA data

  When using DMA mode to update the DAC FIFO, the buffer submitted to DMA can be configured for circular buffering to transmit the data continuously without core intervention. Optionally, an interrupt can be enabled to signal core at the end of each circular buffer.
- Status indication (status bits) and optional interrupt generation to core on DAC data underflow

- Optional generation of interrupt on completion of each DMA work unit, DMA circular buffer wraparound, or on DACC error detection

DACC Functional Description

The following sections describe DACC functionality:

- ADSP-CM40x DACC Register List
- ADSP-CM40x DACC Interrupt List
- DACC Block Diagram
- DACC Signal Descriptions
- DACC Architectural Concepts

NOTE: Analog front end (AFE) start-up requires the loading of some initialization settings. This initialization happens automatically when the ADCC is enabled. Therefore, enable the ADCC before enabling the DACC after power-on reset. If continued operation of the ADCC is unnecessary, it can be disabled while continuing operation of the DACC.

DACC Signal Descriptions

The DACC controls the operation of internal DACs, based on its timing register settings. Because the DACs are internal, none of the DACC signals are available as external package pins.

NOTE: There are two instances of the DAC interface on the ADSP-CM40x. Each offers separate clock, frame sync, and data signals for controlling two DACs independently.

A number of signals connect the DACC to the DACs for DAC clock, frame sync, and data. Using these signals, the DACC regulates the DAC data transfer and data conversion. These signals appear in the ***ADSP-CM40x DACC-to-DAC0/1 Signal Descriptions*** table.

Table 27-1: ADSP-CM40x DACC-to-DAC0/1 Signal Descriptions

Name	I/O	Description
DACC_ACLK	O	DACC DAC0 (A) clock
DACC_AFS	O	DACC DAC0 (A) frame sync
DACC_AD0	O	DACC DAC0 (A) data 0 for writing conversion data to the DAC
DACC_BCLK	O	DACC DAC1 (B) clock
DACC_BFS	O	DACC DAC1 (B) frame sync
DACC_BD0	O	DACC DAC1 (B) data 0 for writing conversion data to the DAC

The DACC also has a number of signals to send the status and error information to the processor. These signals appear in the ***ADSP-CM40x DACC-to-Core Signal Descriptions*** table.

Table 27-2: ADSP-CM40x DACC-to-Core Signal Descriptions

Signal Name	I/O	Signal Description
DACC_DAC0	O	DAC0 Event when a DAC0 frame is completed
DACC_DAC1	O	DAC1 Event when a DAC1 frame is completed
DACC_ERR	O	DAC Error Interrupt

The following are more detailed descriptions of each DACC signal type.

DAC Clock (DACC_ACLK, DACC_BCLK)

This clock signal is given to DAC, based on which of the other two DAC signals, frame sync and data are driven. The clock edge polarity (that is, the driving edge of CS or SYNC and data) can be configured to rising edge or falling edge using DACC clock polarity bit setting.

The DAC clock signal is internally generated from processor SYSCLK and is configurable as follows:

```
DAC_CLK frequency = SYSCLK / (DACC_TC.DCKDIV + 1)
```

The DAC clock can be given continuous free running or gated (that is, given out only when data is driven). This clock configuration uses the DACC gated clock enable bit setting.

DAC Frame Sync (DACC_AFS, DACC_BFS)

The DAC frame sync signal is active when valid data is driven out to DAC. The DACC uses the data length field (for example, DACC_CTL0.DLEN) to decide the active period of the DAC frame sync.

The frequency of this signal determines the DAC update rate, which can be programmed using DACC frame sync idle field (for example, DACC_TC0.FSIDLE). The DACC uses the frame sync idle field to decide the idle period between two DAC sync active pulses. This field is programmed in terms number of DAC clocks. It is a 16-bit field and a value of zero implies 1 DAC clock cycle.

Therefore, the periodicity of `DACC_FS = (DACC_TC.DFSIDLE+1) + DACC_CTL.DLEN`.

If the DACC data length is zero, periodicity of `DAC_FS = (DACC_TC.DFSIDLE+1) +16.`

The polarity of the DAC sync signal can be configured to active high or active low, using the DACC sync polarity bit setting (for example,DACC_CTL0.SYNCPOL).

DAC Data (DACC_AD0, DACC_BD0)

The data to the DAC is serially driven out on DACC data pin (for example, DACC_AD0).

The DACC data length field (for example, DACC_CTL0.DLEN) determines the DAC interface length. It is a four-bit field, allowing DAC interface length to be from 1 to 16 (value of 4'h0 implies a data length of 16 bits).

The DAC data can be transmitted in LSB first format or MSB bit first format, by configuring the DACC LSB first bit (for example, `DACC_CTL0.LSBF`).

On the ADSP-CM40x processor, the DACC controls the 12-bit DACs, which support MSB first format. So, configure the DACC data length to 0xC, and the DACC LSB first bit to 0.

There can be situations where an incomplete or larger DAC frame is sent to the DAC when the data length field is set incorrectly. In these scenarios, the DAC on the ADSP-CM40x processor ignores frames of lower than 11-bits size and discards extra bits if the frame sizes are more than 12 bits.

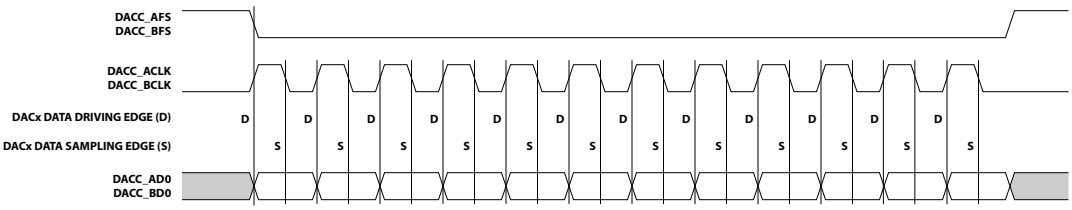

Figure 27-1: DACC Signals

DACC Architectural Concepts

The DACC Functional Block Diagram shows the top-level architecture of DAC interface. The interface consists of:

- A DMA and core interface - This interface permits the core to read and write DACC registers for configuration, control, and word-by-word data transfer. The independent DMA engine of the DACC provides data transfer without core overhead. For more information about this interface, see Core and DMA Interfaces and Data Transfer Modes.
- DAC pending data FIFOs - Each DAC has a four-deep FIFO for pending data transfer. These FIFOs help prevent data underrun. For more information, see Pending Data FIFO.
- DAC clock generation - Each DAC has an independent clock signal that the DACC generates from the system clock (SYSCLK) according to the configuration in DACC registers. Optionally, this clock can be gated (only active while data is driven). For more information, see Clock Modes.
- Frame sync generation - Each DAC has an independent frame sync signal that controls the data transfer rate. Data and the gated clock are only driven while the frame sync is asserted. For more information, see Frame Sync Modes.
- Data output (shift registers) - The last stage of each DAC FIFO is a serial shift register, which the DACC uses to transfer data from the FIFO to the DAC.
- Status and error interrupts - The DACC has an error interrupt output, and the DACs each have a status interrupt output. These outputs go to the SEC of the processor for handling. For more information, see DACC Event Control.

Core and DMA Interfaces

The DACC has a 32-bit core interface through which the core programs the DACC control registers and reads the DACC status registers. The DACC also uses this interface to update the DAC FIFOs in core mode. It updates the DAC FIFO (with data for digital-to-analog conversion) by writing data to the data register of the DAC (for example, DACC_DAT0).

To minimize the core overhead, the DACC provides a 32-bit DMA interface for updating the DAC FIFOs. The DMA data interface width can be programmed to 16-bit or 32-bit width (for example, using the DACC_CTL0.DMAWbit).

The DMA interface supports an optional circular buffering mode for updating the DAC FIFOs. When circular buffered is enabled (for example, using the DACC_CTL0.CBUFEN bit), the DMA continuously transmits the data without core intervention. An interrupt can be enabled (for example, using the DACC_CTL0.CINTEN bit) to signal the core at the end of each circular buffer.

Pending Data FIFO

The DACC provides a separate, 4-deep FIFO for each DAC interface. It updates the FIFOs in core mode or DMA mode, according to the mode settings in the DACC control register (for example, DACC_CTL0).

The pending data FIFO helps reduce the data request rate to core or DMA and reduces possibility of underflow.

The DAC shift register (at the last FIFO stage) reads the data from the DACC FIFO and serially shifts out the data. It shifts the data based on active edges of the DAC clock (for example, DACC_ACLK) when the DAC sync pulse (for example, DACC_AFS) is active.

An *underflow* at the DAC interface occurs when the DAC frame sync is about to go active, but the FIFO of the DAC has no new data for transmission. When this situation occurs, the DACC does not generate the frame sync pulse. Optionally, the DACC can signal this condition to the core with an error interrupt.

DACC Operating Modes

The operating modes of the DACC include its configurable options for data length, FIFO update rate, DAC clock divider and polarity, frame sync idle time and polarity, and broadcast controls. For more information about these options, see the following sections:

- Data Transfer Modes
- Data Length and Update Options
- Clock Modes
- Frame Sync Modes
- Broadcast Control Option

Data Transfer Modes

The DACC transfers data to the DACs through the corresponding FIFO. To update these FIFOs with data for transmission to the corresponding DAC, the DACC supports:

- Core-driven single word transfers
- DMA-driven multiple words transfers

DMA transfers can be set up to transfer a configurable number of words from internal or external memory of processor to the DACC FIFOs automatically, without core intervention. Core-driven transfers can use DACC interrupts to signal the processor core to perform single word transfers to the DACC FIFOs.

Core-Driven Data Write Mode

The DACC provides a DAC data register (for example, DACC_DAT0), for accessing the top entry of the 4-deep FIFO in core mode. The DMA enable bit (for example, DACC_CTL0.DMAEN) selects the data transfer mode of DACC operation as either DMA mode or core mode. Typically, the core checks the FIFO status bits (for example, DACC_STAT.FSTAT0) or relies on related interrupt status before writing into DAC data register. This register must be written only when there is space available in the DAC FIFO. If the core attempts a write into the data register when the DAC FIFO is full, the DACC ignores the core write transaction.

The DACC provides a way to ease status-checking for core write operations. It uses the core-write complete status bit (for example,DACC_ISTAT.CINT0) to indicate whether the DAC FIFO has space available to accommodate new data writes by the core. Optionally, an interrupt for the data request condition can be unmasked (enabled) in the DACC_IMSK register. Then, core writes to the DAC FIFO can be managed with an interrupt service routine.

NOTE: The DAC interface considers only the lower 16 bits of DAC data FIFO register as DAC data. It ignores the upper 16 bits. Reads of this register return the top entry of the FIFO.

DMA-Driven Data Write Mode

The DACC provides a 32-bit in-built DMA interface to minimize the core overhead for updating the DAC data FIFOs. The DMA enable bit (for example, DACC_CTL0.DMAEN) selects the data transfer mode of DACC operation as either DMA mode or core mode. The DMA data interface width bit (for example, DACC_CTL0.DMAW) selects whether the interface is 16 bits or 32 bits wide.

NOTE: When the DAC is in DMA mode, core writes to DAC data register do not result in the data being written into the DACC FIFO, and these writes are ignored.

The DMA unit supports programming of DMA work-units using following registers.

- DAC DMA base pointer register (for example, DACC_BPTR0) - This register contains the memory address of the base pointer for starting a DMA read transfer.
- DAC DMA modify register (for example, DACC_MOD0) - This register contains the address-increment applied between each DMA read from memory, starting from base pointer.

- DAC DMA count register (for example, DACC_CNT0) - This register holds the DMA count in a DMA work-unit.

NOTE: On the ADSP-CM40x processor, the DACC controls two independent DACs. The DACC provides a separate DMA unit (with a pointer, modifier, and count register) for each DAC interface.

The DMA unit supports ***linear buffering*** or ***circular buffering*** modes of operation. The circular buffer enable bit (for example, `DACC_CTL0.CBUFEN`) selects between these modes of operation.

Linear Buffering

In ***linear buffering***, the data transfer starts from the memory location pointed to by the base pointer register. With each data fetch (16 bits or 32 bits), the address is incremented by the number of bytes specified in modifier register to calculate the next fetch address. The DMA fetches data the number of times programmed in the count register and transmits the data to the DAC. After all the data is transmitted, an interrupt status (optionally) is set.

Circular Buffering

In ***circular buffering***, the data transfer starts from the memory location pointed to by the base pointer register. With each data fetch (16 bits or 32 bits), the address is incremented by the number of bytes specified in the modifier register to calculate the next fetch address. The DMA unit provides a looping back mechanism to the base pointer address. After the number of data fetches pointed by the count register, the next fetch is performed from the base pointer location, instead of incrementing the fetch address by the modifier.

When interrupts are disabled in circular buffer mode (for example, using the `DACC_CTL0.CINTEN` bit), the data fetches of next circular buffer are initiated. The data fetch occurs even if the previous circular buffer has data-reads pending the return from memory. If interrupts are enabled, the next circular buffer data request is made only after the reads of previous circular buffer return. The interrupt is signaled after the reads of a circular buffer return from memory.

Data Length and Update Options

The data length of DAC interface can be configured, allowing the DACC to interface with serial DACs with 1-bit to 16-bit data length. The DACC can send data to the DAC in either LSB-first format or MSB-first format, based on the DACC LSB-first bit setting.

The DACC uses the DAC data length bit field (for example, `DACC_CTL0.DLEN`) to choose up to a 16-bit data length for the DAC interface. This value affects the number of DAC clock cycles for each data word and affects the rate for DAC FIFO update.

The period of DAC FIFO update is affected through the DAC frame sync signal (for example, `DACC_AFS`) configuration for frame sync minimum idle time (for example, `DACC_TC0.FSIDLE`). The DAC clock divisor ratio (for example, `DACC_TC0.CKDIV`) also affects the period of the DAC FIFO update.

The following calculation describes the combination of these factors for the DAC FIFO update period (example DAC0 sync period) in DAC clock cycles:

DAC0 sync period = (DACC_TC0.FSIDLE + 1) + (DACC_CTL0.DLEN)

Applying this formula to calculate the required frame sync idle, assume:

- DAC data length is 12 bits
- System clock (SYSCLK) frequency is 100 MHz
- DAC0 clock frequency is 50 MHz
- DAC update frequency is 50 KSPS (kilo-samples-per-second)

First, calculate the DAC clock divisor (for DAC0) as:

DACC_TC0.CKDIV= (SYSCLK frequency / DAC0 clock frequency) – 1

= (100 MHz / 50 MHz) - 1 = 1

Then, calculate the needed DAC frame sync idle time (for DAC0) as:

DACC_TC0.FSIDLE = (DAC clock frequency / DAC update frequency) - DACC_CTL0.DLEN - 1

= (50 MHz / 50 KSPS) - 12 - 1 = (1000 - 12 - 1) = 987 = 0x3DB

Clock Modes

The DACC provides a clock signal to communicate with the interfaced DAC. The DAC clock also is available in gated format to ensure excellent noise immunity during the conversion process. (For example, the clock is active only during the time when data is driven to the DAC)

Clock Frequency Programming

The DAC clock is internally generated from system clock of processor. The DACC_TC0.CKDIV bit field specifies the divider to generate DAC0 clock signal from SYSCLK.

```
DACC_ACLK = (SYSCLK) / (DACC_TC0.CKDIV + 1)
```

Alternatively, the clock divisor value for the required DAC0 clock frequency is calculated as:

```
DACC_TC0.CKDIV = (SYSCLK ÷ DACC_ACLK) – 1
```

Clock Polarity and Gated Clock Programming

The active DAC clock edge (polarity) can be selected as rising edge or falling edge. The DACC sync and data signals (for example,DACC_AFS and DACC_AD0) are driven on the active edges of DAC clock. Configure the clock polarity bit (for example, DACC_CTL0.CKPOL) based on the clock edge the DAC samples the DAC sync and data signals. If the DAC clock is active low (for example, DACC_CTL0.CKPOL =0), the DAC sync and data signals are driven from the falling edge of the DAC clock (for example, DACC_ACLK).

Consider an example where the DAC sync signal is active low (for example, DACC_CTL0.SYNCPOL =0) and the DAC clock signal driving edge polarity is selected as rising edge (for example, DACC_CTL0.CKPOL =1). In

this example, the value of DAC data length is 12 (for example, DACC_CTL0.DLEN =12), and the value of DAC sync idle is 4 (for example, DACC_TC0.FSIDLE =4). (Refer to the ***DAC Clock Programming (DACC_CTLx.GCKEN =0)*** figure).

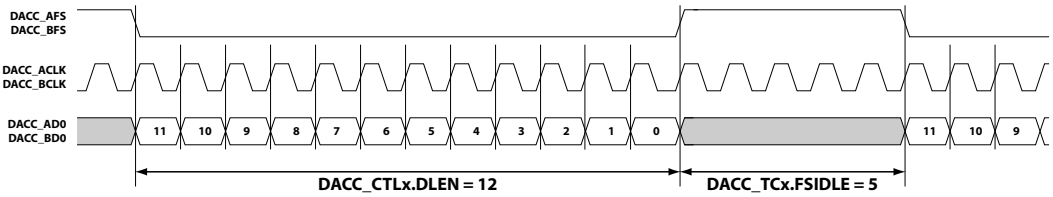

Figure 27-2: DAC Clock Programming (DACC_CTLx.GCKEN =0)

The DACC also provides the DAC clock in gated format, which is active only while the frame sync signal is asserted and valid DAC data is driven. This operation is enabled with the gated clock enable bit (for example, DACC_CTL0.GCKEN). The ***DAC Clock Programming (DACC_CTLx.GCKEN =1)*** figure shows the DAC serial timing for gated clock mode.

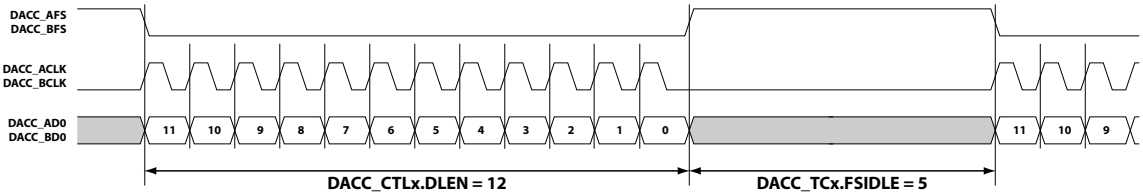

Figure 27-3: DAC Clock Programming (DACC_CTLx.GCKEN =1)

Frame Sync Modes

The DACC provides a frame select signal (for example, DACC_AFS) to select the DAC for communication. The signal (optionally with its edges) indicates the start of data transmission to the DAC (and data conversion). The frame select is asserted while data is driven to the DAC.

The frame sync signal provided to the DAC can be configured as an active-high signal or an active-low signal, based on the protocol the interfaced ADC supports. This sensitivity is configurable using the chip select polarity bit (for example, DACC_CTL0.SYNCPOL).

Between data write sequences to the DAC, when a valid data word is not driven to the DAC, the DAC sync lines must be driven to an inactive level. (The lines are driven to a high or a low level). This action meets the DAC requirement for the frame period. The value is selected with the frame sync idle bit field (for example, DACC_TC0.FSIDLE).

For more information about frame sync idle programming, see Data Length and Update Options.

Broadcast Control Option

The DACC controls multiple DACs. To support synchronized enable time of the DACs (if necessary), the DACC includes a broadcast control register (DACC_BCST_CTL), which provides broadcast write access to the control registers of the DAC. A memory mapped register write to DACC_BCST_CTL writes the value to the DACs control registers. Reading the DACC_BCST_CTL register returns 0x0000,0000.

NOTE: On the ADSP-CM40x, the DACC controls two DACs. Writes to the broadcast control register (DACC_BCST_CTL) go to both DACs control registers (DACC_CTL0 and DACC_CTL1).

DACC Event Control

The DACC can signal the core about its state and various error conditions that occur during its operation, by providing status and error bits through different registers. These conditions include:

- Interrupt status related to data FIFO operations in core mode and DMA mode
- Error status related to DACC operations
- Pending data status (which do not generate interrupts) related to data FIFO operations in core mode and DMA mode

Interrupt Status

The DACC can generate interrupts to signal operation status for individual DACs or error status for the DACC to the system event controller (SEC).

The DACC provides a data interrupt channel for each DAC FIFO (for example, DACC_DAC0). In core mode, this interrupt indicates the request to fill the DACC FIFO. In DMA mode, this interrupt indicates the completion of a DMA work unit. The DACC uses the conditions flagged in the interrupt status register (DACC_ISTAT) to generate these interrupts.

The DACC error interrupt signal (DACC_ERR) indicates error conditions related to DAC controller. The DACC uses the conditions flagged in the error status register (DACC_ERRSTAT) to generate these interrupts.

DAC DMA Mode Interrupt

In DMA mode, this interrupt indicates the status of DMA work-unit. When in linear DMA mode, the DACC can generate this interrupt when a DMA work unit programmed for a DAC interface completes, and all data has transmitted to DAC. When in circular buffer DMA mode (with the circular buffer interrupt enabled), the DACC can generate this interrupt when all read data in a buffer return from memory. The status bit related to this interrupt is sticky. A W1C operation clears the bit.

DAC Core Mode Interrupt

This interrupt indicates the status of DAC FIFO update in core mode, when the DAC data FIFO has space to accommodate new core data writes. The status bit related to this interrupt is a read-only status bit and is cleared when the FIFO gets full.

Error Status

The DACC signals error conditions through the error status register (DACC_ERRSTAT) for data underflow errors and memory access errors. The DACC can use these conditions to generate an error interrupt (DACC_ERR) when they are unmasked (enabled) in the error mask register (DACC_ERRMSK).

Clear all of these sticky error status bits with a W1C (write-1-to-clear) operation. Even if the DACC reports an error, normal operation of the DACC continues. Due to address alignment errors, the lower (violating) bits of the base pointer address or modifier registers are ignored. The data at the aligned address is written into the DAC FIFO. In the underflow error case, transmission of the DAC data is delayed until the DACC FIFO receives any data from the selected core or DMA interface.

Data Underflow Error

When an underflow occurs in a DAC FIFO, the related underflow error bit (for example, `DACC_ERRSTAT.DUVF0`) indicates the condition. For more information about underflow, see Pending Data FIFO.

Memory Access Error

When an address alignment error condition occurs while DACC DMA is enabled, the related address alignment error bit (for example, `DACC_ERRMSK_SET.AER0`) indicates the condition. An address alignment error occurs when:

- The LSB bit of the base pointer address register or modifier register is non-zero in 16-bit DMA mode.
- The lower 2 bits of the base pointer address register or modifier register are non-zero in 32-bit DMA mode.

When a read response error returns from memory for transfers to a DAC, the related memory access error bit (for example, `DACC_ERRSTAT.MER0`) indicates the condition. A read response error occurs when a DAC DMA attempts to access the reserved memory space of processor.

Pending Status

In addition to interrupt status and error status, the DACC provides bits to provide information about DAC FIFO pending data status.

DMA Data Pending Status

The DMA pending status bit (for example,`DACC_STAT.DPND0`) indicates that DMA read address requests have been made and the related read data is pending DAC interface reception. Before re-enabling the DMA of a DAC interface, check the corresponding pending status bit (for no data still pending).

DAC Data FIFO Status

The FIFO status bit (for example, `DACC_STAT.FSTAT0`) indicates the number of 16-bit data in the DAC data FIFO that await transmission to the DAC. The status increments when DMA mode transfers or core mode writes fill the DAC FIFO. The status decrements when a data transmission starts on the DAC interface.

DACC Programming Model

The programming model for the DACC includes operation flow for core mode transfers and DMA mode transfers. Use the following steps in these flows.

Core Mode Operation Flow

1. Set up the DACC, writing to its memory mapped registers in core mode (for example, `DACC_CTL0.DMAEN` =0 and `DACC_CTL0.EN` =0).
2. Enable DACC operation, writing the DACC_CTL.EN bit (=1) in core mode.

   Result: The DACC clock starts and sync counters begin. The DACC sends sync pulses only if new data is present. With each sync pulse, the DACC updates the DAC with new data.
3. Write DAC data to the DACC FIFO, using register writes in core mode.
4. When DAC updates complete, disable DACC operation (if desired), writing to the DACC_CTL.EN bit (=0) in core mode.

DMA Mode Operation Flow

1. Set up the DACC, writing to its memory mapped registers in core mode (DACC_CTL.DMAEN =1 and DACC_CTL.En =0)

   Result: DAC DMA starts reading data from memory.
2. Poll the FIFO status in DAC_STAT for FIFO >0 in core mode.
3. Enable DACC operation, writing the DACC_CTL.EN bit (=1) and DACC_CTL.DMAEN (=1) in core mode.

   Result: The DACC clock starts and sync counters begin. The DACC sends sync pulses only if new data is present. With each sync pulse, the DACC updates the DAC with new data during the selected DMA word count.
4. When DAC updates complete, disable DACC operation (if desired), writing to the DACC_CTL.EN and DACC_CTL.DMAEN bits (both =0) in core mode.

DACC Programming Concepts

There are general programming concepts for using the DACC when it is present on any processor.

NOTE: There can also be processor-specific guidelines for programming the DACC.

Take care when disabling or enabling the DACC in DMA modes. When disabling the DAC controller (for example, `DACC_CTL0.EN` =0), ensure that pending DMA transfers finish before re-enabling the DACC or re-enabling DMA

(for example, DACC_CTL0.DMAEN =1). This status can be observed from the related DMA pending bit (for example, DACC_STAT.DPND0) of the DAC. When disabling the DACC or DMA in the control register, retain the programming of all other DMA-related controls (for example, DACC_CTL0.CBUFEN). Change all DMA-related programming in DACC registers only after the corresponding DMA pending status bit is cleared (DMA completed).

The memory mapped register status and counter-values are retained on disabling the DACC (helpful for debug). These registers (such as counters and other status) are cleared on a 0-to-1 transition of the enable bit. Error status is not cleared with this transition of 0-to-1. Clear the errors using a W1C operation.

If the DAC controller is disabled while a transaction on the DAC interface is in-progress, the transaction can be dropped midway. Disable the DACC after a DMA work unit completes in DMA mode, or disable the DACC when all data updates are complete in non-DMA mode.

After a DMA work unit finishes (interrupt given), initiate a new work unit using the following procedure:

- Disable both the DACC enable and DMA enable bits (for example, DACC_CTL0.EN =0 and DACC_CTL0.DMAEN =0).
- Configure the DMA count and DMA modifier (for example, DACC_CNT0 and DACC_MOD0), then enable DMA operation (for example, DACC_CTL0.DMAEN =1).
- After the FIFO gets data from the DMA (for example, DACC_STAT.DPND0 =1), enable the DACC (for example, DACC_CTL0.EN =1).

DACC Programming Guidelines (ADSP-CM40x Specific)

There are general programming concepts for using the DACC when it is present on any processor. For the ADSP-CM40x, there are also processor-specific guidelines for programming the DACC.

The ADSP-CM40x processor includes two 12-bit on-chip internal DACs. For control, the DACC provides two DAC interfaces, which can be independently configured for the DAC conversion process.

The settings required for best performance of the DACs are:

- Select clock polarity as active high (for example, DACC_CTL0.CKPOL =1)
- Select frame sync polarity as active low (for example, DACC_CTL0.SYNCPOL =0)
- Select 12-bit data length (for example, DACC_CTL0.DLEN =0xC)
- Select MSB-first format (for example, DACC_CTL0.LSBF =0)
- Enable gated clock operation (for example, DACC_CTL0.GCKEN =1)

ADSP-CM40x DACC Register Descriptions

DAC Controller (DACC) contains the following registers.

Table 27-3: ADSP-CM40x DACC Register List

Name	Description
DACC_BCST_CTL	Broadcast (Write) Control Register
DACC_BPTR0	Base Pointer 0 Register
DACC_BPTR1	Base Pointer 1 Register
DACC_CNT0	Count 0 Register
DACC_CNT1	Count 1 Register
DACC_CNTCUR0	Current Count 0 Register
DACC_CNTCUR1	Current Count 1 Register
DACC_CTL0	Control 0 Register
DACC_CTL1	Control 1 Register
DACC_DAT0	Data FIFO 0 Register
DACC_DAT1	Data FIFO 1 Register
DACC_ERRMSK	Error Mask Register
DACC_ERRMSK_CLR	Error Mask Clear Register
DACC_ERRMSK_SET	Error Mask Set Register
DACC_ERRSTAT	Error Status Register
DACC_IMSK	Interrupt Mask Register
DACC_IMSK_CLR	Interrupt Mask Clear Register
DACC_IMSK_SET	Interrupt Mask Set Register
DACC_ISTAT	Interrupt Status Register
DACC_MOD0	Modify 0 Register
DACC_MOD1	Modify 1 Register
DACC_STAT	Status Register
DACC_TC0	Timing Control 0 Register
DACC_TC1	Timing Control 1 Register

Broadcast (Write) Control Register

The DACC_BCST_CTL register provides a broadcast write access to the DACC_CTL0 and DACC_CTL1 registers. A memory mapped register write to DACC_BCST_CTL writes the data to both control register. A memory mapped register read of the DACC_BCST_CTL register returns 0x0000.

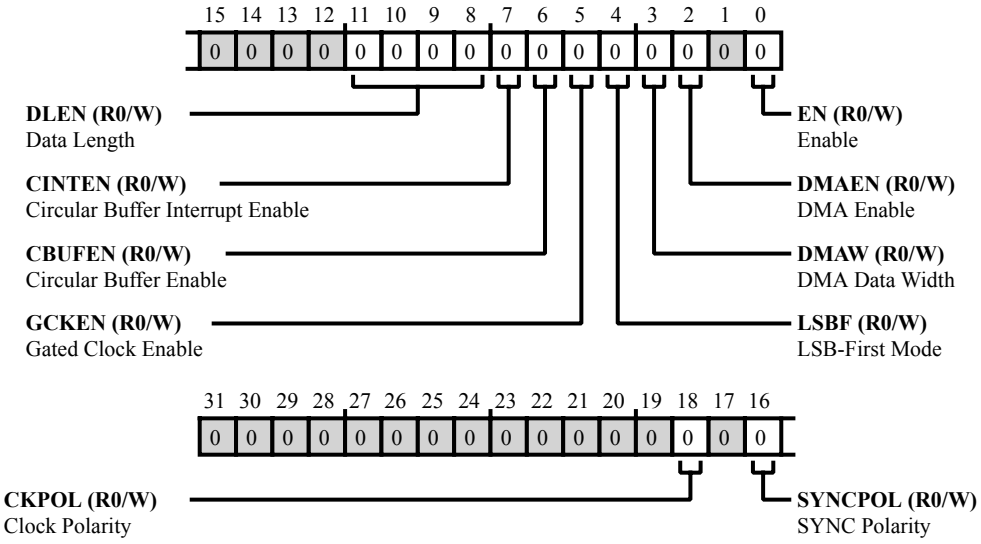

Figure 27-4: DACC_BCST_CTL Register Diagram

Table 27-4: DACC_BCST_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R0/W)	CKPOL	Clock Polarity. The DACC_BCST_CTL.CKPOL broadcast bit selects the polarity of the DAC0 and DAC1 interface clock signals (DACC_ACLK and DACC_BCLK pins) on which to drive the DAC0 output (DACC_AD0 pin) and/or DAC1 output (DACC_BD0 pin).
16 (R0/W)	SYNCPOL	SYNC Polarity. The DACC_BCST_CTL.SYNCPOL broadcast bit selects the polarity for the DAC0 and DAC1 interfaces' sync signals (DACC_AFS and DACC_BFS pins).
11:8 (R0/W)	DLEN	Data Length. The DACC_BCST_CTL.DLEN broadcast bits choose the data length (in bits) for the DAC0 and DAC1 interfaces. A value =0 for this field implies a data length of 16 bits. For data lengths less that 16 bits, use LSB-aligned data and zero fill unused bits.
7 (R0/W)	CINTEN	Circular Buffer Interrupt Enable. The DACC_BCST_CTL.CINTEN broadcast bit enables generation of the DACC_DAC0 and/or DACC_DAC1 interrupts at the end of each related circular buffer. This bit is only valid if the DACC_BCST_CTL.DMAEN bit =1 and the DACC_BCST_CTL.CBUFEN bit =1.

Table 27-4: DACC_BCST_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R0/W)	CBUFEN	Circular Buffer Enable. The DACC_BCST_CTL.CBUFEN broadcast bit enables circular buffer DMA mode for the DAC0 and DAC1 interfaces. This bit is only valid if the DACC_BCST_CTL.DMAEN bit =1.
5 (R0/W)	GCKEN	Gated Clock Enable. The DACC_BCST_CTL.GCKEN broadcast bit enables gated clock mode for the DAC0 and DAC1 interface clocks (DACC_ACLK and DACC_BCLK pins). When enabled, the related clock toggles only when valid data is driven on the DACC_AD0 pin and/or the DACC_BD0 pin. When disabled, the clocks are free running.
4 (R0/W)	LSBF	LSB-First Mode. The DACC_BCST_CTL.LSBF broadcast bit selects between LSB-first mode or MSB-first mode transfers for the DAC0 and DAC1 interfaces.
3 (R0/W)	DMAW	DMA Data Width. The DACC_BCST_CTL.DMAW broadcast bit selects the DMA data width for the DAC0 and DAC1 interfaces. This bit is only valid if the DACC_BCST_CTL.DMAEN bit =1.
2 (R0/W)	DMAEN	DMA Enable. The DACC_BCST_CTL.DMAEN broadcast bit enables DMA transfers for the DAC0 and DAC1 interfaces.
0 (R0/W)	EN	Enable. The DACC_BCST_CTL.EN broadcast bit enables operations for the DAC0 and DAC1 interfaces.

Base Pointer 0 Register

The DACC_BPTR0 register provides the base pointer (address) used by DMA read operations for the DAC0 interface. The value of base address (pointer) in the DACC_BPTR0 register at the start of the DMA work unit (start of frame) corresponds to one of the following transfers:

- The first transfer after DACC is enabled in DMA mode
- The first transfer after a wrap around occurs in circular buffering mode

The data for the first transfer is read from memory, starting at the address indicated with the value of the base address. Further DMA data is read from memory at addresses incremented by DACC_MOD0 bytes.

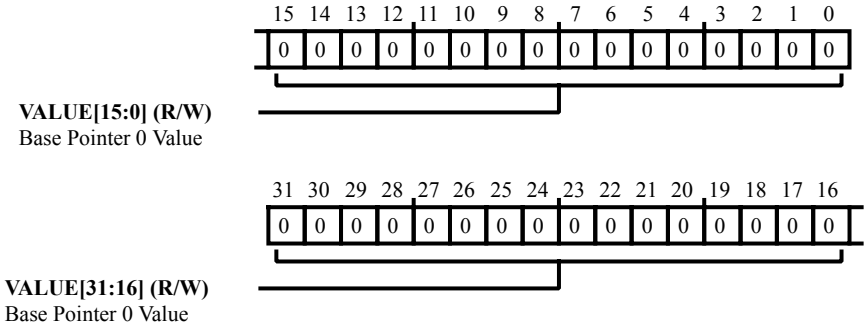

Figure 27-5: DACC_BPTR0 Register Diagram

Table 27-5: DACC_BPTR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Base Pointer 0 Value. The DACC_BPTR0.VALUE bits hold the base pointer (address) for the first DMA transfer.

Base Pointer 1 Register

The DACC_BPTR1 register provides the base pointer (address) used by DMA read operations for the DAC1 interface. The value of base address (pointer) in the DACC_BPTR1 register at the start of the DMA work unit (start of frame) corresponds to one of the following transfers:

- The first transfer after DACC is enabled in DMA mode
- The first transfer after a wrap around occurs in circular buffering mode

The data for the first transfer is read from memory, starting at the address indicated with the value of the base address. Further DMA data is read from memory at addresses incremented by DACC_MOD1 bytes.

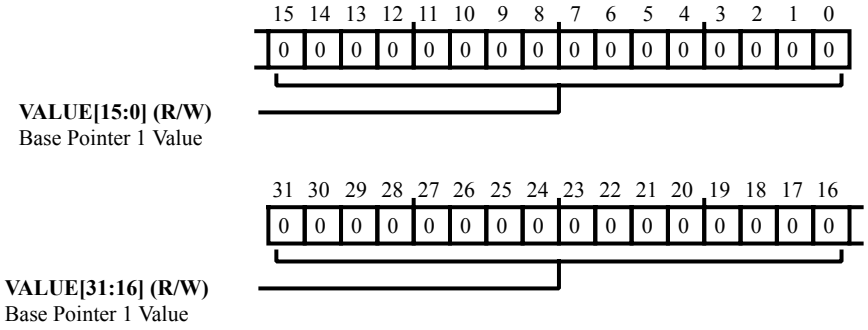

Figure 27-6: DACC_BPTR1 Register Diagram

Table 27-6: DACC_BPTR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Base Pointer 1 Value. The DACC_BPTR1.VALUE bits hold the base pointer (address) for the first DMA transfer.

Count 0 Register

In linear DMA mode, the DACC_CNT0 register holds the transfer count of a DMA work-unit for DAC0. The DMA fetches the indicated number of read data and generates an interrupt after transmitting all data to the DAC. After the data corresponding to the count is transmitted (in linear DMA mode), the DACC does not send further syncs to the DAC.

In circular buffer mode, this register holds the count of DMA after which wrap around to the base pointer address ((DACC_BPTR0)) occurs. The DMA fetches the indicated number of read data and (optionally) generates an interrupt after receiving all data corresponding to one set of DACC_CNT0 counts (one circular buffer) from memory.

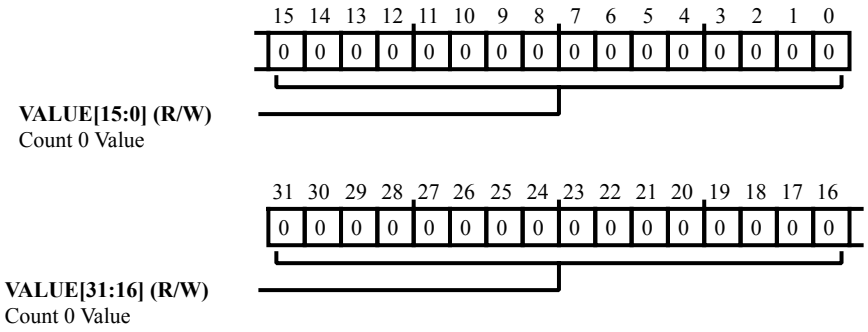

Figure 27-7: DACC_CNT0 Register Diagram

Table 27-7: DACC_CNT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Count 0 Value. The DACC_CNT0.VALUE bits select the transfer count of a DMA work-unit for DAC0.

Count 1 Register

In linear DMA mode, the DACC_CNT1 register holds the transfer count of a DMA work-unit for DAC1. The DMA fetches the indicated number of read data and generates an interrupt after transmitting all data to the DAC. After the data corresponding to the count is transmitted (in linear DMA mode), the DACC does not send further syncs to the DAC.

In circular buffer mode, this register holds the count of DMA after which wrap around to the base pointer address ((DACC_BPTR1)) occurs. The DMA fetches the indicated number of read data and (optionally) generates an interrupt after receiving all data corresponding to one set of DACC_CNT1 counts (one circular buffer) from memory.

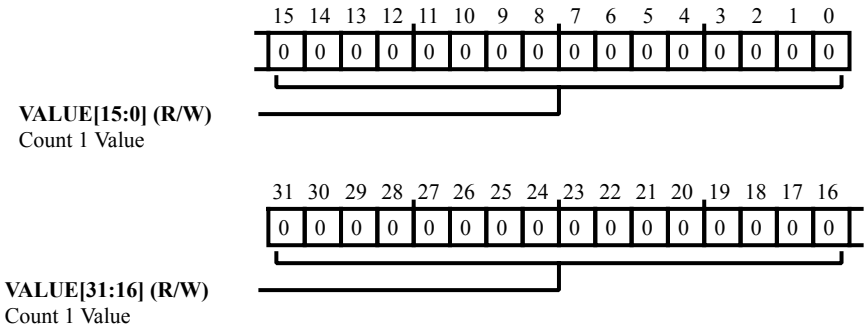

Figure 27-8: DACC_CNT1 Register Diagram

Table 27-8: DACC_CNT1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Count 1 Value. The DACC_CNT1.VALUE bits select the transfer count of a DMA work-unit for DAC1.

Current Count 0 Register

The DACC_CNTCUR0 register holds the current count of the DMA work-unit of the DAC0 interface. This register is loaded from the DACC_CNT0 register when either of the following occur:

- The DACC is enabled in DMA mode
- A wrap around occurs in circular buffering mode

The count decrements with each DMA read from memory.

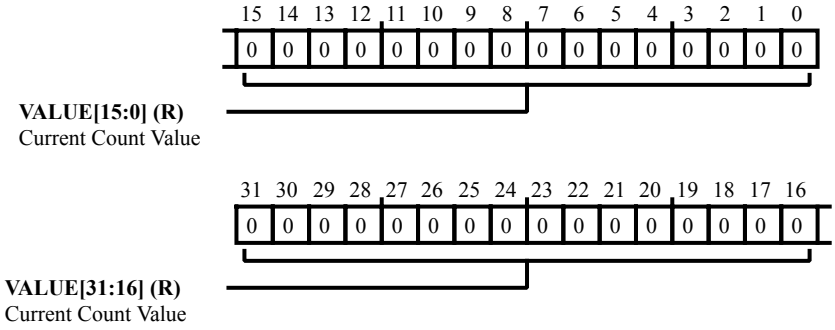

Figure 27-9: DACC_CNTCUR0 Register Diagram

Table 27-9: DACC_CNTCUR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Current Count Value. The DACC_CNTCUR0.VALUE bits hold the current transfer count of a DMA work-unit for DAC0.

Current Count 1 Register

The DACC_CNTCUR1 register holds the current count of the DMA work-unit of the DAC1 interface. This register is loaded from the DACC_CNT1 register when either of the following occur:

- The DACC is enabled in DMA mode
- A wrap around occurs in circular buffering mode

The count decrements with each DMA read from memory.

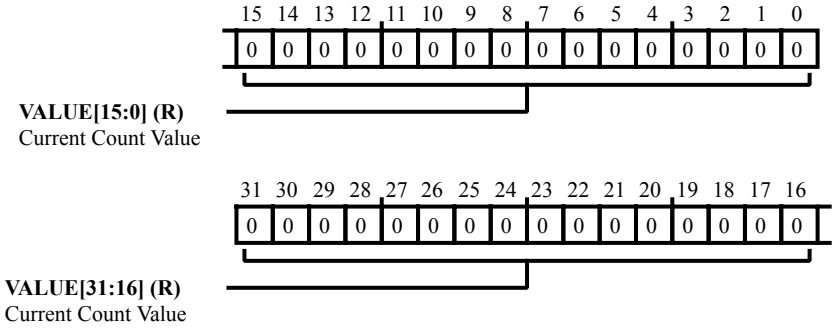

Figure 27-10: DACC_CNTCUR1 Register Diagram

Table 27-10: DACC_CNTCUR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Current Count Value. The DACC_CNTCUR1.VALUE bits hold the current transfer count of a DMA work-unit for DAC1.

Control 0 Register

The DACC_CTL0 register controls the DAC0 interface.

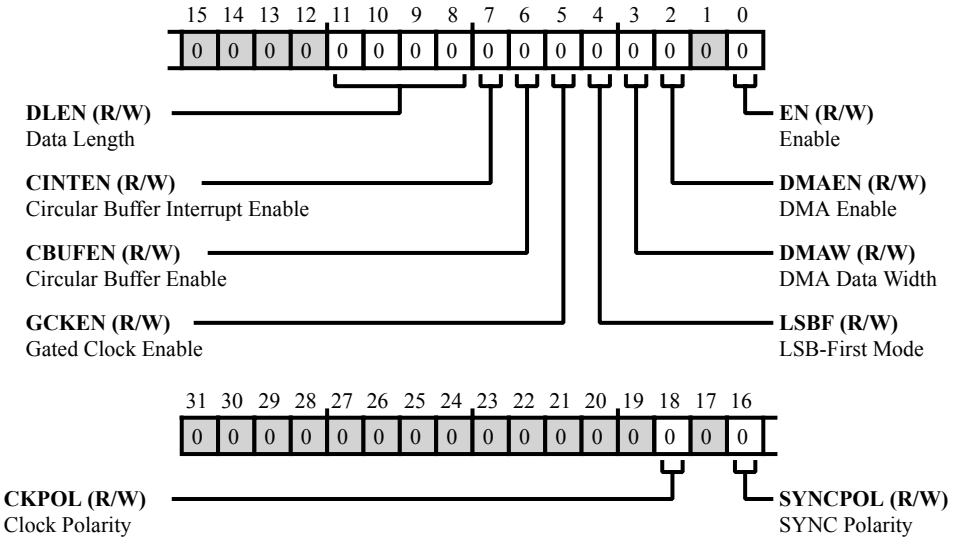

Figure 27-11: DACC_CTL0 Register Diagram

Table 27-11: DACC_CTL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	CKPOL	Clock Polarity. The DACC_CTL0.CKPOL bit selects the polarity of the DAC0 interface clock signal (DACC_ACLK pin) on which to drive the DAC0 output (DACC_AD0 pin).	
		0	Drive on Clock Falling Edge
		1	Drive on Clock Rising Edge
16 (R/W)	SYNCPOL	SYNC Polarity. The DACC_CTL0.SYNCPOL bit selects the polarity for the DAC0 interface sync signal (DACC_AFS pin).	
		0	Active Low
		1	Active High
11:8 (R/W)	DLEN	Data Length. The DACC_CTL0.DLEN bits choose the data length (in bits) for the DAC0 interface. A value =0 for this field implies a data length of 16 bits. For data lengths less that 16 bits, use LSB-aligned data and zero fill unused bits.	

Table 27-11: DACC_CTL0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	CINTEN	Circular Buffer Interrupt Enable. The DACC_CTL0.CINTEN bit enables generation of the DACC_DAC0 interrupt at the end of each circular buffer. This bit is only valid if the DACC_CTL0.DMAEN bit =1 and the DACC_CTL0.CBUFEN bit =1.	
		0	Disable
		1	Enable
6 (R/W)	CBUFEN	Circular Buffer Enable. The DACC_CTL0.CBUFEN bit enables circular buffer DMA mode for the DAC0 interface. This bit is only valid if the DACC_CTL0.DMAEN bit =1.	
		0	Disable
		1	Enable
5 (R/W)	GCKEN	Gated Clock Enable. The DACC_CTL0.GCKEN bit enables gated clock mode for the DAC0 interface clock (DACC_ACLK pin). When enabled, the clock toggles only when valid data is driven on the DACC_AD0 pin. When disabled, the clock is free running.	
		0	Disable Gated Clock Mode
		1	Enable Gated Clock Mode
4 (R/W)	LSBF	LSB-First Mode. The DACC_CTL0.LSBF bit selects between LSB-first mode or MSB-first mode transfers for the DAC0 interface.	
		0	MSB-First Mode
		1	LSB-First Mode
3 (R/W)	DMAW	DMA Data Width. The DACC_CTL0.DMAW bit selects the DMA data width for the DAC0 interface. This bit is only valid if the DACC_CTL0.DMAEN bit =1.	
		0	16-Bit DMA Data
		1	32-Bit DMA Data
2 (R/W)	DMAEN	DMA Enable. The DACC_CTL0.DMAEN bit enables DMA transfers for the DAC0 interface.	
		0	Disable
		1	Enable

Table 27-11: DACC_CTL0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Enable. The `DACC_CTL0.EN` bit enables operations for the DAC0 interface.	
		0	Disable
		1	Enable

Control 1 Register

The DACC_CTL1 register controls the DAC1 interface.

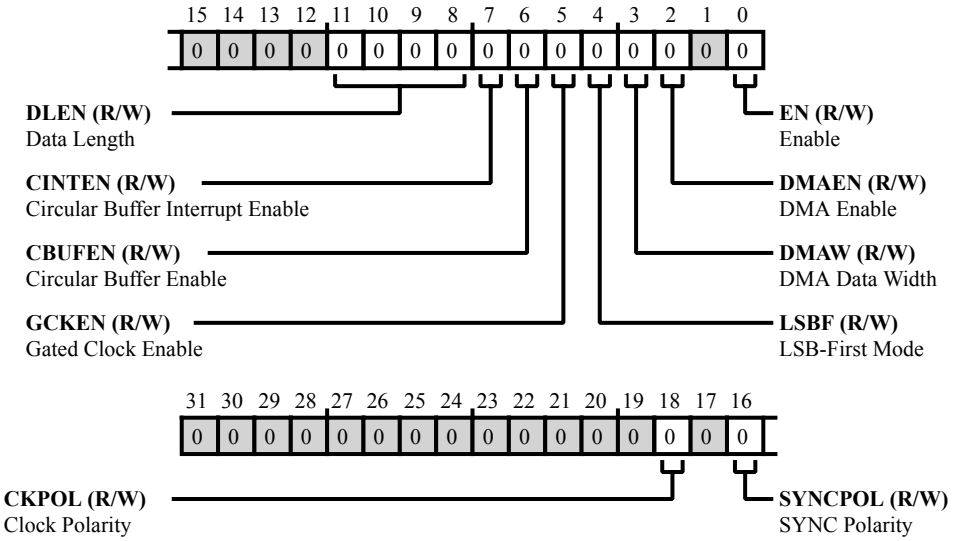

Figure 27-12: DACC_CTL1 Register Diagram

Table 27-12: DACC_CTL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	CKPOL	Clock Polarity. The DACC_CTL1.CKPOL bit selects the polarity of the DAC1 interface clock signal (DACC_BCLK pin) on which to drive the DAC1 output (DACC_BD0 pin).	
		0	Drive on Clock Falling Edge
		1	Drive on Clock Rising Edge
16 (R/W)	SYNCPOL	SYNC Polarity. The DACC_CTL1.SYNCPOL bit selects the polarity for the DAC1 interface sync signal (DACC_BFS pin).	
		0	Active Low
		1	Active High
11:8 (R/W)	DLEN	Data Length. The DACC_CTL1.DLEN bits choose the data length (in bits) for the DAC1 interface. A value =0 for this field implies a data length of 16 bits. For data lengths less that 16 bits, use LSB-aligned data and zero fill unused bits.	

Table 27-12: DACC_CTL1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	CINTEN	Circular Buffer Interrupt Enable. The DACC_CTL1.CINTEN bit enables generation of the DACC_DAC1 interrupt at the end of each circular buffer. This bit is only valid if the DACC_CTL1.DMAEN bit =1 and the DACC_CTL1.CBUFEN bit =1.	
		0	Disable
		1	Enable
6 (R/W)	CBUFEN	Circular Buffer Enable. The DACC_CTL1.CBUFEN bit enables circular buffer DMA mode for the DAC1 interface. This bit is only valid if the DACC_CTL1.DMAEN bit =1.	
		0	Disable
		1	Enable
5 (R/W)	GCKEN	Gated Clock Enable. The DACC_CTL1.GCKEN bit enables gated clock mode for the DAC1 interface clock (DACC_BCLK pin). When enabled, the clock toggles only when valid data is driven on the DACC_BD0 pin. When disabled, the clock is free running.	
		0	Disable Gated Clock Mode
		1	Enable Gated Clock Mode
4 (R/W)	LSBF	LSB-First Mode. The DACC_CTL1.LSBF bit selects between LSB-first mode or MSB-first mode transfers for the DAC1 interface.	
		0	MSB-First Mode
		1	LSB-First Mode
3 (R/W)	DMAW	DMA Data Width. The DACC_CTL1.DMAW bit selects the DMA data width for the DAC1 interface. This bit is only valid if the DACC_CTL1.DMAEN bit =1.	
		0	16-Bit DMA Data
		1	32-Bit DMA Data
2 (R/W)	DMAEN	DMA Enable. The DACC_CTL1.DMAEN bit enables DMA transfers for the DAC1 interface.	
		0	Disable
		1	Enable

Table 27-12: DACC_CTL1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Enable. The `DACC_CTL1.EN` bit enables operations for the DAC1 interface.	
		0	Disable
		1	Enable

Data FIFO 0 Register

The DACC_DAT0 register provides a memory mapped register location for the DAC0 data FIFO. This location is used in non-DMA mode, permitting the core writes to transmit data to the DAC. Only the lower 16 bits are considered as DAC data; the upper 16 bits are ignored.

The core should only write this register when no DAC DMA is in progress (DACC_ISTAT.DINT0 =1). If the core attempts a write into this register during an active DAC DMA (DACC_ISTAT.DINT0 =0), the DACC ignores the write transaction. Reads of DACC_DAT0 return the value of the top entry of the DAC0 FIFO.

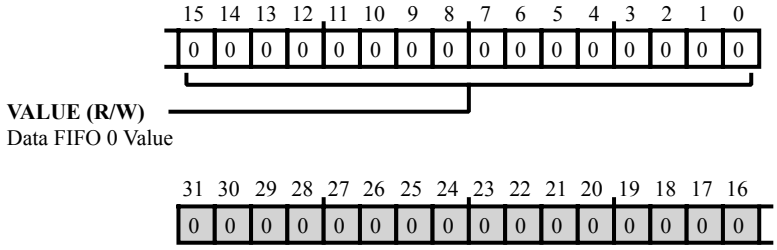

Figure 27-13: DACC_DAT0 Register Diagram

Table 27-13: DACC_DAT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Data FIFO 0 Value. The DACC_DAT0.VALUE bits provides a memory mapped location for the DAC0 data FIFO. Core writes to these bits (when no DMA is active) are transmitted to the DAC. Reads of these bits return the value of the top entry of the DAC0 FIFO.

Data FIFO 1 Register

The DACC_DAT1 register provides a memory mapped register location for the DAC1 data FIFO. This location is used in non-DMA mode, permitting the core writes to transmit data to the DAC. Only the lower 16 bits are considered as DAC data; the upper 16 bits are ignored.

The core should only write this register when no DAC DMA is in progress (DACC_ISTAT.DINT1 =1). If the core attempts a write into this register during an active DAC DMA (DACC_ISTAT.DINT1 =0), the DACC ignores the write transaction. Reads of DACC_DAT1 return the value of the top entry of the DAC1 FIFO.

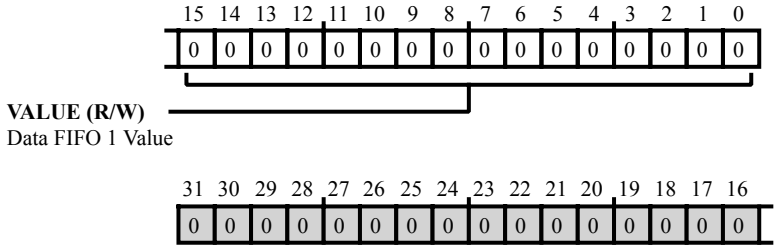

Figure 27-14: DACC_DAT1 Register Diagram

Table 27-14: DACC_DAT1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Data FIFO 1 Value. The DACC_DAT1.VALUE bits provides a memory mapped location for the DAC1 data FIFO. Core writes to these bits (when no DMA is active) are transmitted to the DAC. Reads of these bits return the value of the top entry of the DAC1 FIFO.

Error Mask Register

The DACC_ERRMSK register masks (disables) or unmasks (enables) reporting of DAC related errors.

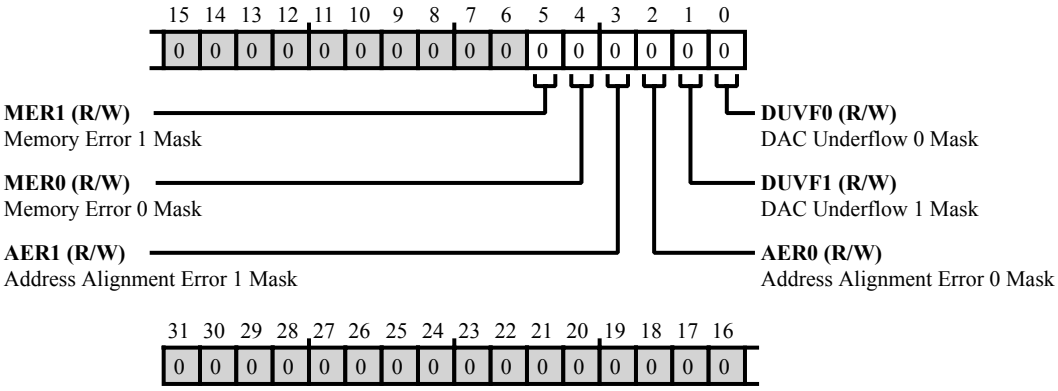

Figure 27-15: DACC_ERRMSK Register Diagram

Table 27-15: DACC_ERRMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	MER1	Memory Error 1 Mask. The DACC_ERRMSK.MER1 bit masks (disables) generating an error interrupt on a DAC1 memory error.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)
4 (R/W)	MER0	Memory Error 0 Mask. The DACC_ERRMSK.MER0 bit masks (disables) generating an error interrupt on a DAC0 memory error.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)
3 (R/W)	AER1	Address Alignment Error 1 Mask. The DACC_ERRMSK.AER1 bit masks (disables) generating an error interrupt on a DAC1 address alignment error.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)

Table 27-15: DACC_ERRMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	AER0	Address Alignment Error 0 Mask. The DACC_ERRMSK.AER0 bit masks (disables) generating an error interrupt on a DAC0 address alignment error.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)
1 (R/W)	DUVF1	DAC Underflow 1 Mask. The DACC_ERRMSK.DUVF1 bit masks (disables) generating an error interrupt on a DAC1 underflow error.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)
0 (R/W)	DUVF0	DAC Underflow 0 Mask. The DACC_ERRMSK.DUVF0 bit masks (disables) generating an error interrupt on a DAC0 underflow error.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)

Error Mask Clear Register

The DACC_ERRMSK_CLR register can be used to selectively clear bits in the DACC_ERRMSK register without affecting other bits in the register. Writing a 1 to any bit position in DACC_ERRMSK_CLR clears the corresponding bit in DACC_ERRMSK. Reading the DACC_ERRMSK_CLR register returns the data present in the DACC_ERRMSK register.

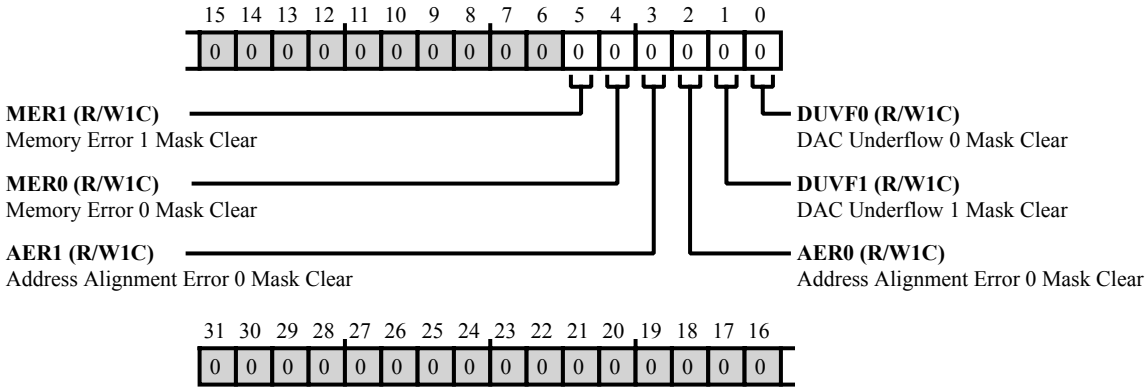

Figure 27-16: DACC_ERRMSK_CLR Register Diagram

Table 27-16: DACC_ERRMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W1C)	MER1	Memory Error 1 Mask Clear. Write 1 to DACC_ERRMSK_CLR.MER1 to clear the corresponding bit in DACC_ERRMSK.
4 (R/W1C)	MER0	Memory Error 0 Mask Clear. Write 1 to DACC_ERRMSK_CLR.MER0 to clear the corresponding bit in DACC_ERRMSK.
3 (R/W1C)	AER1	Address Alignment Error 0 Mask Clear. Write 1 to DACC_ERRMSK_CLR.AER1 to clear the corresponding bit in DACC_ERRMSK.
2 (R/W1C)	AER0	Address Alignment Error 0 Mask Clear. Write 1 to DACC_ERRMSK_CLR.AER0 to clear the corresponding bit in DACC_ERRMSK.
1 (R/W1C)	DUVF1	DAC Underflow 1 Mask Clear. Write 1 to DACC_ERRMSK_CLR.DUVF1 to clear the corresponding bit in DACC_ERRMSK.
0 (R/W1C)	DUVF0	DAC Underflow 0 Mask Clear. Write 1 to DACC_ERRMSK_CLR.DUVF0 to clear the corresponding bit in DACC_ERRMSK.

Error Mask Set Register

The DACC_ERRMSK_SET register can be used to selectively set bits in the DACC_ERRMSK register without affecting other bits in the register. Writing a 1 to any bit position in DACC_ERRMSK_SET sets the corresponding bit in DACC_ERRMSK. Reading the DACC_ERRMSK_SET register returns the data present in the DACC_ERRMSK register.

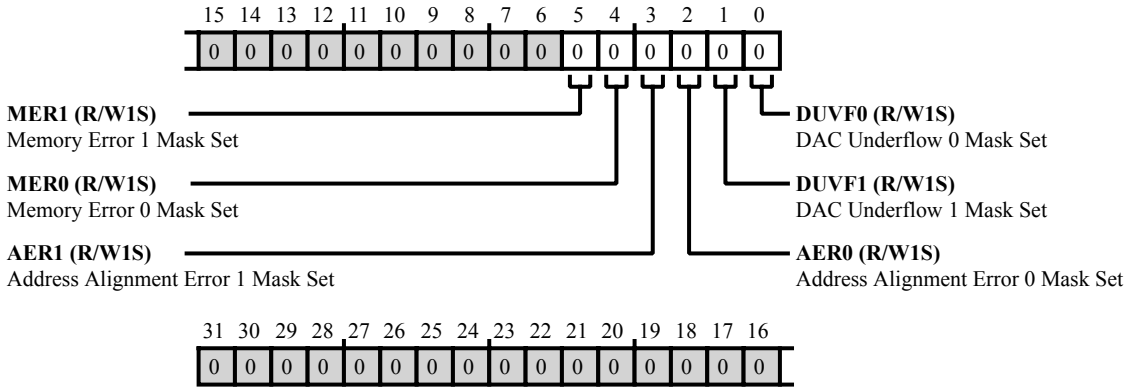

Figure 27-17: DACC_ERRMSK_SET Register Diagram

Table 27-17: DACC_ERRMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W1S)	MER1	Memory Error 1 Mask Set. Write 1 to DACC_ERRMSK_SET.MER1 to set the corresponding bit in DACC_ERRMSK.
4 (R/W1S)	MER0	Memory Error 0 Mask Set. Write 1 to DACC_ERRMSK_SET.MER0 to set the corresponding bit in DACC_ERRMSK.
3 (R/W1S)	AER1	Address Alignment Error 1 Mask Set. Write 1 to DACC_ERRMSK_SET.AER1 to set the corresponding bit in DACC_ERRMSK.
2 (R/W1S)	AER0	Address Alignment Error 0 Mask Set. Write 1 to DACC_ERRMSK_SET.AER0 to set the corresponding bit in DACC_ERRMSK.
1 (R/W1S)	DUVF1	DAC Underflow 1 Mask Set. Write 1 to DACC_ERRMSK_SET.DUVF1 to set the corresponding bit in DACC_ERRMSK.
0 (R/W1S)	DUVF0	DAC Underflow 0 Mask Set. Write 1 to DACC_ERRMSK_SET.DUVF0 to set the corresponding bit in DACC_ERRMSK.

Error Status Register

The DACC_ERRSTAT register indicates error status for DACC operations. When any bit in this register is set, the DACC generates the DACC_ERR interrupt.

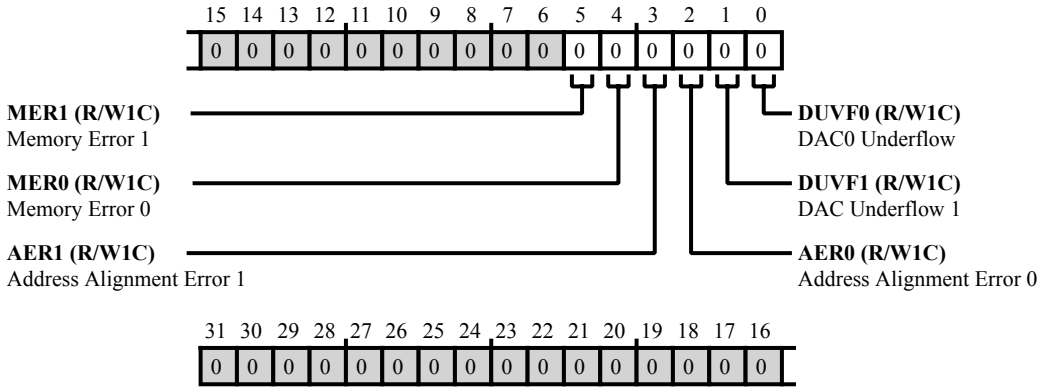

Figure 27-18: DACC_ERRSTAT Register Diagram

Table 27-18: DACC_ERRSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1C)	MER1	Memory Error 1. The DACC_ERRSTAT.MER1 bit indicates whether a memory error has occurred (erroneous read response received) during a DMA transfer for the DAC1 interface.	
		0	No Status
		1	Memory Error Occurred
4 (R/W1C)	MER0	Memory Error 0. The DACC_ERRSTAT.MER0 bit indicates whether a memory error has occurred (erroneous read response received) during a DMA transfer for the DAC0 interface.	
		0	No Status
		1	Memory Error Occurred
3 (R/W1C)	AER1	Address Alignment Error 1. The DACC_ERRSTAT.AER1 bit indicates whether an address alignment error has occurred during a DMA transfer for the DAC1 interface. Recommended practice is to clear this bit (W1C) when enabling DMA with the DACC_CTL1.DMAEN bit.	
		0	No Status
		1	Alignment Error Occurred

Table 27-18: DACC_ERRSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	AER0	Address Alignment Error 0. The DACC_ERRSTAT.AER0 bit indicates whether an address alignment error has occurred during a DMA transfer for the DAC0 interface. Recommended practice is to clear this bit (W1C) when enabling DMA with the DACC_CTL0.DMAEN bit.	
		0	No Status
		1	Alignment Error Occurred
1 (R/W1C)	DUVF1	DAC Underflow 1. The DACC_ERRSTAT.DUVF1 bit indicates whether a data underflow has occurred in the FIFO for the DAC1 interface (for example, no data was present in the FIFO when the DACC_BFS pin is about to be asserted).	
		0	No Status
		1	Underflow Occurred
0 (R/W1C)	DUVF0	DAC0 Underflow. The DACC_ERRSTAT.DUVF0 bit indicates whether a data underflow has occurred in the FIFO for the DAC0 interface (for example, no data was present in the FIFO when the DACC_AFS pin is about to be asserted).	
		0	No Status
		1	Underflow Occurred

Interrupt Mask Register

The DACC_IMSK register masks (disables) generation of DACC_DAC0 or DACC_DAC1 interrupts based on status in the DACC_ISTAT register.

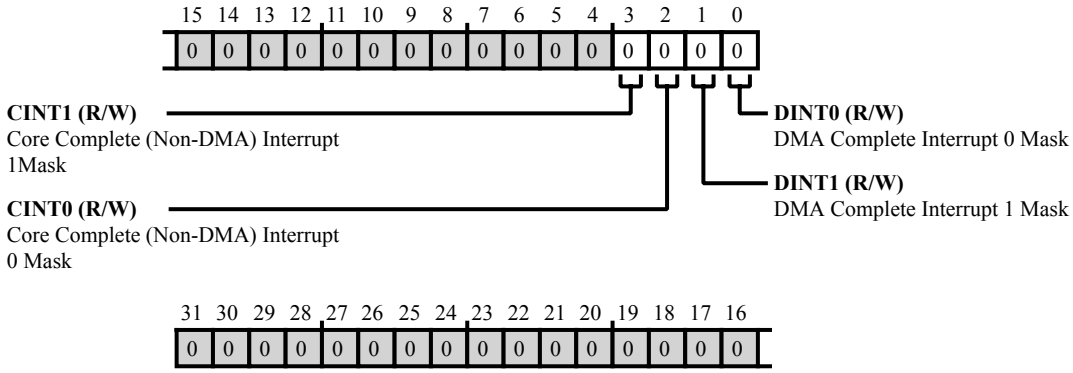

Figure 27-19: DACC_IMSK Register Diagram

Table 27-19: DACC_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	CINT1	Core Complete (Non-DMA) Interrupt 1Mask. The DACC_IMSK.CINT1 bits mask (disable) the DACC_ISTAT.CINT1 DMA complete interrupts for data transfer related to DAC1.	
		0	Unmask (Enable) DAC Interrupt
		1	Mask (Disable) DAC Interrupt
2 (R/W)	CINT0	Core Complete (Non-DMA) Interrupt 0 Mask. The DACC_IMSK.CINT0 bits mask (disable) the DACC_ISTAT.CINT0 DMA complete interrupts for data transfer related to DAC0.	
		0	Unmask (Enable) DAC Interrupt
		1	Mask (Disable) DAC Interrupt
1 (R/W)	DINT1	DMA Complete Interrupt 1 Mask. The DACC_IMSK.DINT1 bits mask (disable) the DACC_ISTAT.DINT1 DMA complete interrupts for data transfer related to DAC1.	
		0	Unmask (Enable) DAC Interrupt
		1	Mask (Disable) DAC Interrupt

Table 27-19: DACC_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	DINT0	DMA Complete Interrupt 0 Mask. The DACC_IMSK.DINT0 bits mask (disable) the DACC_ISTAT.DINT0 DMA complete interrupts for data transfer related to DAC0.	
		0	Unmask (Enable) DAC Interrupt
		1	Mask (Disable) DAC Interrupt

Interrupt Mask Clear Register

The DACC_IMSK_CLR register can be used to selectively clear bits in the DACC_IMSK register without affecting other bits in the register. Writing a 1 to any bit position in DACC_IMSK_CLR clears the corresponding bit in DACC_IMSK. Reading the DACC_IMSK_CLR register returns the data present in the DACC_IMSK register.

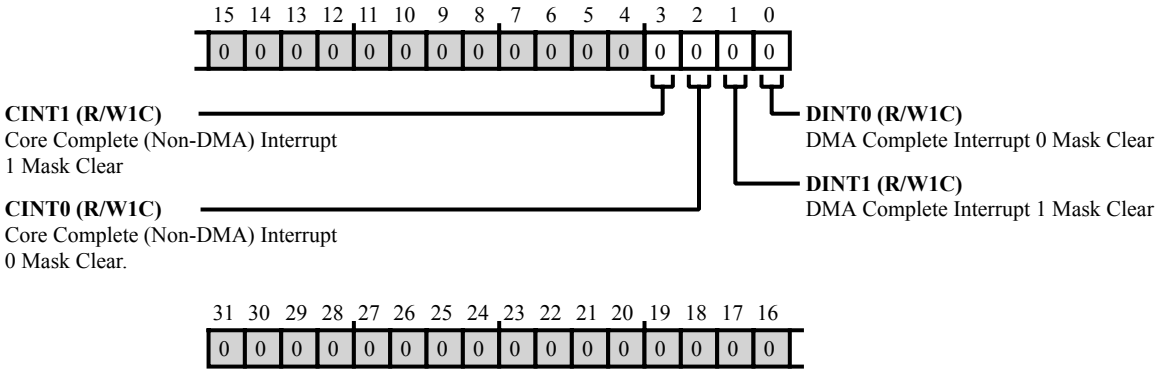

Figure 27-20: DACC_IMSK_CLR Register Diagram

Table 27-20: DACC_IMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W1C)	CINT1	Core Complete (Non-DMA) Interrupt 1 Mask Clear. The DACC_IMSK_CLR.CINT1 bits permit clearing the DACC_IMSK.CINT1 bit without affecting other bits in the register. Write 1 to the DACC_IMSK_CLR.CINT1 bit to clear the DACC_IMSK.CINT1 bit.
2 (R/W1C)	CINT0	Core Complete (Non-DMA) Interrupt 0 Mask Clear.. The DACC_IMSK_CLR.CINT0 bits permit clearing the DACC_IMSK.CINT0 bit without affecting other bits in the register. Write 1 to the DACC_IMSK_CLR.CINT0 bit to clear the DACC_IMSK.CINT0 bit.
1 (R/W1C)	DINT1	DMA Complete Interrupt 1 Mask Clear. The DACC_IMSK_CLR.DINT1 bits permit clearing the DACC_IMSK.DINT1 bit without affecting other bits in the register. Write 1 to the DACC_IMSK_CLR.DINT1 bit to clear the DACC_IMSK.DINT1 bit.
0 (R/W1C)	DINT0	DMA Complete Interrupt 0 Mask Clear. The DACC_IMSK_CLR.DINT0 bits permit clearing the DACC_IMSK.DINT0 bit without affecting other bits in the register. Write 1 to the DACC_IMSK_CLR.DINT0 bit to clear the DACC_IMSK.DINT0 bit.

Interrupt Mask Set Register

The DACC_IMSK_SET register can be used to selectively set bits in the DACC_IMSK register without affecting other bits in the register. Writing a 1 to any bit position in DACC_IMSK_SET sets the corresponding bit in DACC_IMSK. Reading the DACC_IMSK_SET register returns the data present in the DACC_IMSK register.

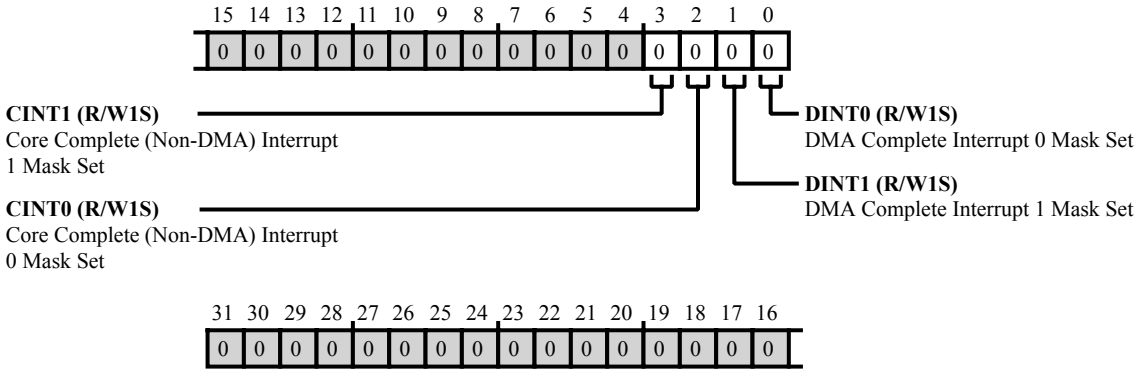

Figure 27-21: DACC_IMSK_SET Register Diagram

Table 27-21: DACC_IMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W1S)	CINT1	Core Complete (Non-DMA) Interrupt 1 Mask Set. The DACC_IMSK_SET.CINT1 bits permit setting the DACC_IMSK.CINT1 bit without affecting other bits in the register. Write 1 to the DACC_IMSK_SET.CINT1 bit to set the DACC_IMSK.CINT1 bit.
2 (R/W1S)	CINT0	Core Complete (Non-DMA) Interrupt 0 Mask Set. The DACC_IMSK_SET.CINT0 bits permit setting the DACC_IMSK.CINT0 bit without affecting other bits in the register. Write 1 to the DACC_IMSK_SET.CINT0 bit to set the DACC_IMSK.CINT0 bit.
1 (R/W1S)	DINT1	DMA Complete Interrupt 1 Mask Set. The DACC_IMSK_SET.DINT1 bits permit setting the DACC_IMSK.DINT1 bit without affecting other bits in the register. Write 1 to the DACC_IMSK_SET.DINT1 bit to set the DACC_IMSK.DINT1 bit.
0 (R/W1S)	DINT0	DMA Complete Interrupt 0 Mask Set. The DACC_IMSK_SET.DINT0 bits permit setting the DACC_IMSK.DINT0 bit without affecting other bits in the register. Write 1 to the DACC_IMSK_SET.DINT0 bit to set the DACC_IMSK.DINT0 bit.

Interrupt Status Register

The DACC_ISTAT register indicates DAC0 and DAC1 interrupt status. The DACC generates interrupts corresponding to DAC 0 on the DACC_DAC0 output and generates the interrupts corresponding to DAC1 on the DACC_DAC1 output.

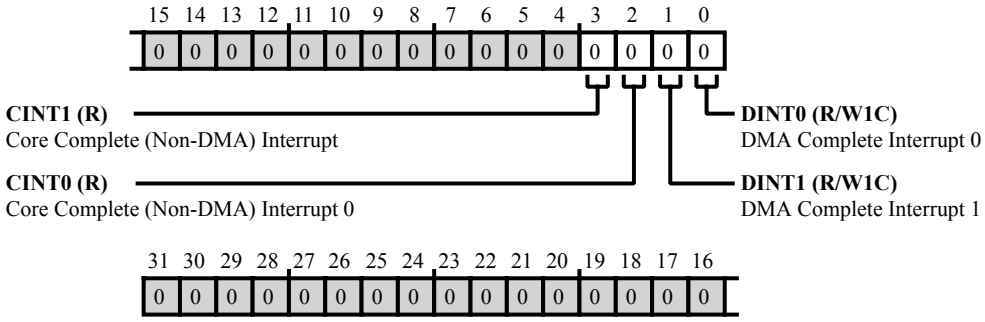

Figure 27-22: DACC_ISTAT Register Diagram

Table 27-22: DACC_ISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/NW)	CINT1	Core Complete (Non-DMA) Interrupt. The DACC_ISTAT.CINT1 bit indicates completion of a core write to the DAC1 interface, implying the DAC data FIFO again has space to accommodate new data writes by core. When cleared, this bit indicates the DAC FIFO is full.	
		0	No Status
		1	Core Complete (Non-DMA)
2 (R/NW)	CINT0	Core Complete (Non-DMA) Interrupt 0. The DACC_ISTAT.CINT0 bit indicates completion of a core write to the DAC0 interface, implying the DAC data FIFO again has space to accommodate new data writes by core. When cleared, this bit indicates the DAC FIFO is full.	
		0	No Status
		1	Core Complete (Non-DMA)
1 (R/W1C)	DINT1	DMA Complete Interrupt 1. The DACC_ISTAT.DINT1 bit indicates completion of the DMA work unit programmed for the DAC1 interface. When set in linear DMA mode, the DACC has transmitted all data to the DAC. When set in circular buffer mode, all read data in a buffer has returned from memory.	
		0	No Status
		1	DMA Complete

Table 27-22: DACC_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W1C)	DINT0	DMA Complete Interrupt 0. The DACC_ISTAT.DINT0 bit indicates completion of the DMA work unit programmed for the DAC0 interface. When set in linear DMA mode, the DACC has transmitted all data to the DAC. When set in circular buffer mode, all read data in a buffer has returned from memory.	
		0	No Status
		1	DMA Complete

Modify 0 Register

The DACC_MOD0 register contains the address increment applied between each DMA read from memory, starting at the base-address (DACC_BPTR0). The value is a signed, two's complement byte address increment.

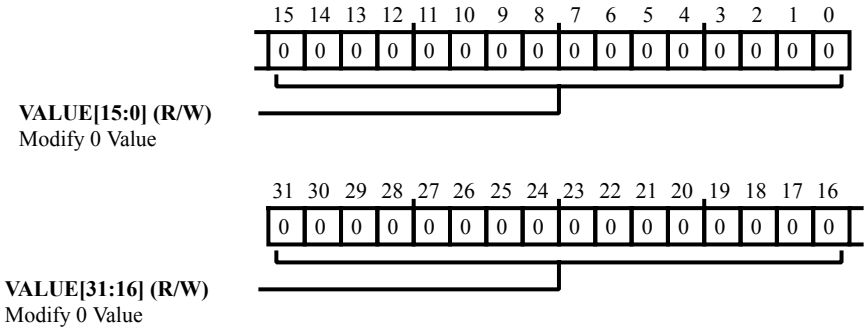

Figure 27-23: DACC_MOD0 Register Diagram

Table 27-23: DACC_MOD0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Modify 0 Value. The DACC_MOD0.VALUE bits hold memory offset increment applied between each DMA read from memory, starting at the base address.

Modify 1 Register

The DACC_MOD1 register contains the address increment applied between each DMA read from memory, starting at the base-address (DACC_BPTR1). The value is a signed, two's complement byte address increment.

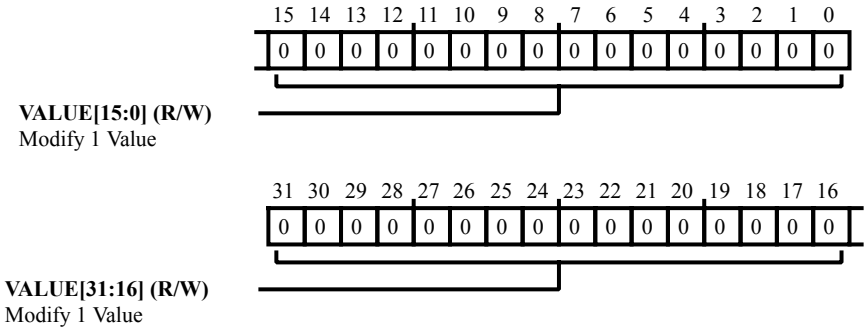

Figure 27-24: DACC_MOD1 Register Diagram

Table 27-24: DACC_MOD1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Modify 1 Value. The DACC_MOD1.VALUE bits hold memory offset increment applied between each DMA read from memory, starting at the base address.

Status Register

The DACC_STAT register indicates status for DACC operations.

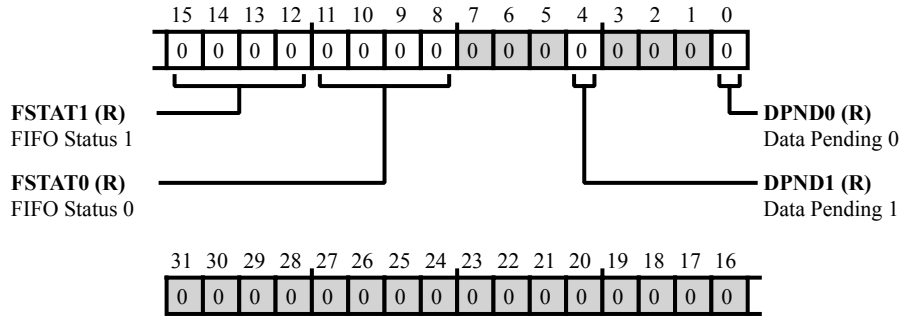

Figure 27-25: DACC_STAT Register Diagram

Table 27-25: DACC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:12 (R/NW)	FSTAT1	FIFO Status 1. The DACC_STAT.FSTAT1 bits indicates the number of 16-bit data in DAC1 FIFO remaining to be transmitted to DAC. The status increments when DMA or core fills the DAC FIFO. The status decrements when a data transmission starts on the DAC interface. These bits are cleared when the DACC_CTL1.DMAEN bit has a 0-1 transition.	
11:8 (R/NW)	FSTAT0	FIFO Status 0. The DACC_STAT.FSTAT0 bits indicates the number of 16-bit data in DAC0 FIFO remaining to be transmitted to DAC. The status increments when DMA or core fills the DAC FIFO. The status decrements when a data transmission starts on the DAC interface. These bits are cleared when the DACC_CTL0.DMAEN bit has a 0-1 transition.	
4 (R/NW)	DPND1	Data Pending 1. The DACC_STAT.DPND1 bit indicates whether the DAC1 interface has made a DMA read access request and is pending (waiting) to receive the data. If DMA for the DAC1 interface is disabled (DACC_CTL1.DMAEN =0), wait until DACC_STAT.DPND1 =0 before enabling DMA for the DAC. This bit is cleared when DACC_CTL1.DMAEN has a 0-1 transition.	
		0	No Status
		1	Pending Read Data

Table 27-25: DACC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/NW)	DPND0	Data Pending 0. The DACC_STAT.DPND0 bit indicates whether the DAC0 interface has made a DMA read access request and is pending (waiting) to receive the data. If DMA for the DAC0 interface is disabled (DACC_CTL0.DMAEN =0), wait until DACC_STAT.DPND0 =0 before enabling DMA for the DAC. This bit is cleared when DACC_CTL0.DMAEN has a 0-1 transition.	
		0	No Status
		1	Pending Read Data

Timing Control 0 Register

The DACC_TC0 register controls timing related to the DAC0 interface clock and sync signals.

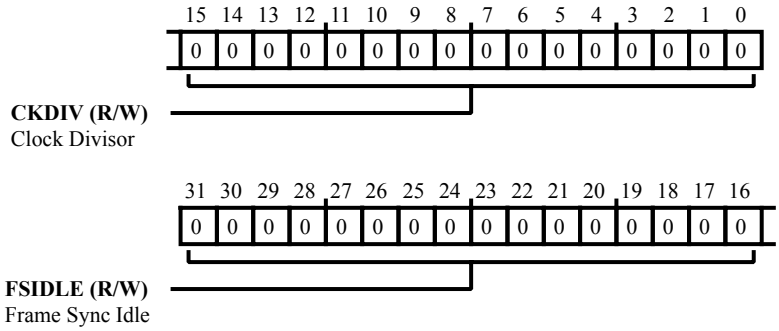

Figure 27-26: DACC_TC0 Register Diagram

Table 27-26: DACC_TC0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	FSIDLE	Frame Sync Idle. The DACC_TC0.FSIDLE value selects minimum idle time (in DACC_ACLK cycles) required between on DACC_AFS asserted edge and the next DACC_AFS asserted edge. A value of 0 implies 1 cycles idle. For DACC_CTL0.DLEN not =0, the DACC_AFS asserted period can be calculated from the DACC_TC0.FSIDLE and DACC_CTL0.DLEN values as: DACC_AFS period = (DACC_TC0.FSIDLE+1) + DACC_CTL0.DLEN For DACC_CTL0.DLEN =0, the DACC_AFS asserted period can be calculated as: DACC_AFS period = (DACC_TC0.FSIDLE+1) + 16
15:0 (R/W)	CKDIV	Clock Divisor. The DACC_TC0.CKDIV bits select the clock divisor ratio (SYSCLK:ACK) for clocks to be sent to ADC calculated as: ACK frequency = (SYSCLK frequency) / (CKDIV+1) Yielding, DACC_TC0.CKDIV =0 represents a ratio of 1:1, =1 represents a ratio of 1:2, DACC_TC0.CKDIV =2 represents a ratio of 1:3, and so on.

Timing Control 1 Register

The DACC_TC1 register controls timing related to the DAC1 interface clock and sync signals.

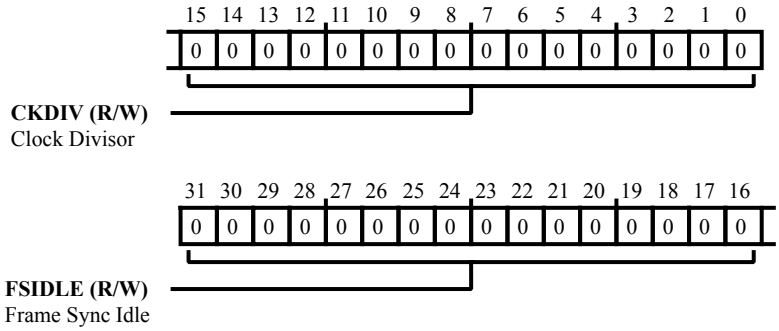

Figure 27-27: DACC_TC1 Register Diagram

Table 27-27: DACC_TC1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	FSIDLE	Frame Sync Idle. The DACC_TC1.FSIDLE value selects minimum idle time (in DACC_BCLK cycles) required between on DACC_BFS asserted edge and the next DACC_BFS asserted edge. A value of 0 implies 1 cycles idle. For DACC_CTL1.DLEN not =0, the DACC_BFS asserted period can be calculated from the DACC_TC1.FSIDLE and DACC_CTL1.DLEN values as: DACC_BFS period = (DACC_TC1.FSIDLE+1) + DACC_CTL1.DLEN For DACC_CTL1.DLEN =0, the DACC_BFS asserted period can be calculated as: DACC_BFS period = (DACC_TC1.FSIDLE+1) + 16
15:0 (R/W)	CKDIV	Clock Divisor. The DACC_TC1.CKDIV bits select the clock divisor ratio (SYSCLK:ACK) for clocks to be sent to ADC calculated as: ACK frequency = (SYSCLK frequency) / (CKDIV+1) Yielding, DACC_TC1.CKDIV =0 represents a ratio of 1:1, =1 represents a ratio of 1:2, DACC_TC1.CKDIV =2 represents a ratio of 1:3, and so on.

28 Harmonic Analysis Engine (HAE)

The harmonic analysis engine (HAE) analyzes harmonic frequencies present on the voltage and current input samples. The HAE receives input samples from two source channels whose frequencies are 45–65 Hz. The HAE then processes the input samples and produces output results. The output results consist of power quality measurements of the fundamental and up to 12 more harmonics.

HAE Features

The HAE features include:

- Processing of two 24-bit signed input channels, consisting of one voltage and one current. The input full scale is limited to ~ +/-6,000,000
- Processing of fundamental frequencies between 45-65 Hz
- Processing of input samples at a nominal 8 kHz rate
- Processing of the fundamental plus 12 harmonic frequencies
- Active, reactive, apparent, I_{RMS}, V_{RMS}, and power factor on the fundamental frequency
- Active, reactive, apparent, I_{RMS}, V_{RMS}, power factor, I_{HD+n}, and V_{HD+n} on the 12 harmonic frequencies
- The accuracy of the measurements, relative to a full scale of +/-6,000,000:
    - Fundamental active or reactive powers 0.1% down to 1/1000 of full scale
    - Fundamental apparent power 0.2% down to 1/1000 of full scale
    - Fundamental I_{RMS}/V_{RMS} 0.1% down to 1/1000 of full scale
    - Fundamental power factor 0.3% based on active and apparent accuracy
    - Harmonic active or reactive powers 1% down to 1/1000 of full scale
    - Harmonic apparent power 2% down to 1/1000 of full scale
    - Harmonic I_{RMS}/V_{RMS} 1% down to 1/1000 of full scale
    - Harmonic power factor 3% based on active and apparent accuracy

- Harmonic I_{HD+n} 2% down to 1/1000 FS based on the fundamental and harmonic I_{RMS}
- Harmonic V_{HD+n} 2% down to 1/1000 FS based on the fundamental and harmonic V_{RMS}

- Twelve 6-bit fields for selecting harmonics to analyze, limited by bandwidth of the input signal. The fundamental component always is provided.

HAE Functional Description

The following sections provide a functional description of the HAE.

Harmonic engine

The hardware block of the harmonic engine works with other HAE blocks to co-process full and partial results

Harmonic Analyzer

The harmonic analyzer block works with the harmonic engine to co-process full and partial results.

Data Transfer Module

The data transfer module transfers three channels of data to and from the HAE module.

ADSP-CM40x HAE Register List

The Harmonic Analysis Engine (HAE) analyzes harmonics present on voltage and current input samples. The HAE receives input samples from two source channels, processes the samples, and produces output results. The output results consist of power quality measurements of the fundamental and up to twelve additional harmonic components. A set of registers governs HAE operations. For more information on HAE functionality, see the HAE register descriptions.

Table 28-1: ADSP-CM40x HAE Register List

Name	Description
HAE_CFG0	Configuration 0 Register
HAE_CFG1	Configuration 1 Register
HAE_CFG2	Configuration 2 Register
HAE_CFG3	Configuration 3 Register
HAE_CFG4	Configuration 4 Register
HAE_DIDT_COEF	DIDT Coefficient Register
HAE_DIDT_GAIN	DIDT Gain Register
HAE_H[nn]_INDX	Harmonic n Index Register
HAE_ISAMPLE	I (Current) Sample Register

Table 28-1: ADSP-CM40x HAE Register List (Continued)

Name	Description
HAE_IWAVEFORM	I (Current) Waveform Register
HAE_RESULTS_RAM	Results RAM Register
HAE_RUN	Run Register
HAE_STAT	Status Register
HAE_VLEVEL	Voltage Level Register
HAE_VSAMPLE	V (Voltage) Sample Register
HAE_VWAVEFORM	V (Voltage) Waveform Register

ADSP-CM40x HAE Interrupt List

Table 28-2: ADSP-CM40x HAE Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
37	HAE0_RXDMA_CH0_ERR	HAE0 DMA Channel 14 Error	Level	
38	HAE0_RXDMA_CH1_ERR	HAE0 DMA Channel 15 Error	Level	
39	HAE0_TXDMA_ERR	HAE0 DMA Channel 16 Error	Level	
90	HAE0_STAT	HAE0 Status	Level	
91	HAE0_RXDMA_CH0	HAE0 RX DMA Channel 0 Transfer Complete	Level	14
92	HAE0_RXDMA_CH1	HAE0 RX DMA Channel 1 Transfer Complete	Level	15
93	HAE0_TXDMA	HAE0 TX DMA Channel 0 Transfer Complete	Level	16

HAE Block Diagram

The ***HAE Block Diagram*** shows the functional blocks within the HAE.

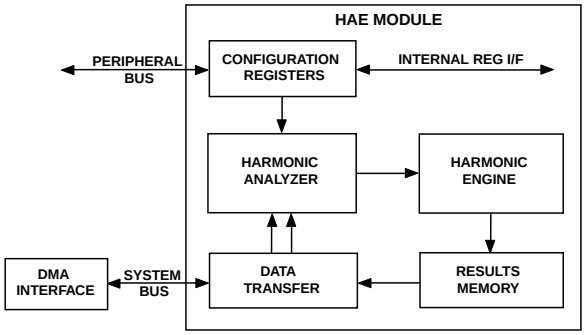

Figure 28-1: HAE Block Diagram

HAE Architectural Concepts

Using the HAE features and event control to their greatest potential requires an understanding of these architectural concepts.

- Harmonic Engine
- Harmonic Analyzer
- Data Transfer Module
- Results Memory

Harmonic Engine

The ***HAE Engine Block Diagram*** presents a synthesized diagram of the harmonic engine, its settings, and its output registers.

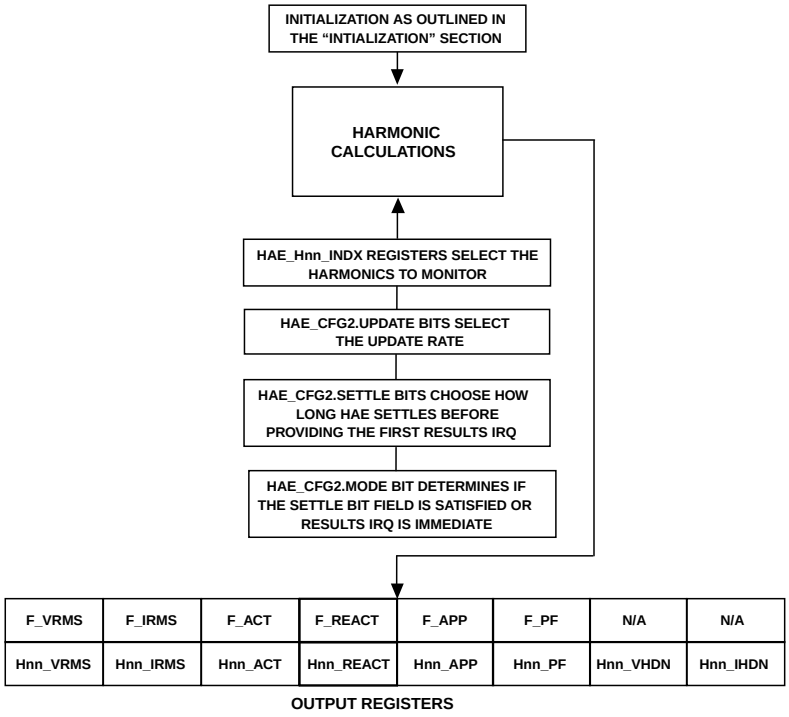

Figure 28-2: HAE Engine Block Diagram

The harmonic engine hardware block works with other HAE blocks to co-process full and partial results (see Harmonic Calculations). At the start of a new sampling period (described in Initialization), the harmonic engine cycles through predefined locations in data RAM, which contain the analyzer processing results. See Harmonic Analyzer for more information.

As the harmonic engine produces results in their final formats (described in HAE Result Ranges and Formats), the results are stored in the results memory (see Results Memory).

The HAE engine computes harmonic information for line frequencies 45–66 Hz. Neutral current can also be analyzed simultaneously with the sum of phase currents. See Theory of Operation for more information.

Harmonic Analyzer

The harmonic analyzer block works with the Harmonic Engine to co-process full and partial results.

To perform harmonic analysis, the HAE contains a pair of voltage and current inputs and a set of 12 indexes to indicate which harmonic components to extract. See Theory of Operation and Harmonic Calculations for more information.

High-Pass Filters (HPFs)

The voltage and current have the option of being high-pass filtered to remove DC offsets. The ***Frequency Response for High-Pass Filters*** figure shows the frequency response of the high-pass filters (the -3 dB point is around 1.1 Hz). The HPFs are enabled by default.

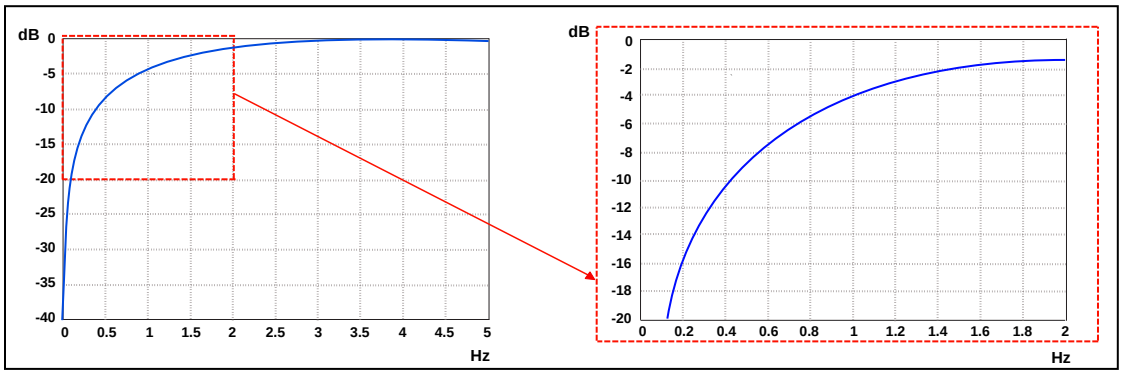

Figure 28-3: Frequency Response for High-Pass Filters

Digital Integrators

For cases when the current is sensed with a di/dt sensor, such as a Rogowski coil, the HAE offers an internal digital integrator for compensation. Ideally, it causes a perfect 90 degrees of phase shift at all the frequencies; at 50 Hz it is close to that value (see the ***Digital Integrator*** figure). The integrator is necessary to restore the signal to its original form before using the signal in HAE calculations. The digital integrator is disabled by default.

With digital integrator enabled for current channel, the amplitude of current will change with the frequency according to a slope of 20 dB per decade. For frequency of 50 Hz, the amplitude of current remains same after passing through the integrator. However, for other frequencies in the range 45-65 Hz, the amplitude of current changes with a slope of 20 dB per decade after passing through integrator.

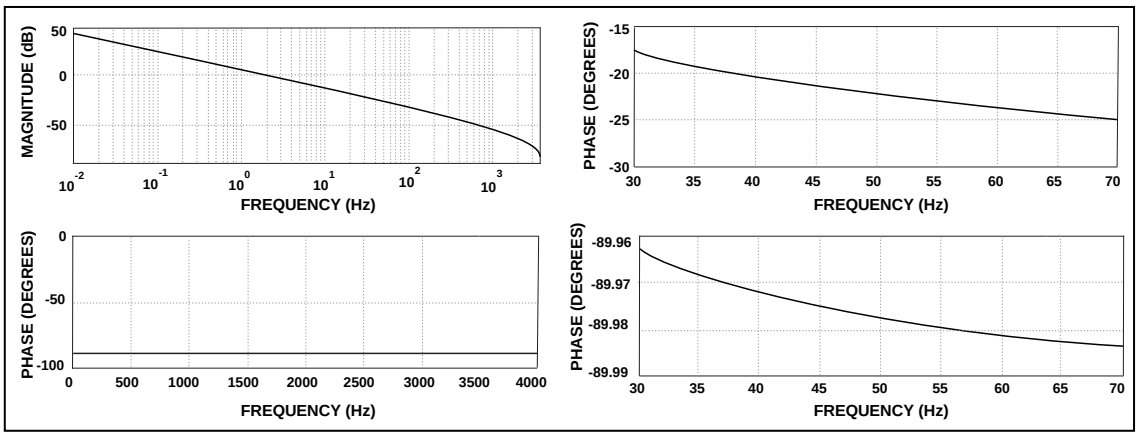

Figure 28-4: Digital Integrator

Phase-Locked Loop and Clock Control (PLL)

The fundamental frequency of the system is extracted from the voltage signal using the phase-locked loop and clock control (PLL) techniques. PLL techniques are optimized for signals with frequencies used in standard power grids around the world (50 Hz or 60 Hz). The techniques consider possible deviations of up to 5 Hz, so the final guaranteed operating range is from 45 Hz to 65 Hz.

The initial detection time of the frequency depends on its value and can take up to several seconds. This activity only happens at the start-up of the HAE block. Once the value of the fundamental frequency is detected, the PLL tracks it continuously.

Settling Times

The block that extracts all of the harmonic RMS and power values is replicated 12 times in parallel for all of the harmonic indexes plus once for the fundamental. Once an index for a particular harmonic changes, there is a settling time of about 700 mSec (~700 / 0.125 = 5600 8K samples). This settling time achieves less than 0.1% error needed for all the internal RMS and powers computations (see the ***Fundamental and Harmonic Values - Settling Times*** plots).

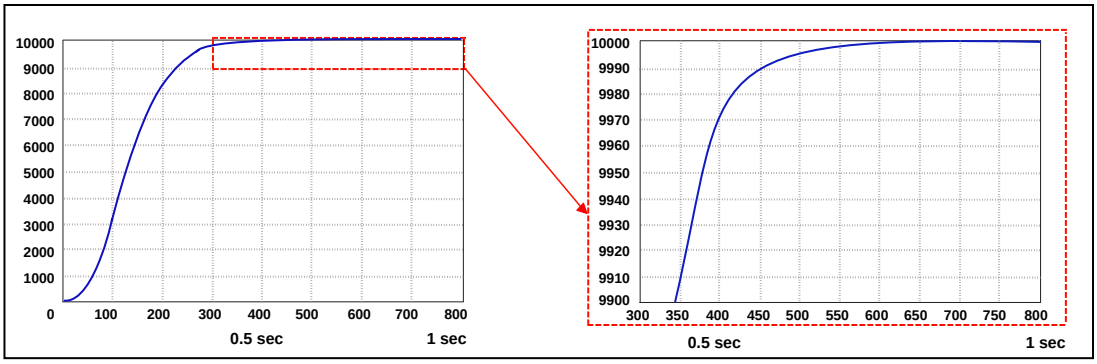

Figure 28-5: Fundamental and Harmonic Values - Settling Times

Data Transfer Module

The data transfer module transfers three channels of data to and from the HAE module.

RX I (Receive Current Sample) Channel

The RX I channel transfers HAE current (I) input samples from system memory through direct memory access (DMA). The RX I request occurs at a nominal 8 kHz rate, depending on the `HAE_CFG1.STARTDIV` bit field. The RX I samples typically come from a SINC filter through a memory buffer. For accurate harmonic results, derive the input samples at the same rate as the HAE sample loop. Therefore, program the SINC and HAE modules to be timing-matched. The SYSCLK / `HAE_CFG1.STARTDIV` determines the HAE sample loop, which must be programmed to be nominally 8 kHz in frequency. There is one `DWORD` transfer each HAE sample loop.

There is also an RX I memory-mapped location (`HAE_ISAMPLE` register) in peripheral space (MMR), which enables the MCU to provide the I channel samples, timed with the `HAE_STAT.RXIRQ` bit, if desired.

RX V (Receive Voltage Sample) Channel

The RX V channel transfers HAE voltage (V) input samples from system memory through DMA. The RX V request occurs at a nominal 8 kHz rate, depending on the `HAE_CFG1.STARTDIV` bit field. The RX V samples typically come from a SINC filter through a memory buffer. For accurate harmonic results, derive the input samples at the same rate as the HAE sample loop. Therefore, program the SINC and HAE modules to be timing-matched. The

SYSCLK / HAE_CFG1.STARTDIV, determines the HAE sample loop which must be programmed to be nominally 8 kHz in frequency. There is one DWORD transfer each HAE sample loop.

There is also an RX V memory-mapped location (HAE_VSAMPLE register) in peripheral space (MMR), which enables the MCU to provide the V channel samples, timed with the HAE_STAT.RXIRQ bit, if desired.

Example:

- The HAE_STAT.RXIRQ bit toggles prior to RX transfers:

  The RX interrupt is used internally in hardware to request I and V samples. The MCU can also use the interrupt when it supplies the input samples, rather than the DMA interface.

- RX transfers indicate that the RX channel is ready:

  The HAE RX channels request one RX sample at an 8 kHz rate, depending on the HAE_CFG1.STARTDIV bit field. The holding register stores the RX data, replacing the previous data. The holding register contents are moved up to an output register and driven to the Harmonic Analyzer module for processing. This operation amounts to a two-deep FIFO, allowing a full sample time of latency on the input sample arrival, nominally 125 uSec (8 kHz).

  If the MCU supplies the input samples, rather than the DMA RX interfaces, there are two memory-mapped locations (HAE_ISAMPLE and HAE_VSAMPLE), where the next samples are written. As previously mentioned, the HAE can use the HAE_STAT.RXIRQ bit to time the sample delivery.

TX (Transmit Results) Channel

The TX channel transfers HAE results from results memory to system memory through DMA. The results for the fundamental and 12 harmonics are stored in 13 8-location fields in the results memory. Therefore, the maximum number of DWORDs to transfer is 13 x 8 = 104. Use the HAE_CFG3.CHANEN bits to select the channels to transfer.

The HAE can request a transfer of results for each 8 kHz sample period, meaning that up to 104 DWORDs can be transferred each 125 uSec. Use the HAE_CFG2.UPDATE bit field to set the request rate to longer intervals. The results also are memory-mapped to peripheral address space, which enables the MCU to read the results, timed with the HAE_STAT.TXIRQ bit, if desired.

Example:

- The HAE_CFG3.CHANEN bit field is set to 0x009 to transfer the fundamental and harmonic contained in the HAE_H3_INDX register (programmed to 5^{th}).

- The HAE_STAT.TXIRQ bit toggles prior to TX transfers:

  The HAE uses the TX interrupt internally in hardware to start the transfer. The MCU can also use the interrupt when the MCU reads the HAE results, rather than the DMA interface transferring it.

- TX transfers indicate that the HAE results data is ready:

  The HAE TX channel can transfer one DWORD every other clock. The internal hardware parses through the HAE_CFG3.CHANEN bit field, eventually traversing all 13 bits. If consecutive channels are enabled, two more

IDLE clocks are inserted between the adjacent channels for a total of three IDLE clocks. When channels are skipped, one more IDLE clock is inserted for each unselected channel. Therefore, there are 3 (inter-channel IDLE) + 2 (skipped channels) = 5 IDLE clocks between the fundamental and HAE_H3_INDX transfers.

Results Memory

The results memory is organized by the fundamental and 12 harmonic indexes. Each of the 13 potentially analyzed frequencies has eight locations dedicated to store various power quality measurements. Each frequency is offset from the next by eight locations.

The ***HAE Results Memory*** figure shows the results memory contents.

	INDEX	0	1	2	3	4	5	6	7
GROUP	OFFSET								
FUNDAMENTAL	0x00	F_IRMS	F_VRMS	F_ACT	F_REACT	F_APP	F_PF	N/A	N/A
H1_INDEX	0x08	H1_IRMS	H1_VRMS	H1_ACT	H1_REACT	H1_APP	H1_PF	H1_IHDN	H1_VHDN
H2_INDEX	0x10	H2_IRMS	H2_VRMS	H2_ACT	H2_REACT	H2_APP	H2_PF	H2_IHDN	H2_VHDN
H3_INDEX	0x18	H3_IRMS	H3_VRMS	H3_ACT	H3_REACT	H3_APP	H3_PF	H3_IHDN	H3_VHDN
H4_INDEX	0x20	H4_IRMS	H4_VRMS	H4_ACT	H4_REACT	H4_APP	H4_PF	H4_IHDN	H4_VHDN
H5_INDEX	0x28	H5_IRMS	H5_VRMS	H5_ACT	H5_REACT	H5_APP	H5_PF	H5_IHDN	H5_VHDN
H6_INDEX	0x30	H6_IRMS	H6_VRMS	H6_ACT	H6_REACT	H6_APP	H6_PF	H6_IHDN	H6_VHDN
H7_INDEX	0x38	H7_IRMS	H7_VRMS	H7_ACT	H7_REACT	H7_APP	H7_PF	H7_IHDN	H7_VHDN
H8_INDEX	0x40	H8_IRMS	H8_VRMS	H8_ACT	H8_REACT	H8_APP	H8_PF	H8_IHDN	H8_VHDN
H9_INDEX	0x48	H9_IRMS	H9_VRMS	H9_ACT	H9_REACT	H9_APP	H9_PF	H9_IHDN	H9_VHDN
H10_INDEX	0x50	H10_IRMS	H10_VRMS	H10_ACT	H10_REACT	H10_APP	H10_PF	H10_IHDN	H10_VHDN
H11_INDEX	0x58	H11_IRMS	H11_VRMS	H11_ACT	H11_REACT	H11_APP	H11_PF	H11_IHDN	H11_VHDN
H12_INDEX	0x60	H12_IRMS	H12_VRMS	H12_ACT	H12_REACT	H12_APP	H12_PF	H12_IHDN	H12_VHDN

Figure 28-6: HAE Results Memory

HAE Results Upper Byte ID

The HAE results are stored in a 24-bit wide memory, as shown in Results Memory.

When the system bus moves the results into memory, the HAE hardware appends a unique CHANNEL:INDEX identifier to the upper byte of each results location. The intent of the ID byte is to help in data parsing of HAE results.

The 32-bit appended result is as follows:

CHANNEL[3:0] ||	INDEX[3:0] ||	RESULTS[23:0]

The CHANNEL in the upper byte corresponds to nn in Hnn_INDX of that particular results location. The INDX in the upper byte corresponds to the INDX within the Hnn_INDX grouping.

For example, location H7_PF has 0x75 in the upper byte of the system bus transfer of that HAE results memory location.

HAE Result Ranges and Formats

The HAE results, stored in the results memory, have formats appropriate for the given measurement. Measurements accuracy is relative to the full scale value of the input samples, and the assumed full scale is +/-6,000,000. If the full scale is above this value, overflow can occur and the results are undefined. A lower full scale limits the dynamic range accuracy of the measurements. For example, a full scale, which is 50% lower than the assumed 6,000,000 full scale, has the dynamic range reduced by 50%. Potentially, this result can be better or worse, as determined by the ADC noise floor supplying the input samples.

The HAE results are as linear as the input samples, within the accuracy ranges specified earlier. The results assume a full-scale input sample of ~=/-6,000,000 and sufficiently low ADC noise floor. Evaluate the system using your specific ADC and full scale specifications to predict the HAE range of accuracy and expected results.

The RMS values of voltage and current calculated from HAE are exactly not same as the theoretical RMS values. The ratio between HAE calculated RMS value and the theoretical value is around 1.11. A gain factor should be added in the software so as to match the theoretical and HAE calculated RMS values. Calibration can be done by calculating the ratio between HAE calculated RMS value and the theoretical one for a particular amplitude of V and I signal and line frequency.

The following list gives the HAE result formats.

- I_{RMS} and V_{RMS}: both fundamental and harmonic I_{RMS} and V_{RMS} are unsigned magnitudes with full scale of ~4,200,000
- Active and reactive power: both active and reactive powers are signed numbers with full scale of ~+/-4,200,000
- Apparent power: apparent power is an unsigned magnitude with full scale of ~4,200,000
- Power factor: each LSB of the fundamental and harmonic power factors equates to a weight of 2^{-23}. Hence, the maximum register value of 0x7FFFFF equates to a power factor value of 1. The minimum register value of 0x800000 corresponds to a power factor of -1. If the power factor is outside of the -1 to +1 range because of offset and gain calibrations, the result is set at -1 or +1. The result depends on the sign of the fundamental reactive power.
- I_{HD+n}/V_{HD+n}: the harmonic distortion plus noise ratios are computed using the RMS of the fundamental and the RMS of the harmonic under analysis. In other words, the ratio only covers the particular harmonic under analysis versus the fundamental. The ratios are stored as 24-bit values in 3.21 signed format. The ratios are limited to +3.9999, and all greater results are clamped to it. The HD+n ratios cannot be negative.

HAE Operating Modes

The HAE uses the DMA interface to transfer data to and from system memory. The HAE configuration registers enable the module and calibrate its frequency (clock) divide and other parameters, as described in HAE Programming Model. The HAE triggers and status signals indicate system events and errors.

HAE Data Transfer Modes

The HAE uses the RX DMA interface to transfer data from system memory. Samples for the current (I) and voltage (V) channels of the AFE or SINC filter compose the data: two `WORDS` are transferred each 8 kHz sample period into the HAE (I and V).

The HAE uses the TX DMA interface to transfer data to system memory. The fundamental and selected harmonic results compose the data: up to 13 (fundamental + 12 harmonics) x 8 = 104 `DWORD`s are transferred each 8K sample period.

See Data Transfer Module for more information.

HAE Event Control

The HAE module uses DMA to transfer samples to and data from system memory. The HAE also can use TX and RX events to time the arrival of input samples and extract the results by the MCU:

- The MCU uses the RX event (`HAE_STAT.RXIRQ`) as an IRQ to time when to write the I and V samples into the HAE.
- The MCU uses the TX event (`HAE_STAT.TXIRQ`) as an IRQ to time when to read the HAE results.

HAE Interrupt Signals

The interrupt signals to the HAE module include:

- The RX interrupt to time the delivery of waveform samples as inputs to the HAE.
- The TX interrupt to time when the HAE results memory is ready with new values for the current sample.

The HAE generates the RX interrupts at a rate specified by the `HAE_CFG1.STARTDIV` bit field and the TX interrupts at a rate specified by the `HAE_CFG2.UPDATE` bit field. Refer to Data Transfer Module for more information.

HAE Status and Error Signals

The trigger and status signals to the HAE module include:

- `HAE_STAT.RDY` (HAE ready status). When the bit is set, the HAE is fully accessible.
- `HAE_STAT.RXIRQ` (RX IRQ status). The bit mirrors the RX interrupt.
- `HAE_STAT.TXIRQ` (TX IRQ status). The bit mirrors the TX interrupt.

The status bit requires a 1 to clear and reenable the corresponding interrupt for the next sample period. Refer to Data Transfer Module for more information.

HAE Programming Model

The following sections provide basic procedures for configuring various HAE operations.

Current and Voltage data can be transferred from system memory to HAE and HAE results can be transferred to system memory using core and DMA accesses. DMA transfers can be set up to transfer a configurable number of I and V samples and HAE results between HAE and system memory automatically. Core-driven transfers use HAE interrupts to signal the processor core to provide I and V data to the sample registers and read the HAE results from the system memory.

The following sections provide recommended programming guidelines to be followed for core and DMA mode.

Configuring the HAE for DMA Transfers

- Enable HAE operations by configuring the `HAE_CFG2.EN` bit.
- Poll for HAE ready status from the `HAE_STAT.RDY` bit configuration.
- Initialize the HAE configuration registers to configure update and settle rates, HPF, integrator for di/dt sensor and line frequency.
- Choose the harmonic channels to be monitored by configuring the `HAE_H[nn]_INDX` registers and enable channels by configuring the `HAE_CFG3.CHANEN` bits.
- Configure the RX DMA channel 0 and the RX DMA channel 1 to receive I and V samples. Configure the TX DMA channel 0 to transmit harmonic calculation results.
- Set the HAE clock divider using the `HAE_CFG1.STARTDIV` field to 8 kHz.
- Set the `HAE_RUN` register.

Configuring the HAE for Core Transfers

- Enable HAE operations by configuring the `HAE_CFG2.EN` bit.
- Poll for HAE ready status using the `HAE_STAT.RDY` bit.
- Initialize the HAE configuration registers to configure update and settle rates, HPF, integrator for di/dt sensor and line frequency.
- Enable receive and transmit IRQ using the `HAE_CFG0` register.
- Configure the `HAE_VLEVEL` register depending on input voltage magnitude requirements.
- Configure the `HAE_H[nn]_INDX` registers to choose the harmonic channels to monitor. Enable the channels using the `HAE_CFG3.CHANEN` bits.
- Set the HAE clock divider using the `HAE_CFG1.STARTDIV` field to 8 kHz.

- Set the HAE_RUN register.
- At each receive IRQ (HAE_STAT.RXIRQ = 1), write I and V sample values to the HAE_ISAMPLE and the HAE_VSAMPLE registers respectively.
- At each transmit IRQ (HAE_STAT.TXIRQ = 1), read the HAE results from the HAE result RAM. The address is provided by the HAE_RESULTS_RAM register.

HAE Programming Concepts

Using the HAE features to their greatest potential requires an understanding of some HAE-related concepts.

- Theory of Operation
- Initialization
- Harmonic Calculations
- Configuring Harmonic Calculations Update Rate

Theory of Operation

The HAE theory of operation is as follows.

Consider a nonsinusoidal AC system supplied by a voltage, v(t), that consumes the current, i(t). Then,

$$v(t) = \sum_{k=1}^{\infty} V_k \sqrt{2} \sin(k\omega t + \varphi_k)$$

$$i(t) = \sum_{k=1}^{\infty} I_k \sqrt{2} \sin(k\omega t + \gamma_k)$$

where:

- V_k, I_k are the RMS voltage and current, respectively, of each harmonic.
- Φ_k, γ_k are the phase delays of each harmonic.
- ω is the angular velocity at the fundamental (line) frequency f.

The HAE harmonic calculations are specified for line frequencies 45–65 Hz.

The maximum number of harmonics which the HAE can accept for a particular line frequency will depend upon the bandwidth of the input waveforms supplied.

When the HAE analyzes I and V samples, the following metering quantities are computed:

- Fundamental current RMS: I_1
- Fundamental voltage RMS: V_1
- RMS of up to 12 harmonics of the current channel: I_n, n = 2, 3,…, 12
- RMS of up to 12 harmonics of the voltage channel: V_n, n = 2, 3,…, 12

- Fundamental active power: $P_1 = V_1 I_1 cos(\varphi_1 - \gamma_1)$
- Fundamental reactive power: $Q_1 = V_1 I_1 sin(\varphi_1 - \gamma_1)$
- Fundamental apparent power: $S_1 = V_1 I_1$
- Power factor of the fundamental:

  $$pf_1 = \text{sgn}(Q_1) \cdot \frac{P_1}{S_1}$$

  Active power of up to 12 harmonics:

  $P_n = V_n I_n \cos(\varphi_n - \gamma_n)$, n = 2, 3,…, 12
- Reactive power of up to 12 harmonics:

  $Q_n = V_n I_n \sin(\varphi_n - \gamma_n)$, n = 2, 3,…, 12
- Apparent power of up to 12 harmonics:

  $S_n = V_n I_n$, n = 2, 3, …, 12
- Power factor of up to 12 harmonics:

  $$pf_n = \text{sgn}(Q_n) \cdot \frac{P_n}{S_n}, \text{ n = 2, 3,…, 12}$$

- Total harmonic distortion of the current channel:

  $$(THD)_I = \frac{\sqrt{I^2 - I_1^2}}{I_1}$$

- Total harmonic distortion of the voltage channel:

  $$(THD)_V = \frac{\sqrt{V^2 - V_1^2}}{V_1}$$

- Harmonic distortion of up to 12 harmonics on the current channel:

  $$HD_{I_n} = \frac{I_n}{I_1}, \text{ n = 2, 3,…, 12}$$

- Harmonic distortion of up to 12 harmonics on the voltage channel:

  $$HD_{V_n} = \frac{V_n}{V_1}, \text{ n = 2, 3,…, 12}$$

Initialization

The HAE typically is initialized with parameters and settings for a given usage scenario, then is interrupt-driven when new results are available.

The HAE programming depends on whether the line frequency is 50 Hz or 60 Hz. If the external sensor is a di/dt type, there are also some differences. The ***HAE Initialization*** flowchart shows the initialization sequence.

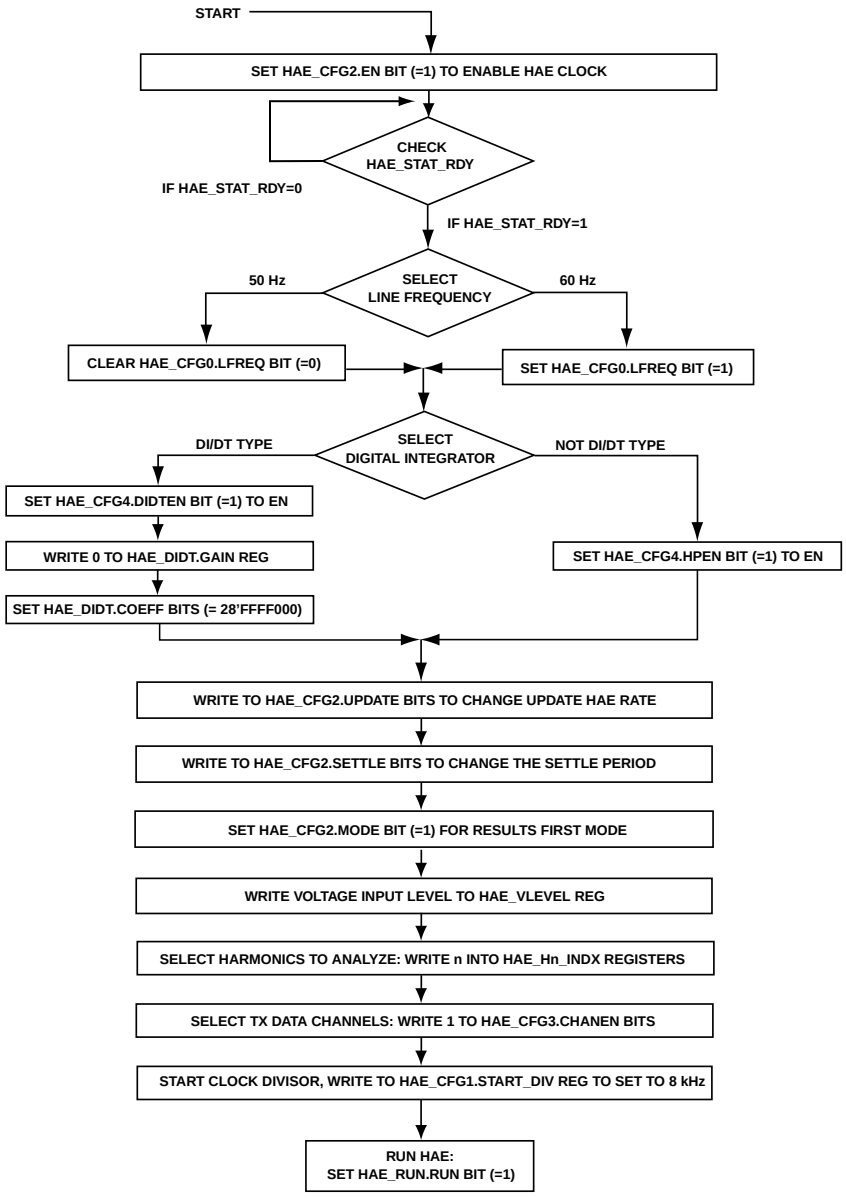

Figure 28-7: HAE Initialization

Harmonic Calculations

When the harmonic engine runs, it computes information about the fundamental and up to 12 harmonics. The HAE simultaneously monitors the indexes of the additional 12 harmonics, provided by the 8-bit registers `HAE_H[nn]_INDX`. Write the index of the harmonic into the register for the harmonic to be monitored. If the second harmonic is monitored, write `2`. If harmonic 51 is desired, write `51`. The HAE always monitors the fundamental component, independent of the values written into `HAE_H[nn]_INDX`. Therefore, if one of these registers is

equal to 1, the HAE monitors the fundamental components multiple times. The maximum index allowed in the HAE_H[nn]_INDX registers is 63.

As a reference, the ***Harmonic Engine Outputs and Registers where Values are Stored*** table presents the harmonic engine outputs and registers that store the outputs.

Table 28-3: Harmonic Engine Outputs and Registers where Values are Stored

Quantity	*Definition*	*HAE Registers*
RMS of the Fundamental Component	V_1, I_1	F_VRMS, F_IRMS
RMS of a Harmonic Component	V_n, I_n, n = 2, 3,…, 12	Hnn_VRMS, Hnn_XIRMS
Active Power of the Fundamental Component	$P_1 = V_1 I_1 \cos(\varphi_1 - \gamma_1)$	F_ACT
Active Power of a Harmonic Component	$P_n = V_n I_n \cos(\varphi_n - \gamma_n)$, n = 2, 3,…, 12	Fnn_ACT
Reactive Power of the Fundamental Component	$Q_1 = V_1 I_1 \sin(\varphi_1 - \gamma_1)$	F_REACT
Reactive Power of a Harmonic Component	$Q_n = V_n I_n \sin(\varphi_1 - \gamma_1)$, n = 2, 3,…, 12	Hnn_REACT
Apparent Power of the Fundamental Component	$S_1 = V_1 I_1$	F_APP
Apparent Power of a Harmonic Component	$S_n = V_n I_n$, n = 2, 3, …, 12	Hnn_APP
Power Factor of the Fundamental Component	$pf_1 = \mathrm{sgn}(Q_1) \cdot \frac{P_1}{S_1}$	F_PF
Power Factor of a Harmonic Component	$pf_n = \mathrm{sgn}(Q_n) \cdot \frac{P_n}{S_n}$, n = 2, 3,…, 12	Hnn_PF
Harmonic Distortion of a Harmonic Component	$HD_{V_n} = \frac{V_n}{V_1}, HD_{I_n} = \frac{I_n}{I_1}$, n = 2, 3,…, 12	Hnn_VHDN, Hnn_IHDN

Configuring Harmonic Calculations Update Rate

The harmonic engine functions at an 8 kHz rate. From the moment the HAE_CFG2 register is initialized, and the harmonic indexes are set in the HAE_H[nn]_INDX index registers, the HAE calculations take typically 750 mSec to settle within the specification parameters.

The HAE module uses the HAE_CFG2.UPDATE bits to manage the update rate of the output registers for the harmonic engine. It manages the update rate independent of the calculations rate of 8 kHz for the engine. The default value of 000 means that the registers are updated every 125 uSec (8 kHz rate). Other update periods are: 250 uSec (001), 1 mSec (010), 16 mSec (011), 128 mSec (100), 512 mSec (101), 1.024 mSec (110). If the HAE_CFG2.UPDATE bits are 111, the harmonic calculations are disabled.

The HAE module provides two ways to manage the harmonic computations. It enables the first approach when bit HAE_CFG2.MODE is cleared to its default value of 0. The state sets status bit HAE_STAT.TXIRQ to 1 after a certain

period and then every time the harmonic calculations are updated at HAE_CFG2.UPDATE frequency. This functionality allows an external microcontroller to access the harmonic calculations only after they have settled. The HAE uses the state of bits HAE_CFG2.SETTLE to determine the time period. The possible values of settling time of the harmonic calculations are 512 mSec (00), 768 mSec(01), 1024 mSec (10), and 1280 mSec (11).

The HAE module enables the second approach when bit HAE_CFG2.MODE is set to 1. The state sets the status bit HAE_STAT.TXIRQ to 1 every time the harmonic calculations are updated at the HAE_CFG2.UPDATE frequency, without waiting for the harmonic calculations to settle. This functionality allows an external microcontroller to access the harmonic calculations immediately after starting. A write to the HAE_STAT register clears the status bit. The corresponding bit (HAE_STAT.TXIRQ) is set to 1.

ADSP-CM40x HAE Register Descriptions

Harmonic Analysis Engine (HAE) contains the following registers.

Table 28-4: ADSP-CM40x HAE Register List

Name	Description
HAE_CFG0	Configuration 0 Register
HAE_CFG1	Configuration 1 Register
HAE_CFG2	Configuration 2 Register
HAE_CFG3	Configuration 3 Register
HAE_CFG4	Configuration 4 Register
HAE_DIDT_COEF	DIDT Coefficient Register
HAE_DIDT_GAIN	DIDT Gain Register
HAE_H[nn]_INDX	Harmonic n Index Register
HAE_ISAMPLE	I (Current) Sample Register
HAE_IWAVEFORM	I (Current) Waveform Register
HAE_RESULTS_RAM	Results RAM Register
HAE_RUN	Run Register
HAE_STAT	Status Register
HAE_VLEVEL	Voltage Level Register
HAE_VSAMPLE	V (Voltage) Sample Register
HAE_VWAVEFORM	V (Voltage) Waveform Register

Configuration 0 Register

The HAE_CFG0 register configures high-level interrupts and specifies the line frequency for HAE operations.

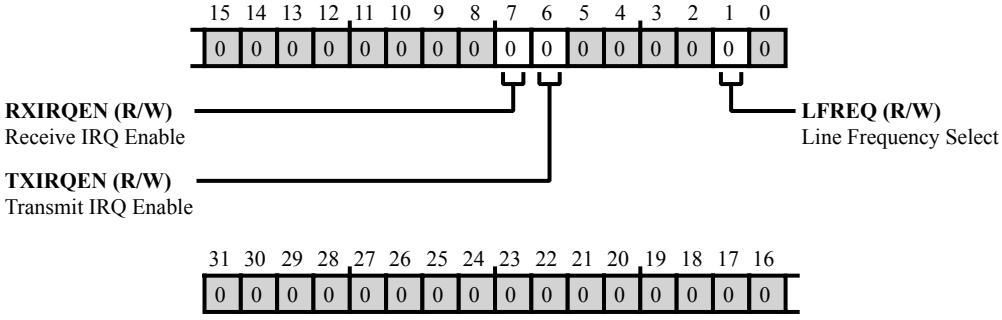

Figure 28-8: HAE_CFG0 Register Diagram

Table 28-5: HAE_CFG0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	RXIRQEN	Receive IRQ Enable. The HAE_CFG0.RXIRQEN bit enables an interrupt, which the HAE triggers on each request for a new input sample on both the I and V channels.	
		0	Disable
		1	Enable
6 (R/W)	TXIRQEN	Transmit IRQ Enable. The HAE_CFG0.TXIRQEN bit enables an interrupt, which the HAE triggers on each result as the result is calculated and ready to transmit.	
		0	Disable
		1	Enable
1 (R/W)	LFREQ	Line Frequency Select. The HAE_CFG0.LFREQ bit specifies the line frequency for the HAE. Set the bit to match the line frequency being analyzed.	
		0	50 Hz
		1	60 Hz

Configuration 1 Register

The HAE_CFG1 register configures the HAE frequency (clock) divider.

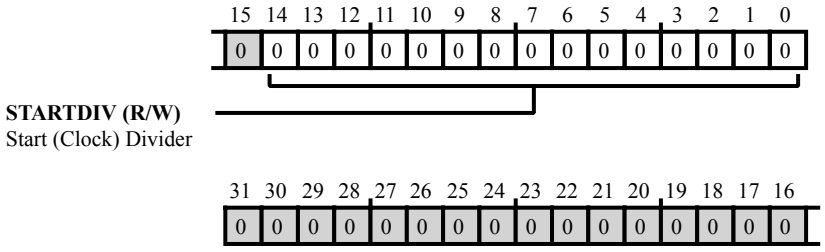

Figure 28-9: HAE_CFG1 Register Diagram

Table 28-6: HAE_CFG1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/W)	STARTDIV	Start (Clock) Divider. The HAE_CFG1.STARTDIV bits provide the sample clock divider. Write the value to divide the main clock down to 8 kHz. The HAE sample loop is determined by SYSCLK / (HAE_CFG1.STARTDIV).

Configuration 2 Register

The HAE_CFG2 register enables and configures HAE operations as related to output results.

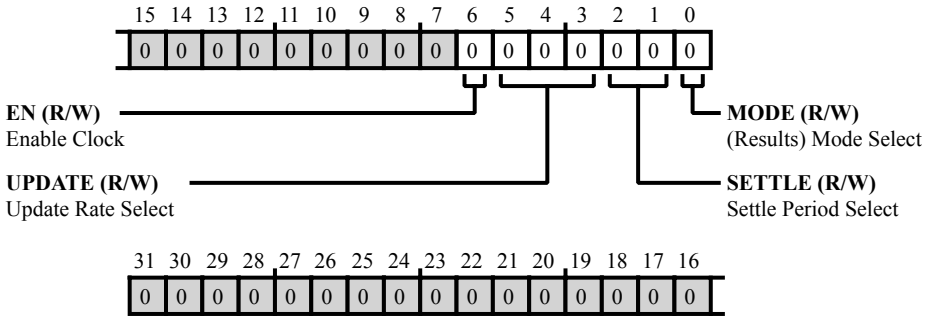

Figure 28-10: HAE_CFG2 Register Diagram

Table 28-7: HAE_CFG2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	EN	Enable Clock. The HAE_CFG2.EN bit enables the HAE the main clock, enabling HAE operation.	
		0	Disable
		1	Enable
5:3 (R/W)	UPDATE	Update Rate Select. The HAE_CFG2.UPDATE bits determine the rate (in microseconds or milliseconds) at which the HAE updates the output results. The HAE_CFG2.UPDATE bits can also disable harmonic calculations.	
		0	125 uSec
		1	250 uSec
		2	1 mSec
		3	16 mSec
		4	128 mSec
		5	512 mSec
		6	1024 mSec
		7	Disable

Table 28-7: HAE_CFG2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2:1 (R/W)	SETTLE	Settle Period Select. The `HAE_CFG2.SETTLE` bits determine the time (in milliseconds) that the HAE waits before producing results. The time is based on an 8K sample count.	
		0	512 mSec
		1	768 mSec
		2	1024 mSec
		3	1280 mSec
0 (R/W)	MODE	(Results) Mode Select. The `HAE_CFG2.MODE` bit determines whether the HAE produces results immediately or waits per the `HAE_CFG2.SETTLE` bit field.	
		0	Results after Settle
		1	Results First

Configuration 3 Register

The HAE_CFG3 register configures HAE data transfer operations by selecting the fundamental and potential of twelve additional harmonic channels. The selected harmonics have their data (results) transferred to memory using DMA. First, the fundamental; followed by the selected channel n, in the order from the lowest to the highest numbered channel. Each selected channel has its eight result words transferred.

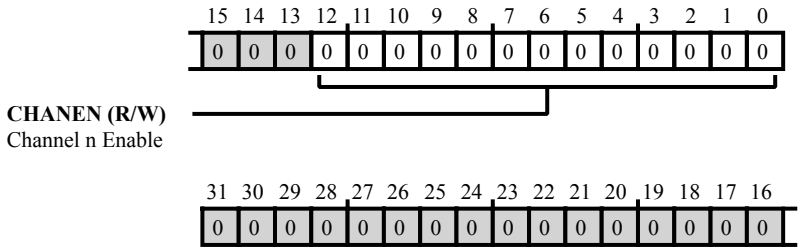

Figure 28-11: HAE_CFG3 Register Diagram

Table 28-8: HAE_CFG3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
12:0 (R/W)	CHANEN	Channel n Enable. Each HAE_CFG3.CHANEN bit enables the fundamental and potential of twelve additional harmonic data channels. The following enumerations apply to each bit. Bit 0 denotes the fundamental channel. Bits 1-12 denote the harmonic channels 1-12, accordingly.	
		0	Disable channel n
		1	Enable channel n

Configuration 4 Register

The HAE_CFG4 register configures the internal digital integrator and high-pass filters (HPS) for HAE operations. The digital integrator is disabled and the filters are enabled by default.

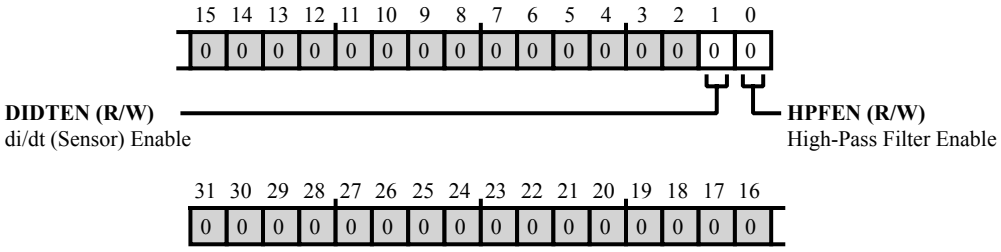

Figure 28-12: HAE_CFG4 Register Diagram

Table 28-9: HAE_CFG4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	DIDTEN	di/dt (Sensor) Enable. The HAE_CFG4.DIDTEN bit enables the internal digital integrator for a di/dt sensor. Set the bit (=1) if the sensor is a di/dt type sensor.	
		0	Disable
		1	Enable
0 (R/W)	HPFEN	High-Pass Filter Enable. The HAE_CFG4.HPFEN bit enables the high-pass filters. Set the bit (=1) unless measuring DC levels.	
		0	Disable
		1	Enable

DIDT Coefficient Register

The HAE_DIDT_COEF register sets the di/dt sensor's coefficient.

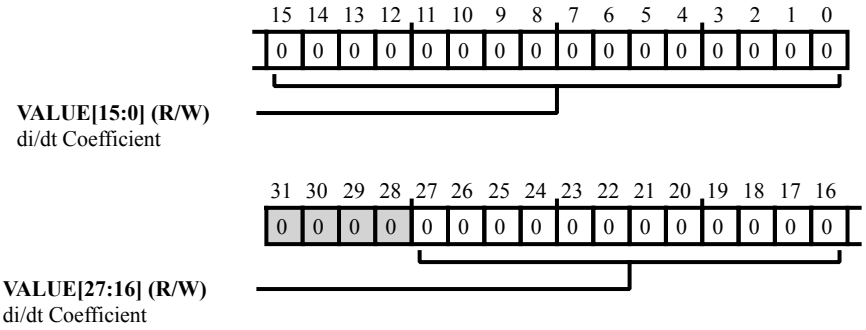

Figure 28-13: HAE_DIDT_COEF Register Diagram

Table 28-10: HAE_DIDT_COEF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:0 (R/W)	VALUE	di/dt Coefficient. The HAE_DIDT_COEF.VALUE bits provide the coefficient for a di/dt type sensor. If the HAE_CFG4.DIDTEN bit is set (=1), set this bit field to 28'FFFF000.

DIDT Gain Register

The HAE_DIDT_GAIN register provides the di/dt sensor's gain.

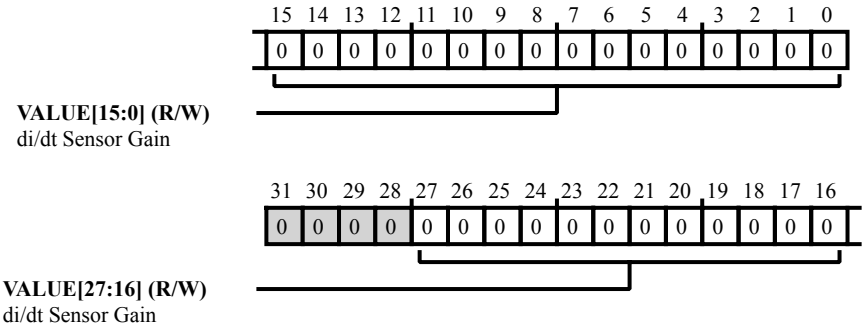

Figure 28-14: HAE_DIDT_GAIN Register Diagram

Table 28-11: HAE_DIDT_GAIN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:0 (R/W)	VALUE	di/dt Sensor Gain. The HAE_DIDT_GAIN.VALUE bits provide the gain for a di/dt type sensor. If the HAE_CFG4.DIDTEN bit is set (=1), set this bit field to 0.

Harmonic n Index Register

The HAE_H[nn]_INDX registers select harmonics for HAE operations. The fundamental always is provided. The harmonic results appear in the results RAM memory region based on the index.

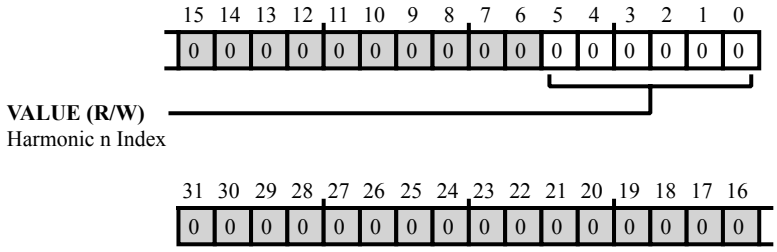

Figure 28-15: HAE_H[nn]_INDX Register Diagram

Table 28-12: HAE_H[nn]_INDX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5:0 (R/W)	VALUE	Harmonic n Index. The HAE_H[nn]_INDX.VALUE bits select the harmonic channel n to analyze. Write the index of the harmonic into the HAE_H[nn]_INDX register for that harmonic to be selected.

I (Current) Sample Register

The HAE_ISAMPLE register provides current (I) input samples for HAE calculations.

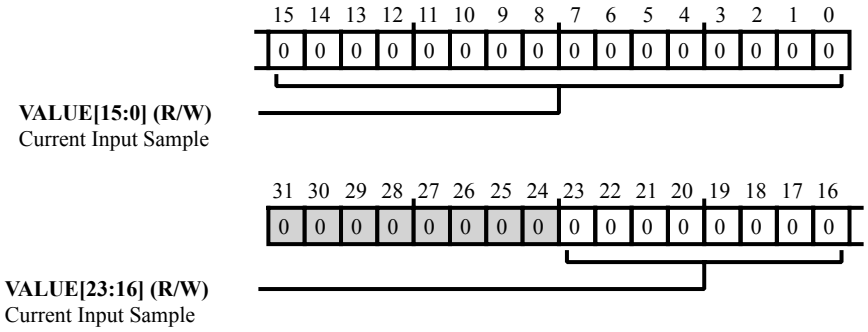

Figure 28-16: HAE_ISAMPLE Register Diagram

Table 28-13: HAE_ISAMPLE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/W)	VALUE	Current Input Sample. The HAE_ISAMPLE.VALUE bits hold the current sample if provided by the MCU. The sample can be timed with the HAE_STAT.RXIRQ bit.

I (Current) Waveform Register

The HAE_IWAVEFORM register holds processed current waveforms produced by the HAE. After some amplitude and phase delay, the waveform follows a sample input.

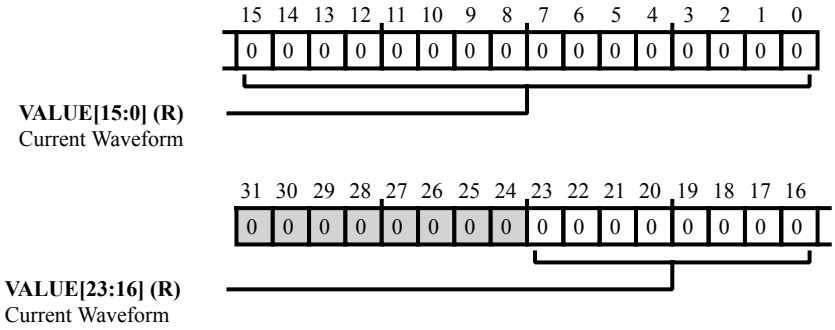

Figure 28-17: HAE_IWAVEFORM Register Diagram

Table 28-14: HAE_IWAVEFORM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/NW)	VALUE	Current Waveform. The HAE_IWAVEFORM.VALUE bits hold the processed current input sample.

Results RAM Register

The HAE_RESULTS_RAM register points to a memory block holding HAE results.

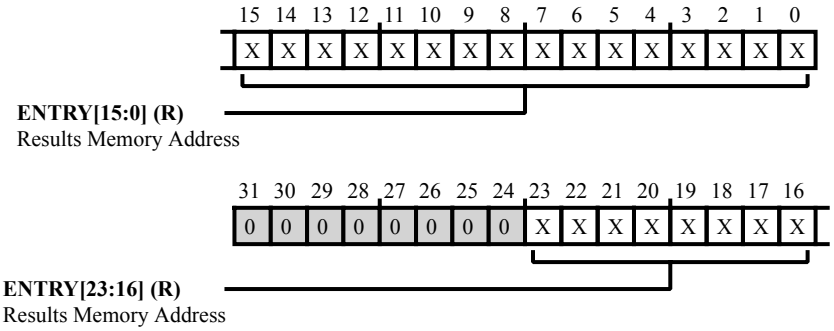

Figure 28-18: HAE_RESULTS_RAM Register Diagram

Table 28-15: HAE_RESULTS_RAM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/NW)	ENTRY	Results Memory Address. The HAE_RESULTS_RAM.ENTRY bits hold the base pointer (address) for the HAE results RAM. The address range is x0c00 - 0x0dff. The size is 128 x 24 bits.

Run Register

The HAE_RUN register starts/idles HAE harmonic calculations.

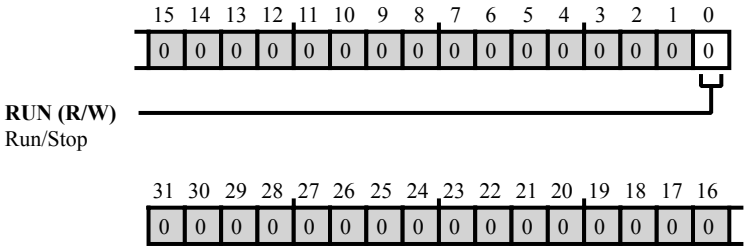

Figure 28-19: HAE_RUN Register Diagram

Table 28-16: HAE_RUN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	RUN	Run/Stop. The HAE_RUN.RUN bit starts the HAE harmonic calculations.	
		0	Stop
		1	Run

Status Register

The HAE_STAT register indicates status for the HAE module and its interrupts.

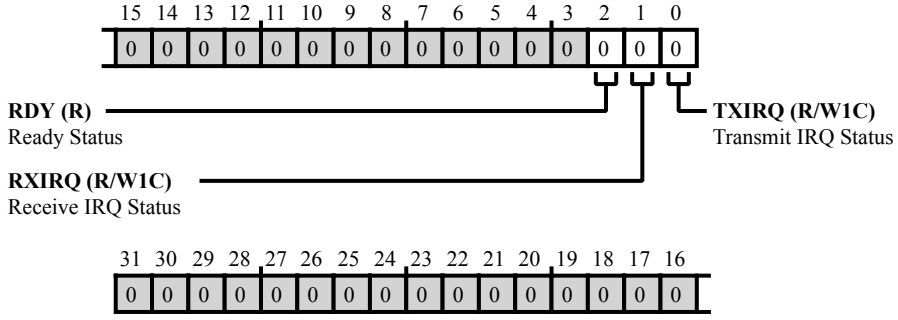

Figure 28-20: HAE_STAT Register Diagram

Table 28-17: HAE_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/NW)	RDY	Ready Status. The HAE_STAT.RDY bit indicates status for the HAE. When the bit is set (=1), the HAE is fully accessible, following the setting of the HAE_CFG2.EN bit.	
		0	No Status
		1	Ready
1 (R/W1C)	RXIRQ	Receive IRQ Status. The HAE_STAT.RXIRQ bit indicates status for an HAE RX interrupt. The bit mirrors the HAE_CFG0.RXIRQEN bit status.	
		0	No Status
		1	Interrupt Detected
0 (R/W1C)	TXIRQ	Transmit IRQ Status. The HAE_STAT.TXIRQ bit indicates status for an HAE TX interrupt. The bit mirrors the HAE_CFG0.TXIRQEN bit status.	
		0	No Status
		1	Interrupt Detected

Voltage Level Register

The HAE_VLEVEL register is used to scale the fixed fundamental voltage level internally. This assists the HAE in locking onto the fundamental voltage channel.

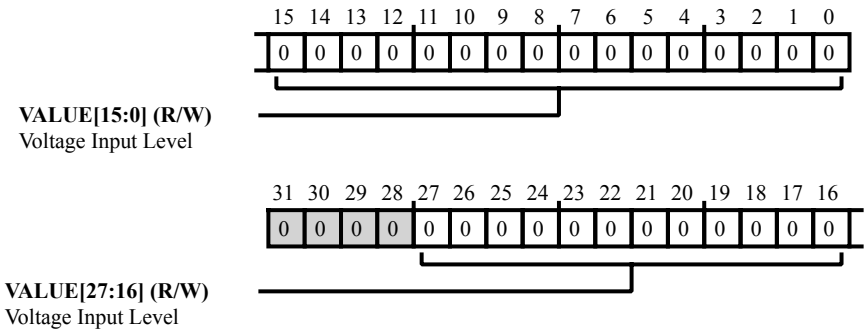

Figure 28-21: HAE_VLEVEL Register Diagram

Table 28-18: HAE_VLEVEL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:0 (R/W)	VALUE	Voltage Input Level. The HAE_VLEVEL.VALUE bits hold a value to scale the fixed fundamental voltage level. Use this formula: VLEVEL = FS/V_{IN} * 5033168, where V_{IN} is the fundamental voltage input level.

V (Voltage) Sample Register

The HAE_VSAMPLE register provides voltage (V) input samples for HAE calculations.

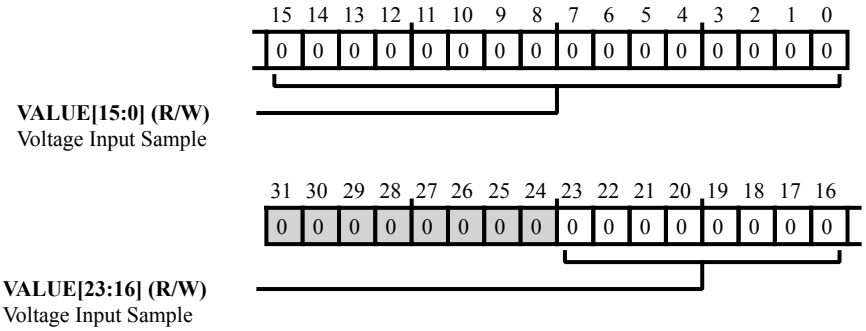

Figure 28-22: HAE_VSAMPLE Register Diagram

Table 28-19: HAE_VSAMPLE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/W)	VALUE	Voltage Input Sample. The HAE_VSAMPLE.VALUE bits hold the voltage sample if provided by the MCU. The sample can be timed with the HAE_STAT.RXIRQ bit.

V (Voltage) Waveform Register

The HAE_VWAVEFORM register contains processed voltage waveforms produced by the HAE. After some amplitude and phase delay, the waveform follows a sample input.

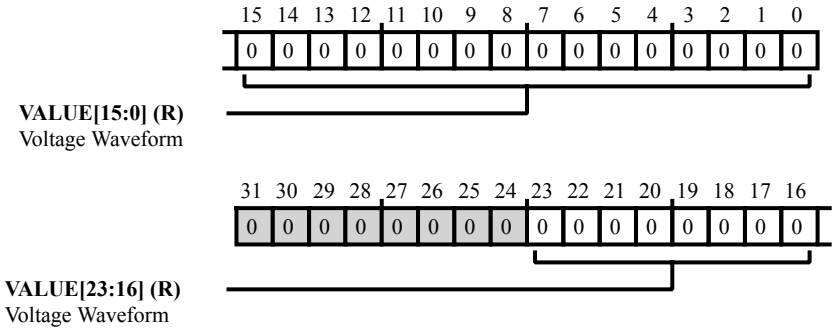

Figure 28-23: HAE_VWAVEFORM Register Diagram

Table 28-20: HAE_VWAVEFORM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/NW)	VALUE	Voltage Waveform. The HAE_VWAVEFORM.VALUE bits hold the processed voltage input sample.

29 Sinus Cardinalis (SINC) Filter

The sinus cardinalis (SINC) filter module processes four independent sigma-delta bit streams by applying a pair of SINC filters to each stream. A SINC filter converts the bit stream from a sigma-delta front-end modulator into a digital word representing the signal level presented to the modulator.

The filter consists of a set of integration and decimation stages implemented directly in logic for efficient execution. The SINC filter supports capture of current or voltage feedback signals from an isolating analog-to-digital converter (ADC). Each modulator bit stream connects to two SINC filters: a primary filter for controlling feedback; a secondary filter for overcurrent detection. The SINC module includes four filter channels and two modulator clock generators.

SINC Filter Features

The SINC features include:

- Four-bit stream filter channels for current or voltage feedback signal processing
- Each channel includes two SINC filter pairs:
  - A primary filter for feedback signal processing
  - A secondary filter for overload detection
- Two modulator clock sources with phase control options
- Configuration of SINC filter channels according to a modulator clock selection
- Programmable order and decimation rates
- Primary filters:
  - Programmable bias and gain with output saturation
  - Dedicated direct memory access (DMA) channels with data interleaving and programmable data ready output triggers
- Secondary filters:
  - Detecting a fault when signals exceed amplitude and duration values

  - Registers preserving the eight most recent samples before a fault event
- Multiple interrupt trigger sources for overload fault and data overflow events

SINC Functional Description

The SINC filter has the functionality described as follows.

Digital filter

The filter removes the modulator sample clock and recovers a digital value of the sampled signal.

DC gain and data resolution

The DC gain of the digital filter is a function of the order and decimation rate.

Frequency response

The frequency response of the filter depends on the order, decimation rate, and modulator clock frequency.

Output scaling

The output scaling and postprocessing functions embedded in the SINC filter blocks differ, depending on the function.

ADSP-CM40x SINC Register List

The SINC filter module processes four independent sigma-delta bit streams by applying a pair of SINC filters to each stream. A SINC filter converts the bit stream from a sigma-delta front-end modulator into a digital word representing the signal level presented to the modulator. Each modulator bit stream connects to two SINC filters: a primary filter for controlling feedback, and a secondary filter for overcurrent detection. A set of registers governs SINC operations. For more information on SINC functionality, see the SINC register descriptions.

Table 29-1: ADSP-CM40x SINC Register List

Name	Description
SINC_BIAS0	Bias for Group 0 Register
SINC_BIAS1	Bias for Group 1 Register
SINC_CLK	Clock Control Register
SINC_CTL	Control Register
SINC_HIS_STAT	History Status Register
SINC_LEVEL0	Level Control for Group 0 Register
SINC_LEVEL1	Level Control for Group 1 Register

Table 29-1: ADSP-CM40x SINC Register List (Continued)

Name	Description
SINC_LIMIT0	(Amplitude) Limits for Secondary Filter 0 Register
SINC_LIMIT1	(Amplitude) Limits for Secondary Filter 1 Register
SINC_LIMIT2	(Amplitude) Limits for Secondary Filter 2 Register
SINC_LIMIT3	(Amplitude) Limits for Secondary Filter 3 Register
SINC_P0SEC_HIST[n]	Pair 0 Secondary (Filter) History n Register
SINC_P1SEC_HIST[n]	Pair 1 Secondary (Filter) History n Register
SINC_P2SEC_HIST[n]	Pair 2 Secondary (Filter) History n Register
SINC_P3SEC_HIST[n]	Pair 3 Secondary (Filter) History n Register
SINC_PHEAD0	Primary (Filters) Head for Group 0 Register
SINC_PHEAD1	Primary (Filters) Head for Group 1 Register
SINC_PPTR0	Primary (Filters) Pointer for Group 0 Register
SINC_PPTR1	Primary (Filters) Pointer for Group 1 Register
SINC_PTAIL0	Primary (Filters) Tail for Group 0 Register
SINC_PTAIL1	Primary (Filters) Tail for Group 1 Register
SINC_RATE0	Rate Control for Group 0 Register
SINC_RATE1	Rate Control for Group 1 Register
SINC_STAT	Status Register

ADSP-CM40x SINC Interrupt List

Table 29-2: ADSP-CM40x SINC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
70	SINC0_STAT	SINC0 Status	Level	

ADSP-CM40x SINC Trigger List

Table 29-3: ADSP-CM40x SINC Trigger List Masters

Trigger ID	Name	Description	Sensitivity
36	SINC0_P0_OVLD	SINC0 Pair 0 Overload Indicator	Edge
37	SINC0_P1_OVLD	SINC0 Pair 1 Overload Indicator	Edge
38	SINC0_P2_OVLD	SINC0 Pair 2 Overload Indicator	Edge
39	SINC0_P3_OVLD	SINC0 Pair 3 Overload Indicator	Edge

Table 29-3: ADSP-CM40x SINC Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
40	SINC0_DATA0	SINC0 Data Move 0 Complete	Edge
41	SINC0_DATA1	SINC0 Data Move 1 Complete	Edge

Table 29-4: ADSP-CM40x SINC Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
57	SINC0_SYNC0	SINC0 Synchronization Input 0	None
58	SINC0_SYNC1	SINC0 Synchronization Input 1	None

SINC Definitions

To make the best use of the SINC, it is useful to understand the following terms.

Decimation

Decimation is the process of discarding samples from a data stream.

Decimation Rate

The decimation rate is the ratio of the filter input data rate to the filter output data rate.

Filter Order

The SINC filter order is the number of integration and decimation stages in the filter.

Modulator Order

The modulator order is the number of comparator and integrator stages in a sigma-delta modulator.

Sigma-Delta Modulator

The sigma-delta modulator is an oversampling analog to digital conversion circuit that generates a digital bit stream whose pulse density is proportional to the analog voltage presented to the input.

SINC Block Diagram

The *SINC Block Diagram* figure shows the functional blocks within the SINC.

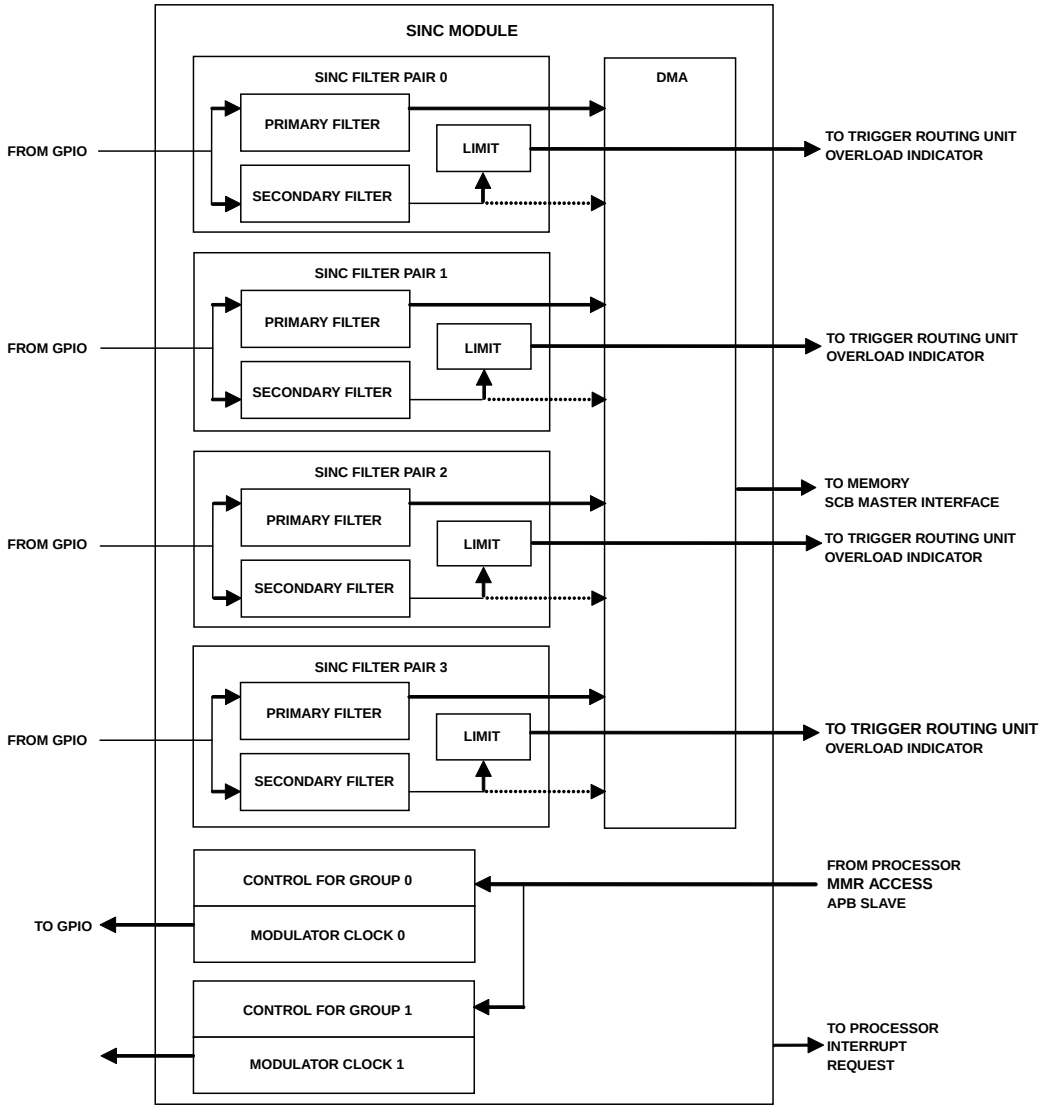

Figure 29-1: SINC Block Diagram

The block diagram shows four SINC filter pairs (`SINC0-3`), two modulator clock sources, and two banks of control registers (units). The module accepts four sigma-delta bit streams from the GPIO input pins and directs the modulator clock source of GROUP 0 to the GPIO output pin. A pulse-width modulation (PWM) signal synchronizes the modulator clocks to optimize system performance. Each SINC filter pair includes the primary filter, secondary filter, DMA interface, and overload limit detection functions.

The primary SINC filter transmits its data to memory using DMA. The secondary SINC filter generates overload signals, which can be routed through the trigger routing unit (TRU) to trip a PWM modulator and generate an interrupt.

The SINC filter pairs can be assigned to either set of control units, where multiple channels of current or voltage-feedback share common filter parameters. The primary filters generate high-resolution signals for closing the feedback control loop. The secondary filters are for rapid-overload fault detection, require lower resolution, but a faster

response. The primary and secondary filters have programmable order and decimation rates. The primary filters also have the programmable output gain stage, while the secondary filters have the programmable overload limit thresholds.

To use the primary and secondary filters, set up the filter parameters once, prior to using the filters. The feedback control algorithm reads the data from the primary filter directly from memory. A PWM interrupt signal can generate the algorithm timing signal, or the SINC module generates a data trigger. The data history of the secondary filter is saved in buffer registers once an overload fault signal is detected. The data history supports fault diagnostics.

SINC Architectural Concepts

The architecture of the SINC includes the following:

- Digital Filter
- DC Gain and Data Resolution
- Frequency Response
- Output Scaling

Digital Filter

The SINC filter has a transfer function that lends itself to an implementation in digital logic, using a series of summation and decimation functions. The filter removes the modulator sample clock and recovers a digital value of the sampled signal. The filter design matches a bipolar SD modulator. The design produces a 50% pulse density for a 0V input, over 50% for positive inputs and less than 50% for negative inputs.

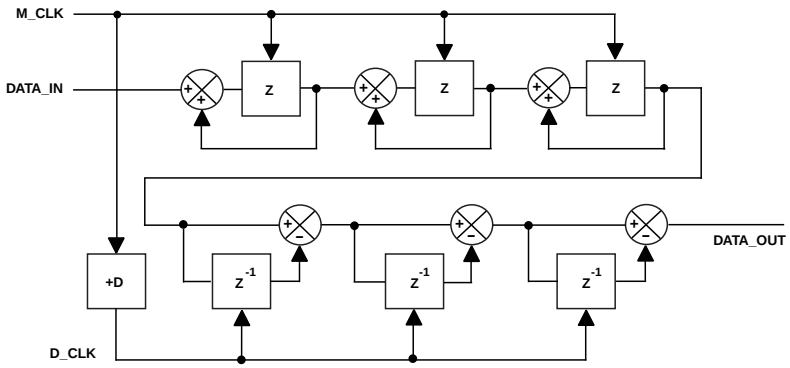

Figure 29-2: SINC Digital Filter

The digital filter is a set of accumulators driven by the modulator clock (`M_CLK`), followed by a set of differentiators driven by the decimation clock (`D_CLK`). The input accumulators convert the input bit stream into a multibyte word, while the output differentiators derive the average 1's density of the bit stream. The number of accumulator and differentiator stages can be three or four, depending on the order of the filter. The DC gain and bandwidth of the filter are functions of the filter order (`O`) and the decimation rate (`D`), which is the ratio of the modulator to the decimation clock.

The calculation of the transfer function of the SINC filter includes the product of the transfer functions for the accumulators and differentiators, and in the z domain. The following equation gives the calculation:

$$H(z) = \left[\frac{1}{D} \text{ x } \frac{1 - z^{-D}}{1 - z^{-1}}\right]^{O}$$

DC Gain and Data Resolution

The DC gain of the digital filter is a function of the order and decimation rate. At 100% ones density input, each accumulator stage counts D pulses, and the gain of the filter is given as follows:

$$G_{dc} = D^{O}$$

The higher the decimation rate, the higher the resolution of the output data. The number of usable data bits is a function of the SNR; the ***Filter Order versus Decimation*** table shows ENOB versus the decimation rate.

Table 29-5: ENOB versus Decimation

Decimation		4	5	6	7	8	16	32	64	128	256
O = 3	SNR (dB)	6.42	11.47	16.41	20.57	23.55	35.02	48.59	62.26	76.46	89.59
	ENOB	0.8	1.6	2.4	3.1	3.6	5.5	7.8	10.0	12.4	14.6
O = 4	SNR (dB)	9.08	14.77	19.78	23.41	25.9	38.05	51.29	64.67	79.15	
	ENOB	1.2	2.1	3.1	3.6	4.0	6.0	8.2	10.4	12.8	

Notes: ENOB versus order and decimation rate.

Test conditions are for a 1.22 kHz tone and a 10 MHz modulator.

Frequency Response

The frequency response of the filter depends on the order, decimation rate, and modulator clock frequency, f_M. The equation is obtained by substituting $e^{j \text{ x } Ts}$ for z in the transfer function, where Ts is the period of the modulator clock:

$$H\left(e^{j\frac{\omega}{f_M}}\right) = \left[\frac{1}{D} \times \frac{\sin\left(D\frac{\omega}{2f_M}\right)}{\sin\left(\frac{\omega}{2f_M}\right)} \times e^{-j(D-1)\frac{\omega}{2f_M}}\right]^{O}$$

The filter has a linear phase response with a constant group delay given by:

$$\tau_d = \left(\frac{D-1}{2}\right)\frac{O}{f_M}$$

The ***Frequency Response*** plots show zeros at multiples the decimation frequency, where the sin term in the numerator goes to zero. This response makes it possible to remove some PWM ripple components from the motor current waveform by matching the decimation frequency to the PWM switching frequency. There are some limitations at

lower PWM frequencies based on available decimation rates. High decimation rates limit the bandwidth of the control loop because of the phase delay, which is 3π radians at the decimation frequency.

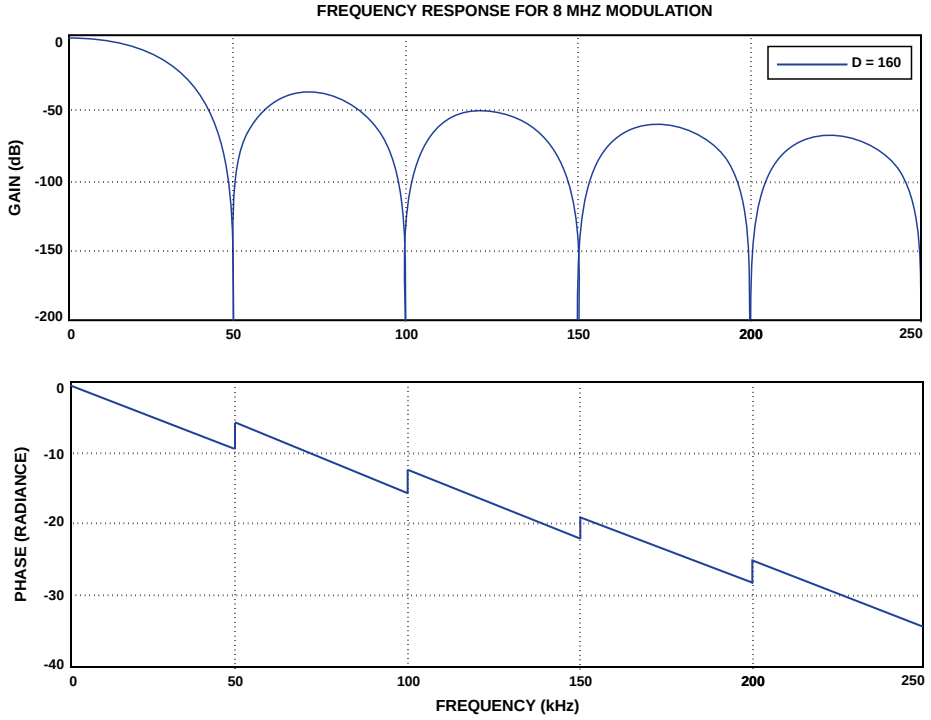

Figure 29-5: Frequency Response

Output Scaling

The output scaling and postprocessing functions embedded in the SINC filter blocks differ, depending on the function. The primary filter used for feedback signal processing includes the output bias and scaling blocks to present a 16-bit signed integer to the control code. The scaling is required at decimation rates higher than 32 to keep the lower 16 bits of the output word.

The secondary filter supports overload detection functions. The secondary filter can detect signals crossing maximum and minimum thresholds. It has a glitch filter that only accepts faults with a minimum number of counts (c) within a certain count window (w). The secondary filter has no output scaling, so the minimum and maximum values in the overload registers must be calculated from the DC gain of the secondary filter. The response time to a step input is approximately 2 x o decimation clock cycles.

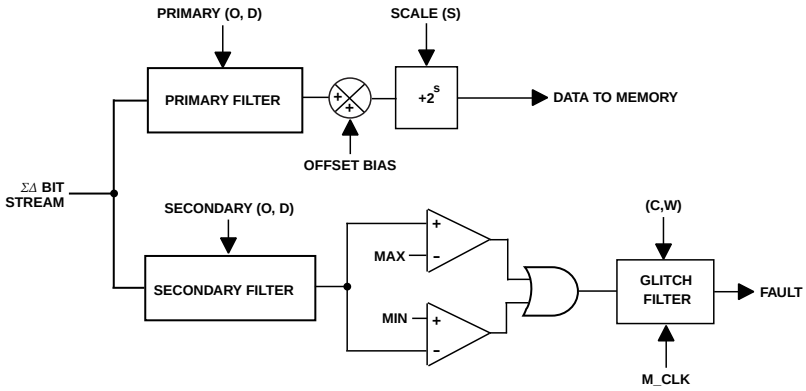

Figure 29-4: Output Scaling

SINC Operating Modes

The SINC filter module has only one operating mode. The module generates the clock source for a sigma-delta modulator analog front end and filters the output data stream for the modulator. The primary SINC filter transfers its data to memory through DMA. The secondary SINC filter output generates an overload trigger signal that the SINC filter module can use as a PWM trip signal. The SINC control registers enable the module and set up the modulator clock sources, filter parameters, DMA transfers, and interrupts masks, as described in SINC Programming Model.

SINC Data Transfer Modes

The only mode of data transfer between the primary SINC filter and memory is through DMA (see Primary DMA Configuration and Data Interrupts). Reading the history registers for the secondary filter is the only way to transfer data between the secondary SINC filter and memory. (see Overload Detection).

SINC Signal Modes

The SINC filter has an interrupt signal and a number of triggers and status signals to indicate system events and errors.

- Primary data transfer trigger:

  The SINC filter can generate a trigger after a user-specified number of primary output sets are transferred to memory. There is one trigger source for each filter group. See Primary DMA Configuration and Data Interrupts for more information.

- Secondary data overload trigger

  The SINC filter can generate a trigger when one of the secondary filters detects an overload condition. There is one trigger source for each secondary filter. See Overload Detection for more information.

- SINC status bits:

The SINC status bits indicate secondary filter overload errors, primary filter saturation errors, primary filter transfer count exceeded, and primary filter data buffer errors.

- Secondary filter overload errors:

  A number of status bits indicate the type of error and the filter channel when a secondary filter detects an overload condition. The status bits `SINC_STAT.GLIM0` and `SINC_STAT.GLIM1` indicate the control group of the secondary filter that detected the overload. The status bits `SINC_STAT.MAX0` through `SINC_STAT.MAX3` indicate when the error a maximum limit on one of the secondary filter channels is passed, causing the error. The status bits `SINC_STAT.MIN0` through `SINC_STAT.MIN3` indicate when a minimum limit on one of the secondary filter channels is passed, causing the error.

- Primary filter data saturation errors:

  A number of status bits indicate the group and filter channel when the SINC filter detects data saturation . The status bits `SINC_STAT.GSAT0` and `SINC_STAT.GSAT1` indicate the filter control group when the SINC filter detects data saturation. The status bits `SINC_STAT.PSAT0` through `SINC_STAT.PSAT3` indicate a primary filter channel that detects data saturation.

- Primary filter transfer count exceeded:

  The status bits `SINC_STAT.PCNT0` and `SINC_STAT.PCNT1` are set every time a specified number of primary filter data sets for that filter group are transferred to memory. The primary filter data set for a group is the data for all the channels in the group. The specified number of data sets is the value in the `SINC_LEVEL0.PCNT-SINC_LEVEL1.PCNT` bits. Write 1 to clear the bits before the next data transfer to generate a trigger.

- Primary filter data buffer errors:

  A number of status bits indicate data buffer errors. The status bits `SINC_STAT.FOVF0` and `SINC_STAT.FOVF1` indicate the filter control group when there is a SINC data buffer overflow. This error occurs when a third sample is presented to the buffer before the first sample transfers to memory. The status bits `SINC_STAT.PFAB0` and `SINC_STAT.PFAB1` indicate the filter group when an error occurs while writing the data to memory.

- SINC status interrupt:

  There is a single SINC filter interrupt that can indicate secondary filter overload errors, primary filter data saturation, or primary filter data buffer overrun. There is one interrupt mask bit for each of these conditions per filter group. See Interrupt Masking for more information.

SINC Event Control

The SINC provides status and error bits through different registers to signal the core about its state and various error conditions that occur during its operation. These conditions include:

- Interrupt status related to data overload, data saturation, data FIFO fault conditions
- Error status related to SINC operations

- History status (which do not generate interrupts) related to data FIFO operations

SINC Interrupt Signals

The interrupt and trigger signals to the SINC filter module include:

- One interrupt signal, SINC_STAT, triggered by fault events, such as detected overload limits and data transfer errors. Manage interrupt generation with the masking bits in the SINC_CTL register:
  - Bits SINC_CTL.ELIM0-SINC_CTL.ELIM1 can enable (unmask) interrupt generation on overload faults when the SINC_STAT.GLIM0-SINC_STAT.GLIM1 bit is set, respectively.
  - Bits SINC_CTL.ESAT0-SINC_CTL.ESAT1 can mask interrupt generation on data saturation faults when the SINC_STAT.GSAT0-SINC_STAT.GSAT1 bit is set, respectively.
  - Bits SINC_CTL.EFOVF0-SINC_CTL.EFOVF1 can mask interrupt generation on data buffer overruns when the SINC_STAT.FOVF0-SINC_STAT.FOVF1 bit is set, respectively.

  The fault bits in the SINC_STAT register must be cleared to clear the interrupt.
- Two data count triggers, one trigger per each control group. The SINC filter module regularly uses the data count triggers to generate a software interrupt or trigger an event. First, set the SINC_CTL.EPCNT0 or SINC_CTL.EPCNT1 masking bit to enable the data count trigger. Then, the TRU must assign the data count master (SINC0_DATA0-1) to an interrupt input.
- Four overload triggers, one trigger per each channel. The SINC filter module can use overload triggers to trip the appropriate PWM block in the case of a fault. The overload trigger is always enabled, and the TRU must assign the masters (SINC0_P0_OVLD through SINC0_P4_OVLD) to the appropriate PWM trip input slave (PWMn_TRIP_TRIGn).

SINC Status and Error Signals

The status and error signals related to SINC operations are as follows.

- SINC_STAT signals:
  - The amplitude and duration limit error signals for secondary SINC filters: SINC_STAT.MAX0 through SINC_STAT.MAX3, SINC_STAT.MIN0 through SINC_STAT.MIN3, and SINC_STAT.GLIM0-SINC_STAT.GLIM1.
  - The output saturation error signals for primary SINC filters: SINC_STAT.MAX0 through SINC_STAT.MAX3, SINC_STAT.MIN0 through SINC_STAT.MIN3, and SINC_STAT.GLIM0-SINC_STAT.GLIM1.
  - The output FIFO overflow error signals for primary SINC filters: SINC_STAT.FOVF0 and SINC_STAT.FOVF1.
  - The output count error signals for primary SINC filters: SINC_STAT.PCNT0 and SINC_STAT.PCNT1.
  - The SCB fabric-related error signals for primary SINC filters: SINC_STAT.PFAB0-SINC_STAT.PFAB1.

- SINC_CLK signals:
  - The phase shift signals for SINC modulator clocks: SINC_CLK.MREQ0-SINC_CLK.MREQ1.
- SINC_HIS_STAT signals:
  - The history saved signals for secondary SINC filters: SINC_HIS_STAT.P0HISPTR through SINC_HIS_STAT.P3HISPTR, which indicate that the data history of the filter is saved in buffer registers due to a detected overload error signal.

SINC Programming Model

The pin multiplexer enables the device input and output pins and connects the signals to the SINC module. Decide the filter grouping in advance. The filter parameters are defined according to the control register group.

Follow these steps to configure the filters:

1. Define the primary and secondary filter parameters by setting the appropriate bits in the control register for each filter channel group.
2. Set the upper and lower overload limits to maximum for each channel to avoid overload trips due to the filter startup transient.
3. Define the modulator clock frequency and startup mode.
4. Enable the SINC channels and assign them to the selected group of control registers.

Set the running overload limits after the filter settles, which is (order * decimation) modulator clock cycles after startup. When the filters are running, the module transfers its data to data RAM on the dedicated DMA channels. Once configured, the control registers do not need accessing, but the status and some data buffer registers typically are read after fault events. In general, adjusting filter parameters during operation leads to unpredictable results. However, you can write to the trigger and interrupt masks, as well as to the secondary threshold levels, during operation.

The DC gain of the converter subsystem depends on the gain of the input modulator (G_M), filter order (O), and decimation rate (D). The primary filter has an output binary scalar (S) to fit data into a 16-bit range:

$$G_M = 0.625 \times \frac{D^O}{2^S}$$

SINC Programming Concepts

Using the features and event control for the SINC to their greatest potential requires an understanding of some SINC-related concepts:

Channel Configuration

Trigger Masking

Interrupt Masking

Modulator Clock

Filter Configuration

Primary Filter Parameters

Primary DMA Configuration and Data Interrupts

Secondary Filter Parameters

Overload Detection

Channel Configuration

The control bits, SINC_CTL.EN0 through SINC_CTL.EN3, configure SINC module channels. These control bits enable or disable the selected SINC filter channel and assign the channel to one of the two control register groups. The selected control register group also determines the filter clock source.

Trigger Masking

The SINC module has two data count triggers, one trigger per each group. The module can use the data count triggers to generate a software interrupt regularly or trigger an event. First, set SINC_CTL.EPCNT0 and SINC_CTL.EPCNT1 masking bit to enable the data count trigger. Then, the TRU must assign the data count master (SINC_DATn) to an interrupt input.

There are also four overload triggers, one trigger per each channel. The SINC module can use overload triggers to trip the appropriate PWM block when there is a fault. The overload trigger is always enabled, and the TRU must assign the masters (SINC0_Pn_OVLD) to the appropriate PWM trip input slave (PWMn_TRIP_TRIGn).

Interrupt Masking

The SINC filter can generate a SINC_STAT interrupt signal when triggered by fault events, such as detected overload limits or data transfer errors.

Enable (unmask) interrupt generation with the SINC_CTL register bits:

- Bits SINC_CTL.ELIM0-SINC_CTL.ELIM1 can enable interrupt generation on overload faults when the SINC_STAT.GLIM0-SINC_STAT.GLIM1 bit is set, respectively.
- Bits SINC_CTL.ESAT0-SINC_CTL.ESAT1 can enable interrupt generation on data saturation faults when the SINC_STAT.GSAT0-SINC_STAT.GSAT1 bit is set, respectively.
- Bits SINC_CTL.EFOVF0-SINC_CTL.EFOVF1 can enable interrupt generation on data buffer overruns when the SINC_STAT.FOVF0-SINC_STAT.FOVF1 bit is set, respectively.

The fault bits in the SINC_STAT register must be cleared to clear the interrupt.

Modulator Clock

The SINC filter has two modulator clock sources. Out of the two modulator clock sources, only the modulator clock for GROUP 0 is available on the GPIO port. Each clock source can be set with an output frequency in the range of 1-20 MHz. The SINC module uses bits in the SINC_CLK register to control the modulator clock output, frequency, and phase. Assign the modulator clocks to the SINC filter channels according to their control group assignments. The SINC module uses the SINC_CLK.MCEN0-SINC_CLK.MCEN1 bit fields to enable the modulator clocks and control the startup behavior of the clock. Start the clock immediately or enable the clock on the first rising edge of an external trigger connected to the SINC0_SYNCn input of the module. This action synchronizes the modulator clock with a PWM waveform source by routing a PWMn_SYNC master to a SINC0_SYNC0 or SINC0_SYNC1 slave using the TRU.

The target frequency is in the range and derived from SYSCLK using an integer divisor in the SINC_CLK.MDIV0 or SINC_CLK.MDIV1 bits. Write to the SINC_CLK.MREQ0 or SINC_CLK.MREQ1 bit to adjust the phase of the clock. This adjustment lengthens the next clock period by the number of SYSCLK periods stored in the respective SINC_CLK.MADJ0 or SINC_CLK.MADJ1 bit field. The SINC_CLK.MREQ0 or SINC_CLK.MREQ1 bit is cleared automatically once the adjustment is complete.

Filter Configuration

Configure the primary and secondary filter parameters, according to the group number, by setting the appropriate bits in the SINC_RATE0-SINC_RATE1, SINC_LEVEL0-SINC_LEVEL1, and SINC_BIAS0-SINC_BIAS1 control registers. Configure the DMA transfers by setting the appropriate bits in the SINC_PHEAD0-SINC_PHEAD1 and SINC_PTAIL0-SINC_PTAIL1 registers. Set the maximum and minimum levels for overload detection in the four limit registers, SINC_LIMIT0-SINC_LIMIT3. Set the overload filtering parameters in the SINC_LEVEL0-SINC_LEVEL1 registers.

Primary Filter Parameters

Set the primary filter to the 3rd or 4th order by the SINC_LEVEL0.PORD or SINC_LEVEL1.PORD bit assigned to the channel. Set the decimation rate for the primary filter using the SINC_RATE0.PDEC or SINC_RATE1.PDEC bits assigned to the channel. Valid decimation rates are in the range 4–256. Set the phase of the primary filter output relative to the number of modulator clocks after enabling the filter using the SINC_RATE0.PADJ or SINC_RATE1.PADJ bits assigned to the channel. Valid PADJ values are in the range 0 to PDEC - 1.

The raw filter output is a 32-bit wide integer, has an offset added, and is scaled to a 16-bit number before transfer to memory. Store the 32-bit two's compliment offset value in the SINC_BIAS0 or SINC_BIAS1 register of the channel. Set the binary scale factor by a mantissa in the range 4–32 stored in the SINC_LEVEL0.PSCALE or SINC_LEVEL1.PSCALE bits. The output is a valid 16-bit signed number. If the number is outside of the valid range, the output is saturated to 0x8000 or 0x7FFF, while the SINC_STAT.PSAT0 or SINC_STAT.GSAT1 fault bit (according to the channel group) is set.

Primary DMA Configuration and Data Interrupts

Transfer the primary SINC filter outputs to a circular buffer in data memory using DMA. There are separate DMA streams for each filter channel group. The output from the primary filter is interleaved with outputs from other primary filters in the same group. The interleaving order is from the lowest to the highest numbered filter.

The SINC module stores the circular buffer head address in the SINC_PHEAD0 or SINC_PHEAD1 register of the channel. It stores the tail address in the SINC_PTAIL0 or SINC_PTAIL1 register of the channel. The data address wraps around to the head address after the tail address is reached. The head and tail addresses must be 16-bit aligned and can be set to the same address. The SINC_PPTR0 or SINC_PPTR1 register of the channel is a read-only register that contains the address of the most recent primary SINC filter data. If there is an overflow condition in the SINC filter output data FIFO, due to a delay DMA transfer, the SINC_STAT.FOVF0, or SINC_STAT.FOVF1 fault bit (according to the channel group) is set.

A SINC data trigger can be generated after a user-specified number of primary filter outputs (data transfers) completes. Specify the data count value by the SINC_LEVEL0.PCNT or SINC_LEVEL1.PCNT bits assigned to the channel, and the trigger is generated every PCNT + 1 data transfers.

The ***SINC Data Buffer Organization*** figure shows the SINC data buffer organization. In the figure, SINC_OUT_X_M[n] is the data for the n th most recent sample in the M th channel in the filter group X, and n = 0 is the most recent data.

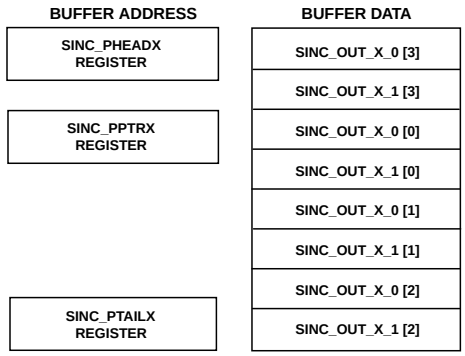

Figure 29-5: SINC Data Buffer Organization

Secondary Filter Parameters

Set the secondary filter to the 3rd or 4th order by the SINC_LEVEL0.SORD or SINC_LEVEL1.SORD bit assigned to the channel. Set the decimation rate for the secondary filter using the SINC_RATE0.SDEC or SINC_RATE1.SDEC bits assigned to the channel. The secondary filter outputs are limited to 16-bit values. Limit the decimation rate according to the filter order:

- Valid decimation rates are in the range 4–40 for the 3rd order filters
- Valid decimation rates are in the range of 4–16 for the 4th order filters

Set the phase of the primary filter output relative to the number of modulator clocks after using the SINC_RATE0.PADJ or SINC_RATE1.SADJ bits to enable the filter. Valid PADJ values are in the range 0 to SDEC - 1.

Overload Detection

The function of the secondary SINC filter is to detect AC current overload conditions and set up the upper and lower limit detection thresholds. There are event count filters on the overload detector outputs to reject short-term transients, if desired. Define the overload thresholds in four 32-bit registers SINC_LIMIT0-SINC_LIMIT3, according to the channel number. Each register contains the 16-bit LMAX and LMIN overload threshold values. The SINC filter module detects an overload condition when the secondary filter output exceeds the threshold for a minimum number of counts (LCNT) within the detection window (LWIN). Set the LCNT and LWIN count values in the SINC_LEVEL0 or SINC_LEVEL1 register assigned to the channel. When the SINC filter module detects an overload condition, the appropriate SINC0_Px_OVLD trigger is generated, and the SINC_STAT.GLIM0 or SINC_STAT.GLIM1 fault bit is set.

The SINC filter module saves the eight most recent data samples for the secondary filter in a local circular buffer to support diagnostics after a fault is triggered. Since 16-bit data is saved, only four buffer registers are required per channel. For example, SINC_P1SEC_HIST0-3 store the eight most recent 16-bit secondary filter outputs from channel 1. The SINC_HIS_STAT register contains four pointers (SINC_HIS_STAT.P0HISPTR through SINC_HIS_STAT.P3HISPTR) to the buffer location of the most recent secondary current samples, per channel.

ADSP-CM40x SINC Register Descriptions

SINC (SINC) contains the following registers.

Table 29-6: ADSP-CM40x SINC Register List

Name	Description
SINC_BIAS0	Bias for Group 0 Register
SINC_BIAS1	Bias for Group 1 Register
SINC_CLK	Clock Control Register
SINC_CTL	Control Register
SINC_HIS_STAT	History Status Register
SINC_LEVEL0	Level Control for Group 0 Register
SINC_LEVEL1	Level Control for Group 1 Register
SINC_LIMIT0	(Amplitude) Limits for Secondary Filter 0 Register
SINC_LIMIT1	(Amplitude) Limits for Secondary Filter 1 Register
SINC_LIMIT2	(Amplitude) Limits for Secondary Filter 2 Register
SINC_LIMIT3	(Amplitude) Limits for Secondary Filter 3 Register

Table 29-6: ADSP-CM40x SINC Register List (Continued)

Name	Description
SINC_P0SEC_HIST[n]	Pair 0 Secondary (Filter) History n Register
SINC_P1SEC_HIST[n]	Pair 1 Secondary (Filter) History n Register
SINC_P2SEC_HIST[n]	Pair 2 Secondary (Filter) History n Register
SINC_P3SEC_HIST[n]	Pair 3 Secondary (Filter) History n Register
SINC_PHEAD0	Primary (Filters) Head for Group 0 Register
SINC_PHEAD1	Primary (Filters) Head for Group 1 Register
SINC_PPTR0	Primary (Filters) Pointer for Group 0 Register
SINC_PPTR1	Primary (Filters) Pointer for Group 1 Register
SINC_PTAIL0	Primary (Filters) Tail for Group 0 Register
SINC_PTAIL1	Primary (Filters) Tail for Group 1 Register
SINC_RATE0	Rate Control for Group 0 Register
SINC_RATE1	Rate Control for Group 1 Register
SINC_STAT	Status Register

Bias for Group 0 Register

The SINC_BIAS0 register controls an output bias added to primary SINC filters of group 0.

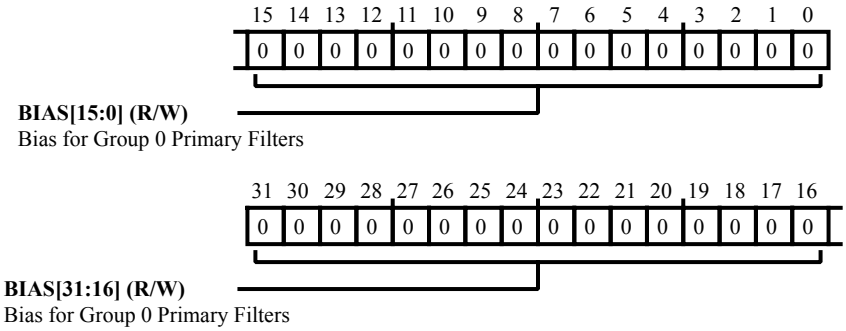

Figure 29-6: SINC_BIAS0 Register Diagram

Table 29-7: SINC_BIAS0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	BIAS	Bias for Group 0 Primary Filters. The SINC_BIAS0.BIAS bits specify a bias for the primary SINC filters output. The bias is added to the output prior to saturation and DMA memory transfer. The valid value is represented in two's complement format; thus, must be programmed to be equal to -(d ^ o) / 2, where d = SINC_RATE0.PDEC and o = SINC_LEVEL0.PORD.

Bias for Group 1 Register

The SINC_BIAS1 register controls an output bias added to primary SINC filters of group 1.

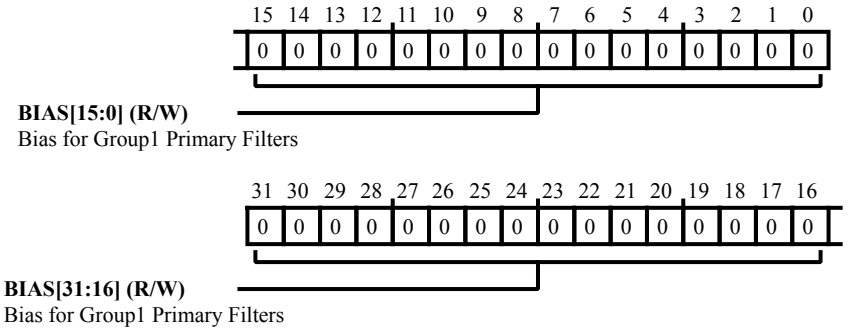

Figure 29-7: SINC_BIAS1 Register Diagram

Table 29-8: SINC_BIAS1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	BIAS	Bias for Group1 Primary Filters. The SINC_BIAS1.BIAS bits specify a bias for the primary SINC filters output. The bias is added to the output prior to saturation and DMA memory transfer. The valid value is represented in two's complement format; thus, must be programmed to be equal to -(d ^ o) / 2, where where d = SINC_RATE1.PDEC and o = SINC_LEVEL1.PORD.

Clock Control Register

The SINC_CLK register generates and enables two SINC modulator clocks. The register also controls each clock's output, frequency, phase, and start-up behavior.

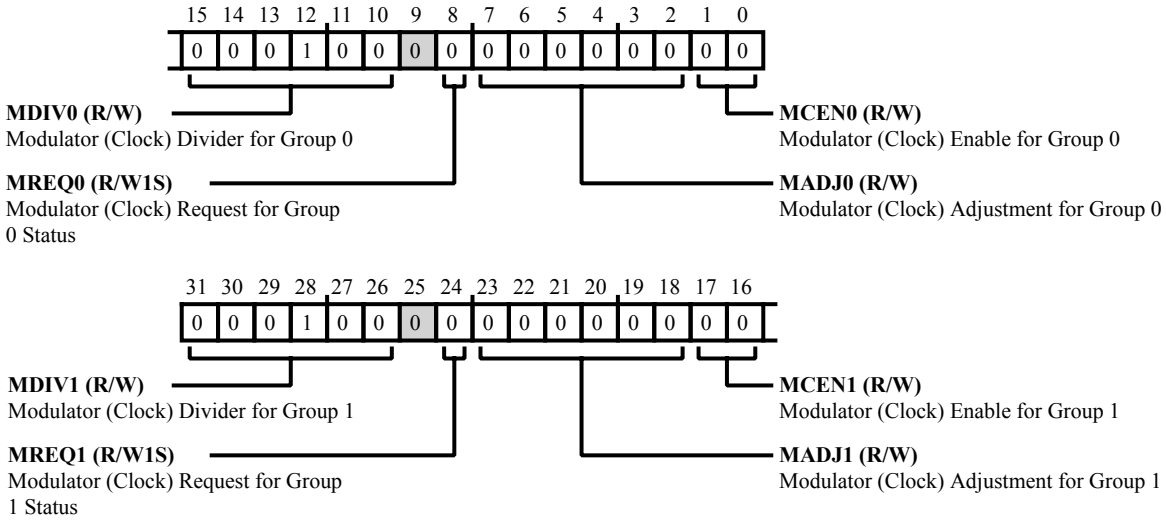

Figure 29-8: SINC_CLK Register Diagram

Table 29-9: SINC_CLK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:26 (R/W)	MDIV1	Modulator (Clock) Divider for Group 1. The SINC_CLK.MDIV1 bits provide the SYSCLK divider to generate the modulator clock for group 1. The valid value is between 1 and 63.	
24 (R/W1S)	MREQ1	Modulator (Clock) Request for Group 1 Status. The SINC_CLK.MREQ1 bit indicates status for a phase shift request of the modulator clock for group 1. If the bits state is changed from clear (=0) to set (=1), the following modulator clock 1 period is lengthened by the number of SYSCLK periods specified by the SINC_CLK.MADJ1 bits. Any writes to this bit while the bit is set are ignored. The bit is cleared by hardware (and only by hardware) once a requested modulator clock adjustment is complete.	
		0	Inactive
		1	Active
23:18 (R/W)	MADJ1	Modulator (Clock) Adjustment for Group 1. The SINC_CLK.MADJ1 bits provide the adjustment value for the modulator clock of group 1. The valid value is between 1 and 63 when SINC_CLK.MREQ1 is set (=1). A write to this bit field effects only an active modulator clock adjustment. See the SINC_CLK.MREQ1 bit filed description.	

Table 29-9: SINC_CLK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17:16 (R/W)	MCEN1	Modulator (Clock) Enable for Group 1. The SINC_CLK.MCEN1 bits enable/disable the modulator clock for group 1 and control the clocks start-up behavior. Commence the clock immediately upon making it enabled, or enable and commence upon the next rising edge of PWMSYNC (PWM synchronizing output clock).	
		0	Disable
		1	Reserved
		2	Enable and Commence
		3	Enable and Commence on Next Rising Edge
15:10 (R/W)	MDIV0	Modulator (Clock) Divider for Group 0. The SINC_CLK.MDIV0 bits provide the SYSCLK divider to generate the modulator clock for group 0. The valid value is between 1 and 63.	
8 (R/W1S)	MREQ0	Modulator (Clock) Request for Group 0 Status. The SINC_CLK.MREQ0 bit indicates status for a phase shift request of the modulator clock for group 0. If the bits state is changed from clear (=0) to set (=1), the following modulator clock 0 period is lengthened by the number of SYSCLK periods specified by the SINC_CLK.MADJ0 bits. Any writes to this bit while the bit is set are ignored. The bit is cleared by hardware (and only by hardware) once a requested modulator clock adjustment is complete.	
		0	Inactive
		1	Active
7:2 (R/W)	MADJ0	Modulator (Clock) Adjustment for Group 0. The SINC_CLK.MADJ0 bits provide the adjustment value for the modulator clock of group 0. The valid value is between 1 and 63 when SINC_CLK.MREQ1 is set (=1). A write to this bit field effects only an active modulator clock adjustment. See the SINC_CLK.MREQ1 bit filed description.	
1:0 (R/W)	MCEN0	Modulator (Clock) Enable for Group 0. The SINC_CLK.MCEN0 bits enable/disable the modulator clock for group 0 and control the clocks start-up behavior. Commence the clock immediately upon making it enabled, or enable and commence upon the next rising edge of PWMSYNC (PWM synchronizing output clock).	
		0	Disable
		1	Reserved
		2	Enable and Commence
		3	Enable and Commence on Next Rising Edge

Control Register

The SINC_CTL register masks (disables) and unmasks (enables) SINC high-level interrupt signals triggered by fault events. The register also enables and assigns SINC filter pairs to one of two control groups.

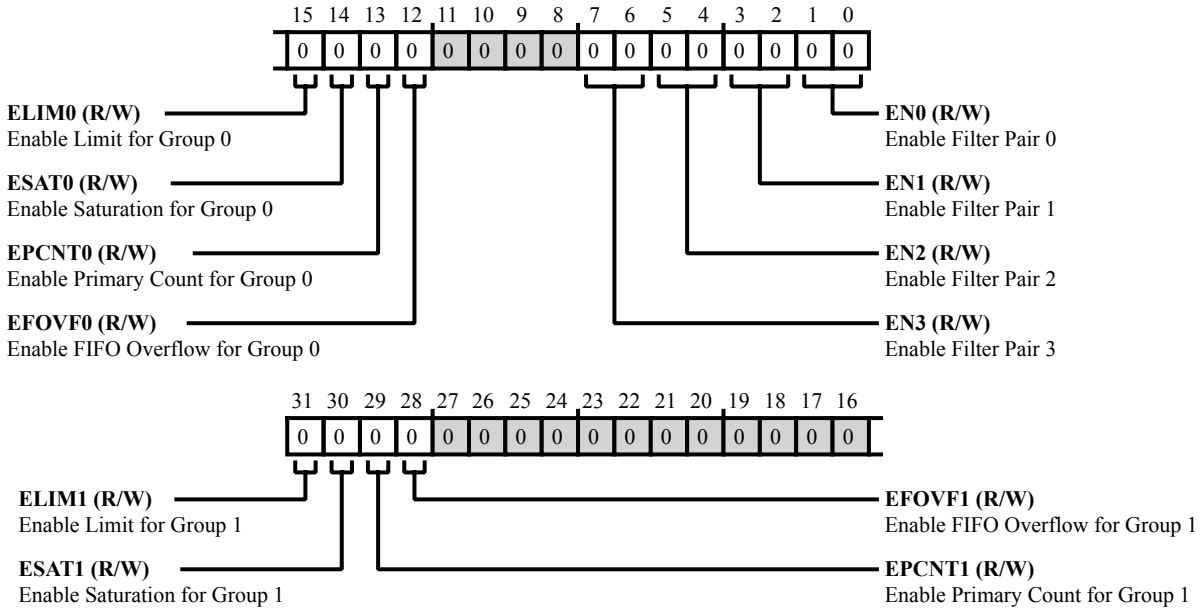

Figure 29-9: SINC_CTL Register Diagram

Table 29-10: SINC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	ELIM1	Enable Limit for Group 1. The SINC_CTL.ELIM1 bit enables (unmasks) the SINC_STAT interrupt on overload conditions if this bit and status bit SINC_STAT.GLIM1 are set (=1).	
		0	Disable
		1	Enable
30 (R/W)	ESAT1	Enable Saturation for Group 1. The SINC_CTL.ESAT1 bit enables (unmasks) the SINC_STAT interrupt on output saturation conditions if this bit and bit SINC_STAT.GSAT1 are set (=1).	
		0	Disable
		1	Enable

Table 29-10: SINC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/W)	EPCNT1	Enable Primary Count for Group 1. The SINC_CTL.EPCNT1 bit enables a trigger event on each SINC_DATA1 request if this bit and status bit SINC_STAT.PCNT1 are set (=1).	
		0	Disable
		1	Enable
28 (R/W)	EFOVF1	Enable FIFO Overflow for Group 1. The SINC_CTL.EFOVF1 bit enables (unmasks) the SINC_STAT interrupt on data FIFO overflow conditions if this bit and status bit SINC_STAT.FOVF1 are set (=1). The SINC_STAT.FOVF1 bit is set (=1) when the group 1 output data FIFO overflows due to delayed SCB fabric ready response.	
		0	Disable
		1	Enable
15 (R/W)	ELIM0	Enable Limit for Group 0. The SINC_CTL.ELIM0 bit enables (unmasks) the SINC_STAT interrupt on overload conditions if this bit and status bit SINC_STAT.GLIM0 are set (=1).	
		0	Disable
		1	Enable
14 (R/W)	ESAT0	Enable Saturation for Group 0. The SINC_CTL.ESAT0 bit enables (unmasks) the SINC_STAT interrupt on output saturation conditions if this bit and status bit SINC_STAT.GSAT0 are set (=1).	
		0	Disable
		1	Enable
13 (R/W)	EPCNT0	Enable Primary Count for Group 0. The SINC_CTL.EPCNT0 bit enables a trigger event on each SINC_DATA0 request if this bit and status bit SINC_STAT.PCNT0 are set (=1).	
		0	Disable
		1	Enable
12 (R/W)	EFOVF0	Enable FIFO Overflow for Group 0. The SINC_CTL.EFOVF0 bit enables (unmasks) the SINC_STAT interrupt on data FIFO overflow conditions if this bit and status bit SINC_STAT.FOVF0 are set (=1). The SINC_STAT.FOVF0 bit is set (=1) when the group 0 output data FIFO overflows due to delayed SCB fabric ready response.	
		0	Disable
		1	Enable

Table 29-10: SINC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/W)	EN3	Enable Filter Pair 3. The SINC_CTL.EN3 bits enable/disable and assign SINC filter pair 3 to the control group.	
		0	Disable
		1	Reserved
		2	Enable and Assign to Group 0
		3	Reserved
5:4 (R/W)	EN2	Enable Filter Pair 2. The SINC_CTL.EN2 bits enable/disable and assign SINC filter pair 2 to the control group.	
		0	Disable
		1	Reserved
		2	Enable and Assign to Group 0
		3	Reserved
3:2 (R/W)	EN1	Enable Filter Pair 1. The SINC_CTL.EN1 bits enable/disable and assign SINC filter pair 1 to the control group.	
		0	Disable
		1	Reserved
		2	Enable and Assign to Group 0
		3	Reserved
1:0 (R/W)	EN0	Enable Filter Pair 0. The SINC_CTL.EN0 bits enable/disable and assign SINC filter pair 0 to the control group.	
		0	Disable
		1	Reserved
		2	Enable and Assign to Group 0
		3	Reserved

History Status Register

The SINC_HIS_STAT provides status for data histories of secondary SINC filters, in the corresponding history buffer registers. The SINC history buffer registers save the most recent filter samples once an overload fault signal is detected.

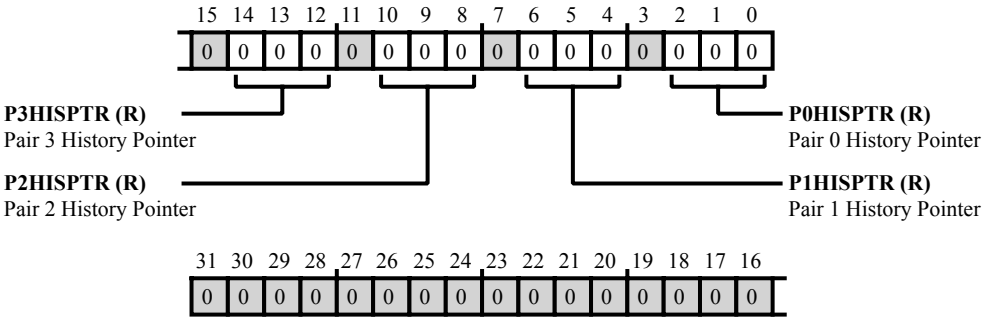

Figure 29-10: SINC_HIS_STAT Register Diagram

Table 29-11: SINC_HIS_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
14:12 (R/NW)	P3HISPTR	Pair 3 History Pointer. The SINC_HIS_STAT.P3HISPTR bits indicate the position for the most recent data sample of secondary SINC filter 3 in the corresponding SINC_P3SEC_HIST[n] register block.	
		0	History Register 3, MS
		1	History Register 0, LS
		2	History Register 0, MS
		3	History Register 1, LS
		4	History Register 1, MS
		5	History Register 2, LS
		6	History Register 2, MS
		7	History Register 3, LS
10:8 (R/NW)	P2HISPTR	Pair 2 History Pointer. The SINC_HIS_STAT.P2HISPTR bits indicate the position for the most recent data sample of secondary SINC filter 2 in the corresponding SINC_P2SEC_HIST[n] register block.	
		0	History Register 3, MS
		1	History Register 0, LS
		2	History Register 0, MS

Table 29-11: SINC_HIS_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		3	History Register 1, LS
		4	History Register 1, MS
		5	History Register 2, LS
		6	History Register 2, MS
		7	History Register 3, LS
6:4 (R/NW)	P1HISPTR	Pair 1 History Pointer. The SINC_HIS_STAT.P1HISPTR bits indicate the position for the most recent data sample of secondary SINC filter 1 in the corresponding SINC_P1SEC_HIST[n] register block.	
		0	History Register 3, MS
		1	History Register, LS
		2	History Register 0, MS
		3	History Register 1, LS
		4	History Register 1, MS
		5	History Register 2, LS
		6	History Register 2, MS
		7	History Register 3, LS
2:0 (R/NW)	P0HISPTR	Pair 0 History Pointer. The SINC_HIS_STAT.P0HISPTR bits indicate the position for the most recent data sample of secondary SINC filter 0 in the corresponding SINC_P0SEC_HIST[n] register block.	
		0	History Register 3, MS
		1	History Register 0, LS
		2	History Register 0, MS
		3	History Register 1, LS
		4	History Register 1, MS
		5	History Register 2, LS
		6	History Register 2, MS
		7	History Register 3, LS

Level Control for Group 0 Register

The SINC_LEVEL0 register controls output scaling and count, excursion limit and window, as well as orders for primary and secondary SINC filters assigned to group 0.

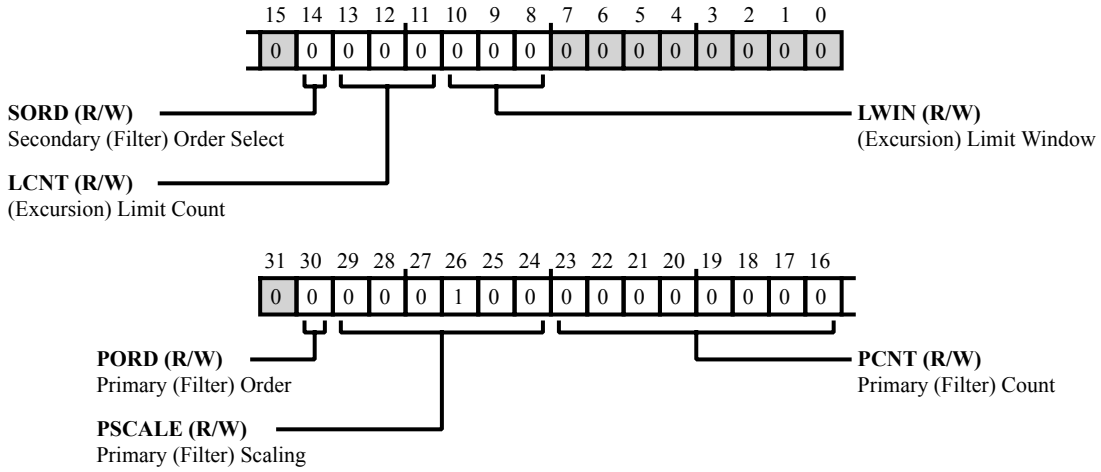

Figure 29-11: SINC_LEVEL0 Register Diagram

Table 29-12: SINC_LEVEL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	PORD	Primary (Filter) Order. The SINC_LEVEL0.PORD bit determines the order for group 1 primary filters.	
		0	Third Order
		1	Fourth Order
29:24 (R/W)	PSCALE	Primary (Filter) Scaling. The SINC_LEVEL0.PSCALE bits specify the scaling applied to the output of group 0 primary filters, prior to DMA transfer to memory. The valid value is between 4 to 32. The SINC integrator, decimator, and bias adjustment produce an integer value up to 32 bits wide. The range of a full-scale signal of a bit stream filtered by a primary SINC filter is approximately (BIAS +- ((0.625 * SINC_RATE0.PDEC) ^ order)). The value requires about (ln2(SINC_RATE0.PDEC) * order) bits of precision (where 'order' is 3 or 4, as specified by the SINC_LEVEL0.PORD bit. This bit field specifies the bit position of the intermediate value, which is transferred on the MSB of 16-bit DMA sample. Thus, the intermediate value is right-shifted by (SINC_LEVEL0.PSCALE - 16) bits if SINC_LEVEL0.PSCALE >= 16, or left-shifted by (16 - SINC_LEVEL0.PSCALE) bits if SINC_LEVEL0.PSCALE < 16. If SINC_LEVEL0.PSCALE >= 16, thus selecting a right shift, the shifted value is rounded up (as if 0.5 * LSB is added) before truncation. Rounding is not necessary for a left shift. If the scaled and rounded value exceeds the range of a signed 16-bit number, the	

Table 29-12: SINC_LEVEL0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		sample is saturated (to 0x8000 or 0x7FFF), and the corresponding saturation status bit (SINC_STAT.PSAT3, SINC_STAT.PSAT2, SINC_STAT.PSAT1, or SINC_STAT.PSAT0 is set.
23:16 (R/W)	PCNT	Primary (Filter) Count. The SINC_LEVEL0.PCNT bits specify the modulo number of outputs for group 0 primary filters. The number must be one less than a desired modulo. Each time the number of outputs specified by this bit filed is transferred, the SINC_STAT.PCNT0 status bit is set (=1). When the SINC_STAT.PCNT0 bit is set (unless masked), it causes a TRU trigger. For example: 8'h00 written to the SINC_LEVEL0.PCNT bit field sets bit SINC_STAT.PCNT0 to 1 after every primary SINC filter output is transferred. 8'hFF written to the SINC_LEVEL0.PCNT bit field sets bit SINC_STAT.PCNT0 to 1 after every 256 primary SINC filter outputs transferred.
14 (R/W)	SORD	Secondary (Filter) Order Select. The SINC_LEVEL0.SORD bit determines the order for group 0 secondary filters. 0 – Third Order 1 – Fourth Order
13:11 (R/W)	LCNT	(Excursion) Limit Count. The SINC_LEVEL0.LCNT bits specify the number (count) of output excursions beyond the amplitude specified for group 0 secondary filters. The number of excursions greater than specified by registers SINC_LIMIT3, SINC_LIMIT2, SINC_LIMIT2, and SINC_LIMIT0 is perceived as an overload and sets (=1) a corresponding MAX or MIN bit in the SINC_STAT register. The valid count is between 1 to 8. If the count is greater than SINC_LEVEL0.LWIN, the bit fields behavior is as it is equal to SINC_LEVEL0.LWIN. See SINC_LEVEL0.LWIN for details. The valid count must be one less than a desired count: =000 require one excursion above the amplitude limit =111 require eight excursions above the amplitude limit.
10:8 (R/W)	LWIN	(Excursion) Limit Window. The SINC_LEVEL0.LWIN bits specify the window size for excursion checking for group 0 secondary filters. The window size is the number of the most recent outputs to be included in a measurement specified by the SINC_LEVEL0.LCNT bits. The valid value must be one less than a desired count (1 to 8), meaning the valid value is 0 to 7.

Level Control for Group 1 Register

The SINC_LEVEL1 register controls output scaling and count, excursion limit and window, as well as orders for primary and secondary SINC filters assigned to group 1.

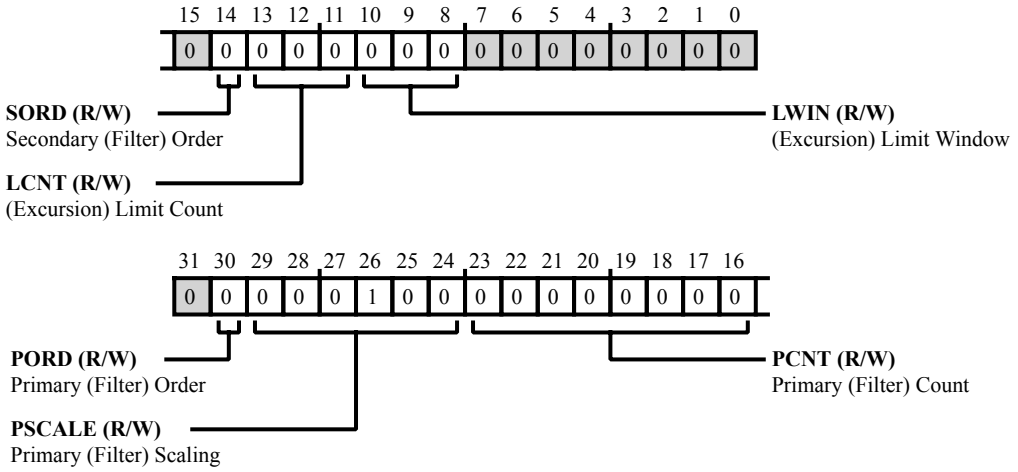

Figure 29-12: SINC_LEVEL1 Register Diagram

Table 29-13: SINC_LEVEL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	PORD	Primary (Filter) Order. The SINC_LEVEL1.PORD bits determines the order for group 1 primary filters.	
		0	Third Order
		1	Fourth Order
29:24 (R/W)	PSCALE	Primary (Filter) Scaling. The SINC_LEVEL1.PSCALE bits specify the scaling applied to the output of group 1 primary filters, prior to DMA transfer to memory. The valid value is between 4 to 32. The SINC integrator, decimator, and bias adjustment produce an integer value up to 32 bits wide. The range of a full-scale signal of a bit stream filtered by a primary SINC filter is approximately (BIAS +- ((0.625 * SINC_RATE1.PDEC) ^ order)). The value requires about (ln2(SINC_RATE1.PDEC) * order) bits of precision (where 'order' is 3 or 4, as specified by the SINC_LEVEL1.PORD bit. This bit field specifies the bit position of the intermediate value, which is transferred on the MSB of 16-bit DMA sample. Thus, the intermediate value is right-shifted by (SINC_LEVEL1.PSCALE - 16) bits if SINC_LEVEL1.PSCALE >= 16, or left-shifted by (16 - SINC_LEVEL1.PSCALE) bits if SINC_LEVEL1.PSCALE < 16. If SINC_LEVEL1.PSCALE >= 16, thus selecting a right shift, the shifted value is rounded up (as if 0.5 * LSB is added) before truncation. Rounding is not necessary for a left shift. If the scaled and rounded value exceeds the range of a signed 16-bit number, the	

Table 29-13: SINC_LEVEL1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		sample is saturated (to 0x8000 or 0x7FFF), and the corresponding saturation status bit (SINC_STAT.PSAT3, SINC_STAT.PSAT2, SINC_STAT.PSAT1, or SINC_STAT.PSAT0 is set.
23:16 (R/W)	PCNT	Primary (Filter) Count. The SINC_LEVEL1.PCNT bits specify the modulo number of outputs for group 1 primary filters. The number must be one less than a desired modulo. Each time the number of outputs specified by this bit filed is transferred, the SINC_STAT.PCNT1 status bit is set (=1). When the SINC_STAT.PCNT1 bit is set (unless masked), it causes a TRU trigger. For example: 8'h00 written to the SINC_LEVEL1.PCNT bit field sets bit SINC_STAT.PCNT1 to 1 after every primary SINC filter output is transferred. 8'hFF written to the SINC_LEVEL1.PCNT bit field sets bit SINC_STAT.PCNT1 to 1 after every 256 primary SINC filter outputs transferred.
14 (R/W)	SORD	Secondary (Filter) Order. The SINC_LEVEL1.SORD bit determines the order for group 1 secondary filters. The SINC_LEVEL1.SORD bit determines the order for group 1 secondary filters. 0 – Third Order 1 – Fourth Order
13:11 (R/W)	LCNT	(Excursion) Limit Count. The SINC_LEVEL1.LCNT bits specify the number (count) of output excursions beyond the amplitude specified for group 1 secondary filters. The number of excursions greater than specified by registers SINC_LIMIT3, SINC_LIMIT2, SINC_LIMIT2, and SINC_LIMIT0 is perceived as an overload and sets (=1) a corresponding MAX or MIN bit in the SINC_STAT register. The valid count is between 1 to 8. If the count is greater than SINC_LEVEL1.LWIN, the bit fields behavior is as it is equal to SINC_LEVEL1.LWIN. See SINC_LEVEL1.LWIN for details. The valid count must be one less than a desired count: =000 require one excursion above the amplitude limit =111 require eight excursions above the amplitude limit.
10:8 (R/W)	LWIN	(Excursion) Limit Window. The SINC_LEVEL1.LWIN bits specify the window size for excursion checking for group 1 secondary filters. The window size is the number of the most recent outputs to be included in a measurement specified by the SINC_LEVEL1.LCNT bits. The valid value must be one less than a desired count (1 to 8), meaning the valid value is 0 to 7.

(Amplitude) Limits for Secondary Filter 0 Register

The SINC_LIMIT0 register controls amplitude limits for a secondary filter of SINC pair 0.

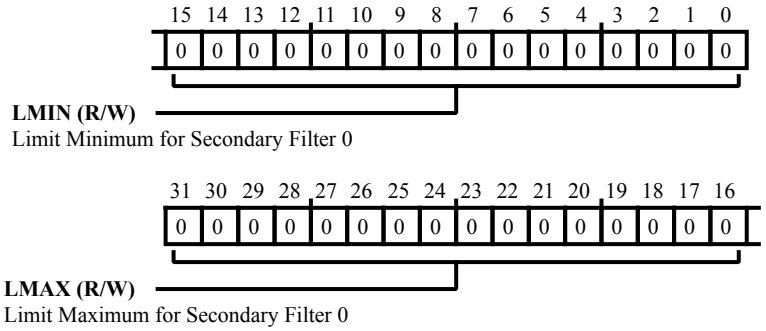

Figure 29-13: SINC_LIMIT0 Register Diagram

Table 29-14: SINC_LIMIT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	LMAX	Limit Maximum for Secondary Filter 0. The SINC_LIMIT0.LMAX bits specify the output signal conditions for the secondary SINC filter 0. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated maximum limit warning bit in register SINC_STAT.
15:0 (R/W)	LMIN	Limit Minimum for Secondary Filter 0. The SINC_LIMIT0.LMIN bits specify the output signal conditions for the secondary SINC filter 0. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated minimum limit warning bit in register SINC_STAT.

(Amplitude) Limits for Secondary Filter 1 Register

The SINC_LIMIT1 register controls amplitude limits for a secondary filter of SINC pair 1.

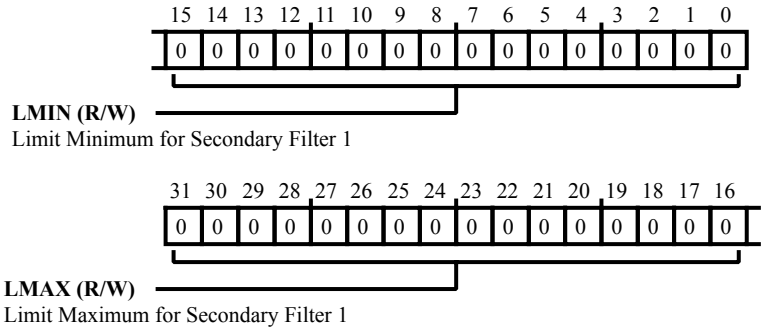

Figure 29-14: SINC_LIMIT1 Register Diagram

Table 29-15: SINC_LIMIT1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	LMAX	Limit Maximum for Secondary Filter 1. The SINC_LIMIT1.LMAX bits specify the output signal conditions for the secondary SINC filter 1. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated maximum limit warning bit in register SINC_STAT.
15:0 (R/W)	LMIN	Limit Minimum for Secondary Filter 1. The SINC_LIMIT1.LMIN bits specify the output signal conditions for the secondary SINC filter 1. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated minimum limit warning bit in register SINC_STAT.

(Amplitude) Limits for Secondary Filter 2 Register

The SINC_LIMIT2 register controls amplitude limits for a secondary filter of SINC pair 2.

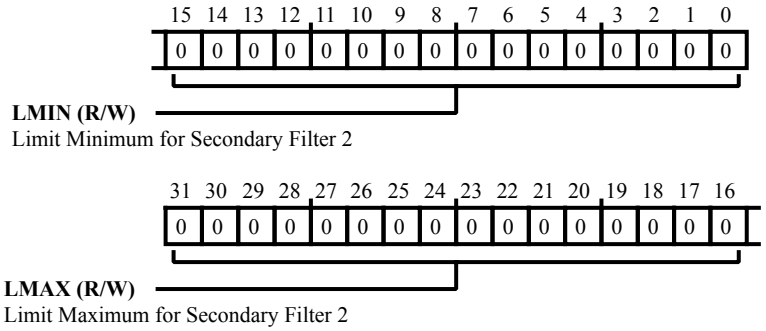

Figure 29-15: SINC_LIMIT2 Register Diagram

Table 29-16: SINC_LIMIT2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	LMAX	Limit Maximum for Secondary Filter 2. The SINC_LIMIT2.LMAX bits specify the output signal conditions for the secondary SINC filter 2. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated maximum limit warning bit in register SINC_STAT.
15:0 (R/W)	LMIN	Limit Minimum for Secondary Filter 2. The SINC_LIMIT2.LMIN bits specify the output signal conditions for the secondary SINC filter 2. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated minimum limit warning bit in register SINC_STAT.

(Amplitude) Limits for Secondary Filter 3 Register

The SINC_LIMIT3 register controls amplitude limits for a secondary filter of SINC pair 3.

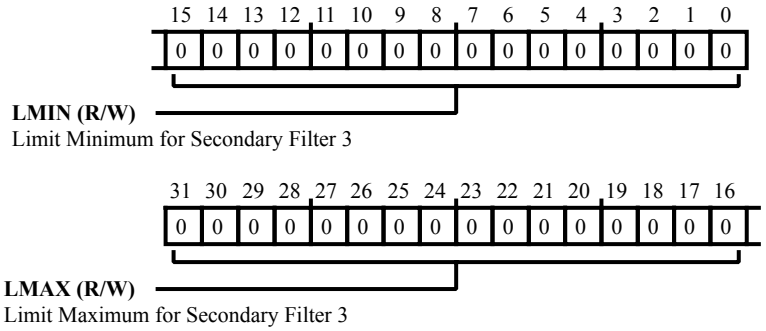

Figure 29-16: SINC_LIMIT3 Register Diagram

Table 29-17: SINC_LIMIT3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	LMAX	Limit Maximum for Secondary Filter 3. The SINC_LIMIT3.LMAX bits specify the output signal conditions for the secondary SINC filter 3. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated maximum limit warning bit in register SINC_STAT.
15:0 (R/W)	LMIN	Limit Minimum for Secondary Filter 3. The SINC_LIMIT3.LMIN bits specify the output signal conditions for the secondary SINC filter 3. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated minimum limit warning bit in register SINC_STAT.

Pair 0 Secondary (Filter) History n Register

The SINC_P0SEC_HIST[n] read-only register provides the eight most recent samples produced by secondary SINC filter 0. The 16-bit samples are stored in the 32-bit register in circular manner, starting with the low-order field of the first SINC_P0SEC_HIST[n] register. The stored values, one compared to the limit, count, and window settings, set the SINC_STAT.MAX0 and SINC_STAT.MIN0 bits.

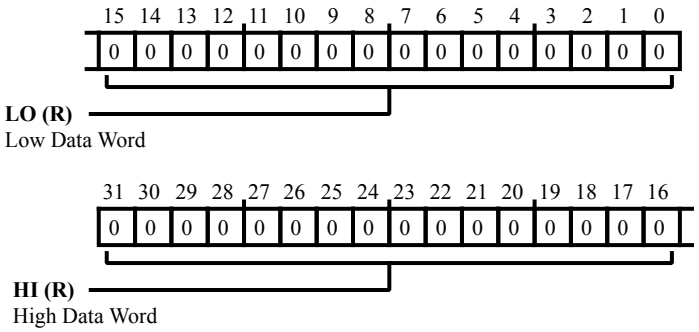

Figure 29-17: SINC_P0SEC_HIST[n] Register Diagram

Table 29-18: SINC_P0SEC_HIST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	HI	High Data Word. The SINC_P0SEC_HIST[n].HI bits provide the 16-bit sample in the most significant half of the 32- bit register.
15:0 (R/NW)	LO	Low Data Word. The SINC_P0SEC_HIST[n].LO bits provide the 16-bit sample in the least significant half of the 32- bit register.

Pair 1 Secondary (Filter) History n Register

The SINC_P1SEC_HIST[n] read-only register provides the eight most recent samples produced by secondary SINC filter 1. The 16-bit samples are stored in the 32-bit register in circular manner, starting with the low-order field of the first SINC_P1SEC_HIST[n] register. The stored values, compared to the limit, count, and window settings, set the SINC_STAT.MAX1 and SINC_STAT.MIN1 bits.

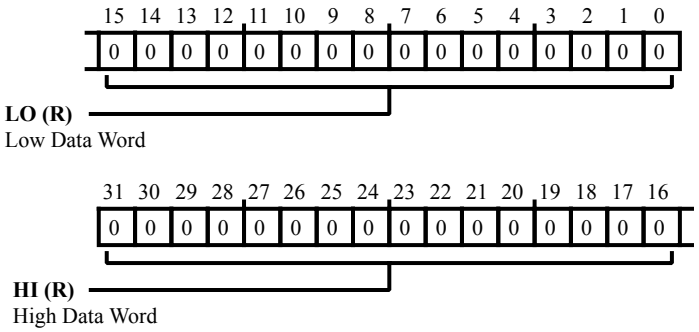

Figure 29-18: SINC_P1SEC_HIST[n] Register Diagram

Table 29-19: SINC_P1SEC_HIST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	HI	High Data Word. The SINC_P1SEC_HIST[n].HI bits provide the 16-bit sample in the most significant half of the 32- bit register.
15:0 (R/NW)	LO	Low Data Word. The SINC_P1SEC_HIST[n].LO bits provide the 16-bit sample in the least significant half of the 32- bit register.

Pair 2 Secondary (Filter) History n Register

The SINC_P2SEC_HIST[n] read-only register provides the eight most recent samples produced by secondary SINC filter 2. The 16-bit samples are stored in the 32-bit register in circular manner, starting with the low-order field of the first SINC_P2SEC_HIST[n] register. The stored values, compared to the limit, count, and window settings, set the SINC_STAT.MAX2 and SINC_STAT.MIN2 bits.

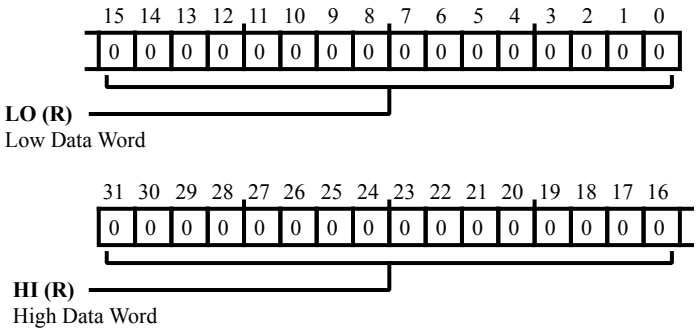

Figure 29-19: SINC_P2SEC_HIST[n] Register Diagram

Table 29-20: SINC_P2SEC_HIST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	HI	High Data Word. The SINC_P2SEC_HIST[n].HI bits provide the 16-bit sample in the most significant half of the 32- bit register.
15:0 (R/NW)	LO	Low Data Word. The SINC_P2SEC_HIST[n].LO bits provide the 16-bit sample in the least significant half of the 32- bit register.

Pair 3 Secondary (Filter) History n Register

The SINC_P3SEC_HIST[n] read-only register provides the eight most recent samples produced by secondary SINC filter 3. The 16-bit samples are stored in the 32-bit register in circular manner, starting with the low-order field of the first SINC_P3SEC_HIST[n] register. The stored values, compared to the limit, count, and window settings, set the SINC_STAT.MAX3 and SINC_STAT.MIN3 bits.

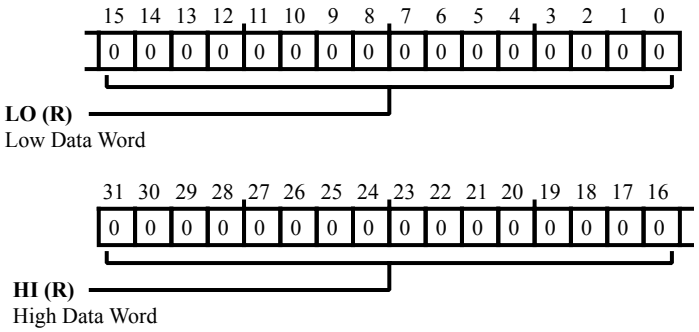

Figure 29-20: SINC_P3SEC_HIST[n] Register Diagram

Table 29-21: SINC_P3SEC_HIST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	HI	High Data Word. The SINC_P3SEC_HIST[n].HI bits provide the 16-bit sample in the most significant half of the 32- bit register.
15:0 (R/NW)	LO	Low Data Word. The SINC_P3SEC_HIST[n].LO bits provide the 16-bit sample in the least significant half of the 32- bit register.

Primary (Filters) Head for Group 0 Register

The SINC_PHEAD0 register stores the head address for a circular buffer in data memory to which to transfer the primary SINC filter outputs (according to control group 0 assignments).

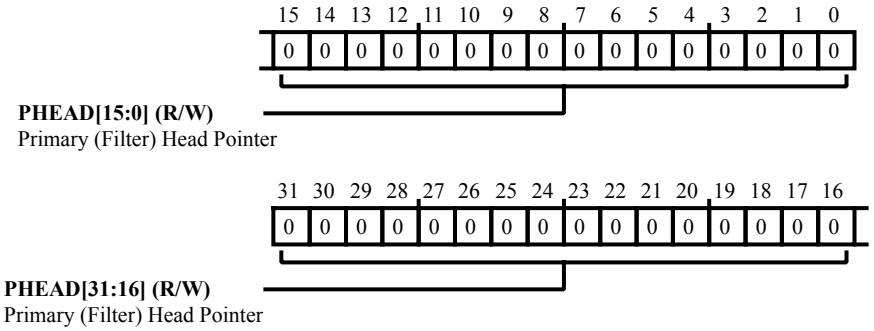

Figure 29-21: SINC_PHEAD0 Register Diagram

Table 29-22: SINC_PHEAD0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	PHEAD	Primary (Filter) Head Pointer. The SINC_PHEAD0.PHEAD bits hold the pointer (address) for DMA transfer to memory. Commencing at and wrapping back to SINC_PHEAD0.PHEAD after SINC_PTAIL0.PTAIL is reached, it forms a circular buffer, to which to transfer the primary SINC filter outputs (group 0). The valid address must be 16-bit aligned (address must be even).

Primary (Filters) Head for Group 1 Register

The SINC_PHEAD1 register stores the head address for a circular buffer in data memory to which to transfer the primary SINC filter outputs (according to control group 1 assignments).

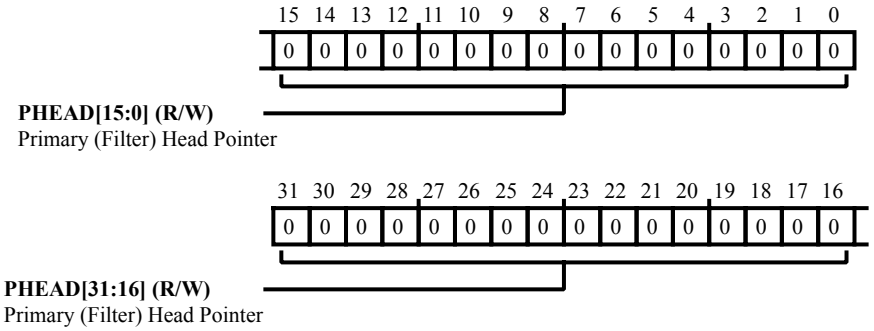

Figure 29-22: SINC_PHEAD1 Register Diagram

Table 29-23: SINC_PHEAD1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	PHEAD	Primary (Filter) Head Pointer. The SINC_PHEAD1.PHEAD bits hold the pointer (address) for DMA transfer to memory. Commencing at and wrapping back to SINC_PHEAD1.PHEAD after SINC_PTAIL1.PTAIL is reached, it forms a circular buffer, to which to transfer the primary SINC filter outputs (group 1). The valid address must be 16-bit aligned (address must be even).

Primary (Filters) Pointer for Group 0 Register

The SINC_PPTR0 read-only register points to a circular buffer holding the most recent results of primary SINC filters, according to control group 0 assignments.

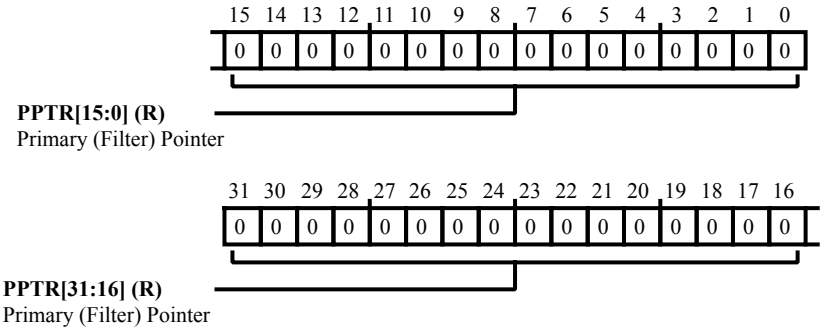

Figure 29-23: SINC_PPTR0 Register Diagram

Table 29-24: SINC_PPTR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	PPTR	Primary (Filter) Pointer. The SINC_PPTR0.PPTR bits hold the address for the last memory location of the most recent set of primary SINC filter results (group 0). The address is incremented once all of the primary SINC filter data (assigned to group 0 and associated to a particular time stamp) is successfully presented to the system fabric. Memory locations beyond the location reported by this register may be partially updated, so the entire circular buffer is not considered valid. Note that in real-time operation, due to fabric latency, write data may be in flight on the system fabric after the point when this bit field is updated. Thus, the write data may not be observed in memory until it has transited the fabric.

Primary (Filters) Pointer for Group 1 Register

The SINC_PPTR1 read-only register points to a circular buffer holding the most recent results of primary SINC filters, according to control group 1 assignments.

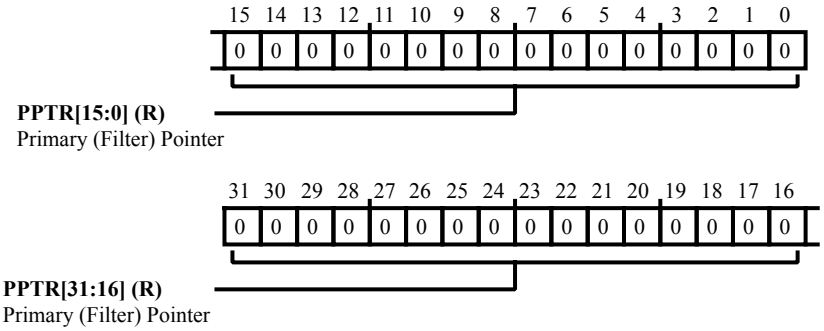

Figure 29-24: SINC_PPTR1 Register Diagram

Table 29-25: SINC_PPTR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	PPTR	Primary (Filter) Pointer. The SINC_PPTR1.PPTR bits hold the address for the last memory location of the most recent set of primary SINC filter results (group 1). The address is incremented once all of the primary SINC filter data (assigned to group 1 and associated to a particular time stamp) is successfully presented to the system fabric. Memory locations beyond the location reported by this register may be partially updated, so the entire circular buffer is not considered valid. Note that in real-time operation, due to fabric latency, write data may be in flight on the system fabric after the point when this bit field is updated. Thus, the write data may not be observed in memory until it has transited the fabric.

Primary (Filters) Tail for Group 0 Register

The SINC_PTAIL0 register stores the tail address for a circular buffer in data memory to which to transfer the primary SINC filter outputs (according to control group 1 assignments).

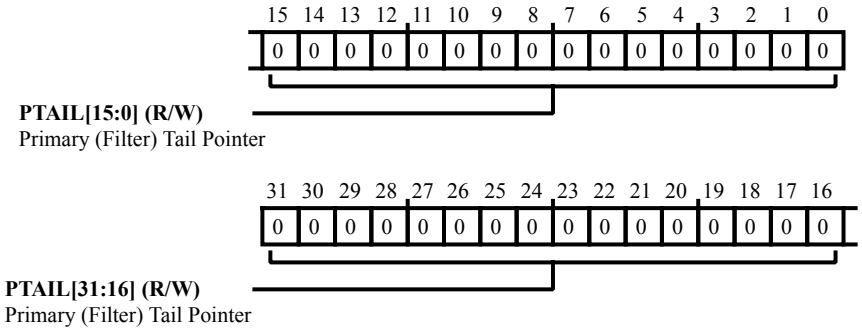

Figure 29-25: SINC_PTAIL0 Register Diagram

Table 29-26: SINC_PTAIL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	PTAIL	Primary (Filter) Tail Pointer. The SINC_PTAIL0.PTAIL bits hold the pointer (address) for DMA transfer to memory. Commencing at and wrapping back to SINC_PHEAD0.PHEAD after SINC_PTAIL0.PTAIL is reached, it forms a circular buffer, to which to transfer the primary SINC filter outputs (group 1). The valid address must be 16-bit aligned (address must be even).

Primary (Filters) Tail for Group 1 Register

The SINC_PTAIL1 register stores the tail address for a circular buffer in data memory to which to transfer the primary SINC filter outputs (according to control group 1 assignments).

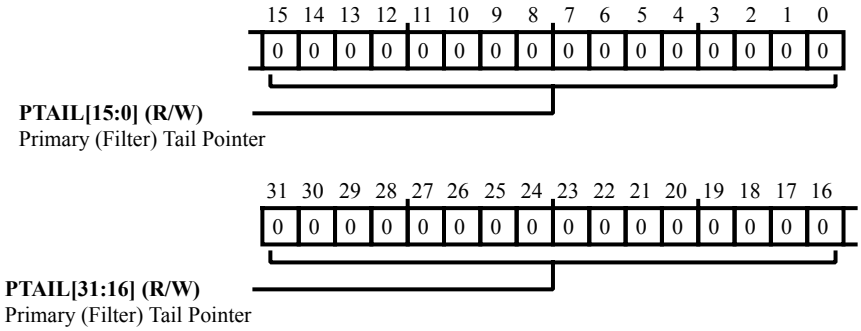

Figure 29-26: SINC_PTAIL1 Register Diagram

Table 29-27: SINC_PTAIL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	PTAIL	Primary (Filter) Tail Pointer. The SINC_PTAIL1.PTAIL bits hold the pointer (address) for DMA transfer to memory. Commencing at and wrapping back to SINC_PHEAD1.PHEAD after SINC_PTAIL1.PTAIL is reached, it forms a circular buffer, to which to transfer the primary SINC filter outputs (group 1). The valid address must be 16-bit aligned (address must be even).

Rate Control for Group 0 Register

The SINC_RATE0 register controls phase adjustments and decimation rates for primary and secondary SINC filters assigned to group 0.

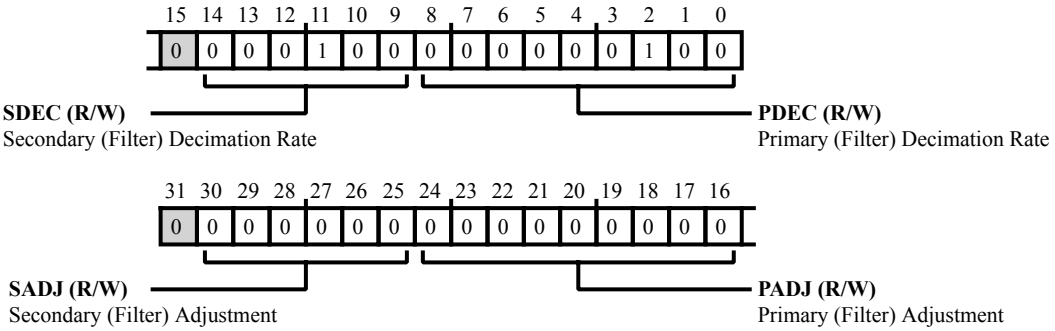

Figure 29-27: SINC_RATE0 Register Diagram

Table 29-28: SINC_RATE0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30:25 (R/W)	SADJ	Secondary (Filter) Adjustment. The SINC_RATE0.SADJ bits provide the phase adjustment for the decimated output of group 0 secondary filters. The valid adjustment is between 0 and (SINC_RATE0.SDEC - 1), in modulator clock cycles, relative to the time the filter is enabled in the SINC_CTL register. The secondary SINC filter calculates an output in modulator clock cycle equivalent to ((SINC_RATE0.SDEC * n) - SINC_RATE0.SADJ), where n is an integer > 1. This bit field can be changed while the filter is running and takes effect after the next decimation sample is generated. The effect of the change requires time to ripple through the filter: a number of output sample periods is equal to the filter order.
24:16 (R/W)	PADJ	Primary (Filter) Adjustment. The SINC_RATE0.PADJ bits provide the phase adjustment for the decimated output of group 0 primary filters. The valid adjustment is between 0 and (SINC_RATE0.PDEC - 1), in modulator clock cycles, relative to the time the filter is enabled in the SINC_CTL register. The primary SINC filter calculates an output in modulator clock cycle equivalent to ((SINC_RATE0.PDEC * n) - SINC_RATE0.PADJ), where n is an integer > 1. This bit field can be changed while the filter is running and takes effect after the next decimation sample is generated. The effect of the change requires time to ripple through the filter: a number of output sample periods is equal to the filter order.
14:9 (R/W)	SDEC	Secondary (Filter) Decimation Rate. The SINC_RATE0.SDEC bits provide the decimation rate for group 0 secondary filters. The valid range depends on the SINC order selected. If the third order (SINC_LEVEL0.SORD = 0), the valid range is 4 to 40.

Table 29-28: SINC_RATE0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		If the forth order (SINC_LEVEL0.SORD = 1), the valid rate is 4 to 16.
8:0 (R/W)	PDEC	Primary (Filter) Decimation Rate. The SINC_RATE0.PDEC bits provide the decimation rate for group 0 primary filters. The valid rate is 256 to 4.

Rate Control for Group 1 Register

The SINC_RATE1 register controls phase adjustments and decimation rates for primary and secondary SINC filters assigned to group 1.

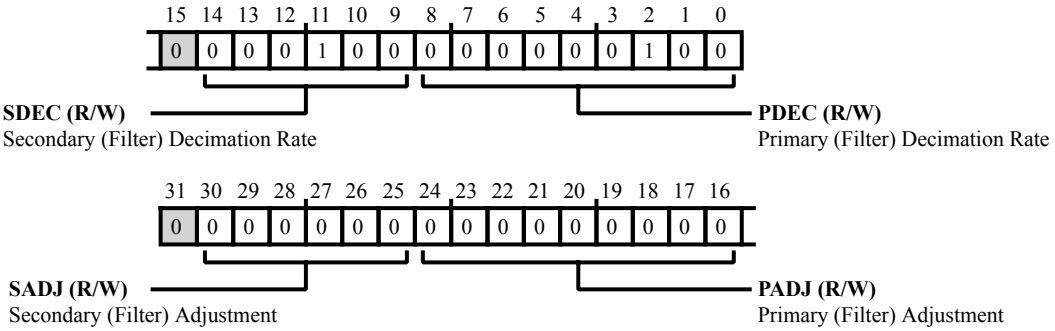

Figure 29-28: SINC_RATE1 Register Diagram

Table 29-29: SINC_RATE1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30:25 (R/W)	SADJ	Secondary (Filter) Adjustment. The SINC_RATE1.SADJ bits provide the phase adjustment for the decimated output of group 1 secondary filters. The valid adjustment is between 0 and (SINC_RATE1.SDEC - 1), in modulator clock cycles, relative to the time the filter is enabled in the SINC_CTL register. The secondary SINC filter calculates an output in modulator clock cycle equivalent to ((SINC_RATE1.SDEC * n) - SINC_RATE1.SADJ), where n is an integer > 1. This bit field can be changed while the filter is running and takes effect after the next decimation sample is generated. The effect of the change requires time to ripple through the filter: a number of output sample periods is equal to the filter order.
24:16 (R/W)	PADJ	Primary (Filter) Adjustment. The SINC_RATE1.PADJ bits provide the phase adjustment for the decimated output of group 1 primary filters. The valid adjustment is between 0 and (SINC_RATE1.PDEC - 1), in modulator clock cycles, relative to the time the filter is enabled in the SINC_CTL register. The primary SINC filter calculates an output in modulator clock cycle equivalent to ((SINC_RATE1.PDEC * n) - SINC_RATE1.PADJ), where n is an integer > 1. This bit field can be changed while the filter is running and takes effect after the next decimation sample is generated. The effect of the change requires time to ripple through the filter: a number of output sample periods is equal to the filter order.
14:9 (R/W)	SDEC	Secondary (Filter) Decimation Rate. The SINC_RATE1.SDEC bits provide the decimation rate for group 1 secondary filters. The valid range depends on the SINC order selected. If the third order (SINC_LEVEL1.SORD = 0), the valid range is 4 to 40.

Table 29-29: SINC_RATE1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		If the forth order (`SINC_LEVEL1.SORD` = 1), the valid rate is 4 to 16.
8:0 (R/W)	PDEC	Primary (Filter) Decimation Rate. The `SINC_RATE1.PDEC` bits provide the decimation rate for group 1 primary filters. The valid rate is 256 to 4.

Status Register

The SINC_STAT register indicates status for SINC output saturation, amplitude and duration limits, overload conditions, and data transfer errors.

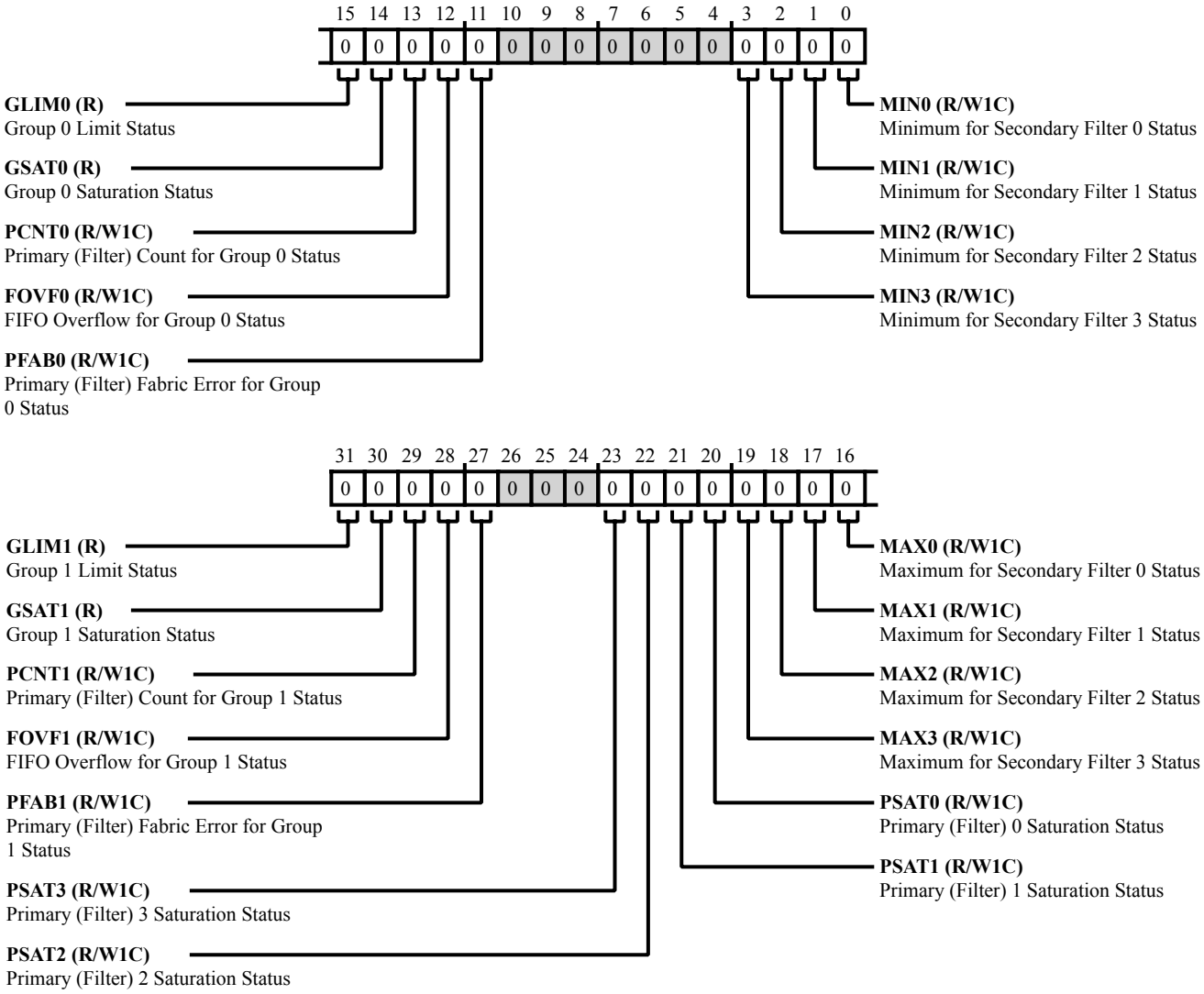

Figure 29-29: SINC_STAT Register Diagram

Table 29-30: SINC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/NW)	GLIM1	Group 1 Limit Status. The SINC_STAT.GLIM1 indicates status for an amplitude and duration limit of secondary SINC filters assigned to group 1. This bit is set (=1) if any limit specified by registers SINC_LIMIT3, SINC_LIMIT2, SINC_LIMIT1, or SINC_LIMIT0, within the duration count and window specified by bits SINC_LEVEL0.LCNT and SINC_LEVEL0.LWIN are exceeded. To identify the offending secondary SINC filter, examine the filters status bits SINC_STAT.MAX3, SINC_STAT.MAX2, SINC_STAT.MAX1, SINC_STAT.MAX0, SINC_STAT.MIN3, SINC_STAT.MIN2, SINC_STAT.MIN1 and SINC_STAT.MAX0 according to the group 1 assignments in the SINC_CTL register.	
		0	Not Exceeded
		1	Exceeded
30 (R/NW)	GSAT1	Group 1 Saturation Status. The SINC_STAT.GSAT1 indicates status for the output saturation bit of primary SINC filters assigned to group 1. The bit is set (=1) if any filter of group 1 has its saturation status bit set (=1). To identify the offending SINC primary filter, examine bits SINC_STAT.PSAT3, SINC_STAT.PSAT2, SINC_STAT.PSAT1, and SINC_STAT.PSAT0 according to the group 1 assignments specified by the SINC_CTL.EN3, SINC_CTL.EN2, SINC_CTL.EN1, and SINC_CTL.EN0 bits.	
		0	Not Set
		1	Set
29 (R/W1C)	PCNT1	Primary (Filter) Count for Group 1 Status. The SINC_STAT.PCNT1 indicates status for the output count of primary SINC filters assigned to group 1. The bit is set (=1) each time the modulo number of outputs (specified by the SINC_LEVEL1.PCNT bits) has been transferred for each primary SINC filter assigned to group 1. Each count in SINC_LEVEL1.PCNT corresponds to one complete set or vector of samples from all SINC filter pairs assigned to group 1. For example, if group 1 is assigned three SINC filters pairs 0, 1, and 3, and SINC_LEVEL1.PCNT is set to 5, then this status bit is set after the transfer of every 5th complete sample vector, comprising 3 x 5= 15 data samples. This bit asserts when the memory transfer on the system SCB fabric is complete, and a valid SCB write data response is received by the SINC filter unit. If this status bit and bit SINC_CTL.EPCNT1 are set (=1), the SINC_DATA1 trigger is asserted. Write 1 to clear.	
		0	Not Reached
		1	Reached

Table 29-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
28 (R/W1C)	FOVF1	FIFO Overflow for Group 1 Status. The SINC_STAT.FOVF1 indicates status for the data output FIFO bit of primary SINC filters assigned to group 1. This bit is set (= 1) if the output FIFO for any filter in group 1 overflows due to slow SCB fabric response. The FIFO for each primary SINC filter contains two data sample locations. An overflow occurs if a third data sample is generated before the first sample's data is transferred into the SCB fabric write data channel. After any overflow signaled by this bit occurs, all further SCB transmissions generated by group 1 are UNSPECIFIED until all SINC filters of the group are shut down and restarted. Clearing this status bit (=0) alone is not sufficient to re-sync the DMA stream. Write 1 to clear. If this status bit and bit SINC_CTL.EFOVF1 are set (=1), the SINC_STAT interrupt is asserted.	
		0	No Overflow
		1	Overflow
27 (R/W1C)	PFAB1	Primary (Filter) Fabric Error for Group 1 Status. The SINC_STAT.PFAB1 indicates error status for the output of any primary SINC filter assigned to group 1. The bit is set (=1) if the SCB fabric provides a write error response for a filter output transfer associated with group 1, or if an overrun occurs for a filter in group 1. An interrupt is requested whenever this bit =1 (not maskable).	
		0	Disabled
		1	Enabled
23 (R/W1C)	PSAT3	Primary (Filter) 3 Saturation Status. The SINC_STAT.PSAT3 bit indicates whether the primary SINC filter 3 requires saturation.	
		0	Not Saturated
		1	Saturated
22 (R/W1C)	PSAT2	Primary (Filter) 2 Saturation Status. The SINC_STAT.PSAT2 bit indicates whether the primary SINC filter 2 requires saturation.	
		0	Not Saturated
		1	Saturated

Table 29-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W1C)	PSAT1	Primary (Filter) 1 Saturation Status. The SINC_STAT.PSAT1 bit indicates whether the primary SINC filter 1 requires saturation.	
		0	Not Saturated
		1	Saturated
20 (R/W1C)	PSAT0	Primary (Filter) 0 Saturation Status. The SINC_STAT.PSAT0 bit indicates whether the primary SINC filter 0 requires saturation.	
		0	Not Saturated
		1	Saturated
19 (R/W1C)	MAX3	Maximum for Secondary Filter 3 Status. The SINC_STAT.MAX3 bit indicates whether the output of the secondary SINC filter 3 exceeded its maximum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the SINC_LIMIT3.LMAX bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the SINC_CTL.EN3 bits. For group 0, the duration limit is SINC_LEVEL0.LCNT counts within a window of SINC_LEVEL0.LWIN samples. For group 1, the duration limit is SINC_LEVEL1.LCNT counts within a window of SINC_LEVEL1.LWIN samples.	
		0	Not Exceeded
		1	Exceeded
18 (R/W1C)	MAX2	Maximum for Secondary Filter 2 Status. The SINC_STAT.MAX2 bit indicates whether the output of the secondary SINC filter 2 exceeded its maximum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the SINC_LIMIT2.LMAX bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the SINC_CTL.EN2 bits. For group 0, the duration limit is SINC_LEVEL0.LCNT counts within a window of SINC_LEVEL0.LWIN samples. For group 1, the duration limit is SINC_LEVEL1.LCNT counts within a window of SINC_LEVEL1.LWIN samples.	
		0	Not Exceeded
		1	Exceeded

Table 29-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W1C)	MAX1	Maximum for Secondary Filter 1 Status. The SINC_STAT.MAX1 bit indicates whether the output of the secondary SINC filter 0 exceeded its maximum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the SINC_LIMIT1.LMAX bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the SINC_CTL.EN1 bits. For group 0, the duration limit is SINC_LEVEL0.LCNT counts within a window of SINC_LEVEL0.LWIN samples. For group 1, the duration limit is SINC_LEVEL1.LCNT counts within a window of SINC_LEVEL1.LWIN samples.	
		0	Not Exceeded
		1	Exceeded
16 (R/W1C)	MAX0	Maximum for Secondary Filter 0 Status. The SINC_STAT.MAX0 bit indicates whether the output of the secondary SINC filter 0 exceeded its maximum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the SINC_LIMIT0.LMAX bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the SINC_CTL.EN0 bits. For group 0, the duration limit is SINC_LEVEL0.LCNT counts within a window of SINC_LEVEL0.LWIN samples. For group 1, the duration limit is SINC_LEVEL1.LCNT counts within a window of SINC_LEVEL1.LWIN samples.	
		0	Not Exceeded
		1	Exceeded
15 (R/NW)	GLIM0	Group 0 Limit Status. The SINC_STAT.GLIM0 indicates status for an amplitude and duration limit of secondary SINC filters assigned to group 0. This bit is set (=1) if any limit specified by registers SINC_LIMIT3, SINC_LIMIT2, SINC_LIMIT1, or SINC_LIMIT0, within the duration count and window specified by bits SINC_LEVEL1.LCNT and SINC_LEVEL1.LWIN are exceeded. To identify the offending secondary SINC filter, examine the filters status bits SINC_STAT.MAX3, SINC_STAT.MAX2, SINC_STAT.MAX1, SINC_STAT.MAX0, SINC_STAT.MIN3, SINC_STAT.MIN2, SINC_STAT.MIN1 and SINC_STAT.MAX0 according to the group 0 assignments in the SINC_CTL register.	
		0	Not Exceeded
		1	Exceeded

Table 29-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/NW)	GSAT0	Group 0 Saturation Status. The SINC_STAT.GSAT0 indicates status for the output saturation bit of primary SINC filters assigned to group 0. The bit is set (=1) if any filter of group 0 has its saturation status bit set (=1). To identify the offending SINC primary filter, examine bits SINC_STAT.PSAT3, SINC_STAT.PSAT2, SINC_STAT.PSAT1, and SINC_STAT.PSAT0 according to the group 0 assignments specified by the SINC_CTL.EN3, SINC_CTL.EN2, SINC_CTL.EN1, and SINC_CTL.EN0 bits.	
		0	Not Set
		1	Set
13 (R/W1C)	PCNT0	Primary (Filter) Count for Group 0 Status. The SINC_STAT.PCNT0 indicates status for the output count of primary SINC filters assigned to group 0. The bit is set (=1) each time the modulo number of outputs (specified by the SINC_LEVEL0.PCNT bits) has been transferred for each primary SINC filter assigned to group 0. Each count in SINC_LEVEL0.PCNT corresponds to one complete set or vector of samples from all SINC filter pairs assigned to group 1. For example, if group 0 is assigned three SINC filters pairs 0, 1, and 3, and SINC_LEVEL0.PCNT is set to 5, then this status bit is set after the transfer of every 5th complete sample vector, comprising 3 x 5= 15 data samples. This bit asserts when the memory transfer on the system SCB fabric is complete, and a valid SCB write data response is received by the SINC filter unit. If this status bit and bit SINC_CTL.EPCNT0 are set (=1), the SINC_DATA0 trigger is asserted. Write 1 to clear.	
		0	Not Reached
		1	Reached

Table 29-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1C)	FOVF0	FIFO Overflow for Group 0 Status. The SINC_STAT.FOVF0 indicates status for the data output FIFO bit of primary SINC filters assigned to group 0. This bit is set (= 1) if the output FIFO for any filter in group 0 overflows due to slow SCB fabric response. The FIFO for each primary SINC filter contains two data sample locations. An overflow occurs if a third data sample is generated before the first sample's data is transferred into the SCB fabric write data channel. After any overflow signaled by this bit occurs, all further SCB transmissions generated by group 1 are UNSPECIFIED until all SINC filters of the group are shut down and restarted. Clearing this status bit (=0) alone is not sufficient to re-sync the DMA stream. Write 1 to clear. If this status bit and bit SINC_CTL.EFOVF0 are set (=1), the SINC_STAT interrupt is asserted.	
		0	No Overflow
		1	Overflow
11 (R/W1C)	PFAB0	Primary (Filter) Fabric Error for Group 0 Status. The SINC_STAT.PFAB0 indicates error status for the output of any primary SINC filter assigned to group 0. The bit is set (=1) if the SCB fabric provides a write error response for a filter output transfer associated with group 0, or if an overrun occurs for a filter in group 0. An interrupt is requested whenever this bit is =1 (not maskable).	
		0	Disabled
		1	Enabled
3 (R/W1C)	MIN3	Minimum for Secondary Filter 3 Status. The SINC_STAT.MIN3 bit indicates whether the output of the secondary SINC filter 3 exceeded its minimum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the SINC_LIMIT3.LMIN bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the SINC_CTL.EN3 bits. For group 0, the duration limit is SINC_LEVEL0.LCNT counts within a window of SINC_LEVEL0.LWIN samples. For group 1, the duration limit is SINC_LEVEL1.LCNT counts within a window of SINC_LEVEL1.LWIN samples.	
		0	Not Exceeded
		1	Exceeded

Table 29-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	MIN2	Minimum for Secondary Filter 2 Status. The SINC_STAT.MIN2 bit indicates whether the output of the secondary SINC filter 2 exceeded its minimum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the SINC_LIMIT2.LMIN bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the SINC_CTL.EN2 bits. For group 0, the duration limit is SINC_LEVEL0.LCNT counts within a window of SINC_LEVEL0.LWIN samples. For group 1, the duration limit is SINC_LEVEL1.LCNT counts within a window of SINC_LEVEL1.LWIN samples.	
		0	Not Exceeded
		1	Exceeded
1 (R/W1C)	MIN1	Minimum for Secondary Filter 1 Status. The SINC_STAT.MIN1 bit indicates whether the output of the secondary SINC filter 1 exceeded its minimum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the SINC_LIMIT1.LMIN bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the SINC_CTL.EN1 bits. For group 0, the limit is SINC_LEVEL0.LCNT counts within a window of SINC_LEVEL0.LWIN samples. For group 1, the limit is SINC_LEVEL1.LCNT counts within a window of SINC_LEVEL1.LWIN samples.	
		0	Not Exceeded
		1	Exceeded

Table 29-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W1C)	MIN0	Minimum for Secondary Filter 0 Status. The SINC_STAT.MIN0 bit indicates whether the output of the secondary SINC filter 0 exceeded its minimum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the SINC_LIMIT0.LMIN bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the SINC_CTL.EN0 bits. For group 0, the limit is SINC_LEVEL0.LCNT counts within a window of SINC_LEVEL0.LWIN samples. For group 1, the limit is SINC_LEVEL1.LCNT counts within a window of SINC_LEVEL1.LWIN samples.	
		0	Not Exceeded
		1	Exceeded

30 Reset Control Unit (RCU)

Reset is the initial state of the whole processor (or one of the cores). It is the result of a hardware or software triggered event. In this state, all control registers are set to their default values and functional units are idle. Exiting a full system reset starts with Core-0 only being ready to boot. Exiting a Core n only reset starts with this Core n being ready to boot.

The reset control unit (RCU) controls how all the functional units enter and exit reset. Differences in functional requirements and clocking constraints define how reset signals are generated. Programs must guarantee that none of the reset functions puts the system into an undefined state or causes resources to stall. This functionality is important when only one of the cores is reset (programs must ensure that there is no pending system activity involving the core that is being reset).

RCU Features

The RCU module supports the following features:

- Hardware reset through the $\overline{\texttt{SYS\_HWRST}}$ pin
- Software system reset through RCU registers
- Software system reset by the ARM core
- Hardware system reset through:
  - TRU module
  - SEC module
  - the oscillator watchdog module of the CGU
- Core reset through RCU registers

RCU Functional Description

This section provides information on the function of RCU module.

Hardware reset using SYS_HWRST pin

Asserting the SYS_HWRST pin resets all functional units, except the RTC (if present)

Hardware reset through RCU

RCU can perform a full system reset which can be initiated through hardware blocks like the SEC, the TRU, and the oscillator watchdog

Software reset using RCU registers

Setting RCU_STAT.SWRST issues a software reset for all system units except the RTC module and RCU_BCODE, RCU_CRCTL and RCU_STAT registers

Core reset RCU registers

Core can be individually reset by software, or by setting the CRn bit in the RCU_CRCTL register

ADSP-CM40x RCU Trigger List

Table 30-1: ADSP-CM40x RCU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
None			

Table 30-2: ADSP-CM40x RCU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
0	RCU0_SYSRST0	RCU0 System Reset Assert (Slave 0)	None
1	RCU0_SYSRST1	RCU0 System Reset Assert (Slave 1)	None

ADSP-CM40x RCU Register List

The Reset Control Unit (RCU) controls how all the functional units in the processor enter and exit Reset. Differences in functional requirements and clocking constraints exist (units in different clock domains have to enter reset asynchronously, but units exit reset in a deterministic way) define how reset signals are generated. Reset signals propagate through all the functional units asynchronously. For more information on RCU functionality, see the RCU register descriptions.

Table 30-3: ADSP-CM40x RCU Register List

Name	Description
RCU_BCODE	Boot Code Register
RCU_CRCTL	Core Reset outputs Control Register

Table 30-3: ADSP-CM40x RCU Register List (Continued)

Name	Description
RCU_CRSTAT	Core Reset outputs Status Register
RCU_CTL	Control Register
RCU_MSG	Message Register
RCU_MSG_CLR	Message Clear Bits Register
RCU_MSG_SET	Message Set Bits Register
RCU_REVID	Revision ID Register
RCU_SRRQSTAT	System Reset Request Status
RCU_STAT	Status Register
RCU_SVECT0	Software Vector Register 0
RCU_SVECT_LCK	SVECT Lock Register

RCU Definitions

To make the best use of the RCU, it is useful to understand the terms in this section.

The target or source defines the following are types of resets.

Hardware Reset (by target)

All functional units except the debug interface are set to their default states. History is lost.

System Reset (by target)

All functional units except the RCU are set to their default states.

Core n Only Reset (by target)

Affects Core n only. The system software must guarantee that a bus master cannot access the core in reset state.

Hardware Reset (by source)

The $\overline{\text{SYS\_HWRST}}$ input signal is asserted active (pulled low).

System Reset (by source)

Software can trigger the reset by writing to the RCU_CONTROL register or by another functional unit such as the TRU or any of the generic reset inputs.

Core n Only Reset (by source)

For CORERESETEN =0, triggered by software. For CORERESETEN =1, triggered by software or system.

RCU Architectural Concepts

To understand the architecture of the RCU, it is important to consider the reset sources and how differing resets affect the functional units of the processor.

The RCU provides the hardware that controls how all the functional units enter and exit reset. Differences in functional requirements and clocking constraints define how reset signals are generated. For example, units in different clock domains must enter reset asynchronously but exit reset in a deterministic way.

The program must guarantee that none of the reset functions put the system in an undefined state or cause resources to stall. This functionality is important when only one of the cores is reset. The program must guarantee that there is no pending system activity involving Core n before it is reset. For example, there must be no pending transactions to core n when the core is reset.

The ***RCU Reset Sources*** table defines how reset sources affect the different functional units.

Table 30-4: RCU Reset Sources

Reset Source	Reset Type	Affected Functional Units
$\overline{\texttt{SYS\_HWRST}}$ pin assertion	Hardware Reset	All functional units, except RTC (if present)
SYSCLK clock domain system reset	System Reset	All functional units, except: • RTC (if present), • RCU_STAT, • RCU_BCODE, and • the units on the VDDEXT power domain
RCU_CTL.SYSRST bit set (software triggered reset)	System Reset	All functional units, except: • RTC (if present), • RCU_STAT, • RCU_BCODE, and • the units on the VDDEXT power domain
RCU_CRCTL.CRN bit set (software triggered reset)	Core Only Reset	Core n only, for ($15 \geq n \geq 0$)

RCU Status and Error Signals

The RCU_STAT register reflects status and error information. There are three kinds of errors that can occur in the RCU. The reset out error is triggered when RSTOUT is both asserted and deasserted at the same time. The lock write error occurs if an attempt is made to write a lock RCU register. The address error occurs if a read-only register is written to or if an attempt is made to a reserved address within the RCU MMR address range.

Resetting a Core Through a System Master

The RCU allows reset of a given core n using a system master.

NOTE: In processors with multiple cores, another core can act as a system master.

A core can be reset by software, either by setting any of the RCU_CRCTL.CRN bits ($15 \geq n \geq 0$). A core that resets itself cannot guarantee that all the system transactions to or from it have completed. Although a core reset can be triggered by the core itself, it is recommended that a system master trigger the reset. The core can then be reset to restore its proper operation when it cannot execute software. The following steps provide the suggested sequence to reset a core.

1. Clear the RCU_CRSTAT.CRN bit.
2. Disable interrupts to the core.
3. Set the RCU_CRCTL.CRN bit to reset the core.
4. Poll the RCU_CRSTAT.CRN bit until the core is in reset.
5. Clear the RCU_CRCTL.CRN bit to take the core out of reset.
6. Poll the RCU_CRSTAT.CRN bit until the core is out of reset.

ADSP-CM40x Specific Information

The following RCU related information is specific to ADSP-CM40x processors. When applying RCU feature to ADSP-CM40x systems, be aware that:

- The ADSP-CM40x processor is a single core processor.
- The ADSP-CM40x processor does not have an RTC (real-time clock).
- The SCLK domain sources for system reset come from the TRU and SEC (same as SYSCLK sources).
- The SYSCLK domain sources for system reset come from the TRU and SEC.
- The RCU on the ADSP-CM40x processor supports oscillator watchdog reset.
- Core reset through the RCU on the ADSP-CM40x processor is not supported.

NOTE: L1 memory and cache contents are not preserved though a soft reset transition unless either ***(a)*** the chip is in bypass or ***(b)*** the CGU CCLK buffer is disabled.

ADSP-CM40x RCU Register Descriptions

Reset Control Unit (RCU) contains the following registers.

Table 30-5: ADSP-CM40x RCU Register List

Name	Description
RCU_BCODE	Boot Code Register
RCU_CRCTL	Core Reset outputs Control Register

Table 30-5: ADSP-CM40x RCU Register List (Continued)

Name	Description
RCU_CRSTAT	Core Reset outputs Status Register
RCU_CTL	Control Register
RCU_MSG	Message Register
RCU_MSG_CLR	Message Clear Bits Register
RCU_MSG_SET	Message Set Bits Register
RCU_REVID	Revision ID Register
RCU_SRRQSTAT	System Reset Request Status
RCU_STAT	Status Register
RCU_SVECT0	Software Vector Register 0
RCU_SVECT_LCK	SVECT Lock Register

Boot Code Register

The RCU_BCODE register can be used to determine if and how core boots. This register is set to its default values by Hard Reset.

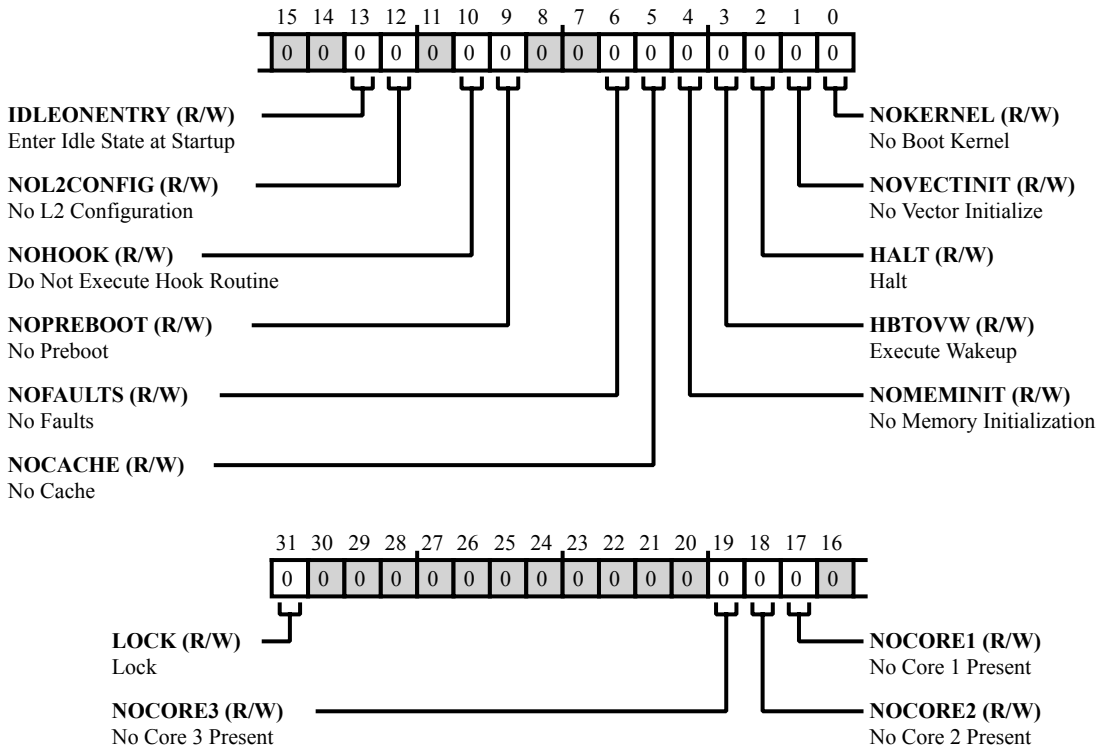

Figure 30-1: RCU_BCODE Register Diagram

Table 30-6: RCU_BCODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the RCU_BCODE.LOCK bit is set, the RCU_BCODE register is read only (locked).	
		0	Unlock
		1	Lock
19 (R/W)	NOCORE3	No Core 3 Present. The RCU_BCODE.NOCORE3 bit indicates the presence of core 3.	
		0	Core does not exist
		1	Core exists

Table 30-6: RCU_BCODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	NOCORE2	No Core 2 Present. The RCU_BCODE.NOCORE2 bit indicates the presence of core 2.	
		0	Core does not exist
		1	Core exists
17 (R/W)	NOCORE1	No Core 1 Present. The RCU_BCODE.NOCORE1 bit indicates the presence of core 1.	
		0	Core does not exist
		1	Core exists
13 (R/W)	IDLEONENTRY	Enter Idle State at Startup. The RCU_BCODE.IDLEONENTRY bit configures the Boot ROM to enter the idle state at startup.	
		0	Do not enter idle state
		1	Enter idle state
12 (R/W)	NOL2CONFIG	No L2 Configuration. The RCU_BCODE.NOL2CONFIG bit configures the Boot ROM to not perform the L2 memory configuration.	
		0	Configure L2 memory
		1	Do not configure L2 memory
10 (R/W)	NOHOOK	Do Not Execute Hook Routine. The RCU_BCODE.NOHOOK bit configures the Boot ROM to not perform the hook routine.	
		0	Perform hook routine
		1	Do not perform hook routine
9 (R/W)	NOPREBOOT	No Preboot. The RCU_BCODE.NOPREBOOT bit configures the Boot ROM to not perform the customer preboot routine.	
		0	Perform preboot
		1	Do not perform preboot
6 (R/W)	NOFAULTS	No Faults. The RCU_BCODE.NOFAULTS bit configures the Boot ROM to not perform fault initialization.	
		0	Perform fault initialization
		1	Do not perform fault initialization

Table 30-6: RCU_BCODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	NOCACHE	No Cache. The RCU_BCODE.NOCACHE bit configures the Boot ROM to not perform a cache initialization and to not enable the cache.	
		0	Enable and initialize cache
		1	Do not initialize or enable cache
4 (R/W)	NOMEMINIT	No Memory Initialization. The RCU_BCODE.NOMEMINIT bit configures the Boot ROM to not perform a memory initialization.	
		0	Perform memory initialization
		1	Do not perform memory initialization
3 (R/W)	HBTOVW	Execute Wakeup. The RCU_BCODE.HBTOVW bit configures the Boot ROM to execute a wakeup.	
		0	Do not wakeup
		1	Execute wakeup
2 (R/W)	HALT	Halt. The RCU_BCODE.HALT bit configures the Boot ROM to execute the no boot routine.	
		0	Do not execute routine
		1	Execute routine
1 (R/W)	NOVECTINIT	No Vector Initialize. The RCU_BCODE.NOVECTINIT bit configures the Boot ROM to not vector to the application.	
		0	Vector
		1	Do not vector
0 (R/W)	NOKERNEL	No Boot Kernel. The RCU_BCODE.NOKERNEL bit configures the Boot ROM to not execute the boot kernel.	
		0	Execute boot kernel
		1	Do not execute boot kernel

Core Reset outputs Control Register

The RCU core reset control n registers (RCU_CRCTL) include a lock bit (RCU_CRCTL.LOCK) and a core reset bit (RCU_CRCTL.CRN) for each core reset signal on the product. Only bits enabled by the CORE_DISABLE_MASK can be written. Bits disabled by the CORE_DISABLE_MASK are read as "1" and cannot be modified by software.

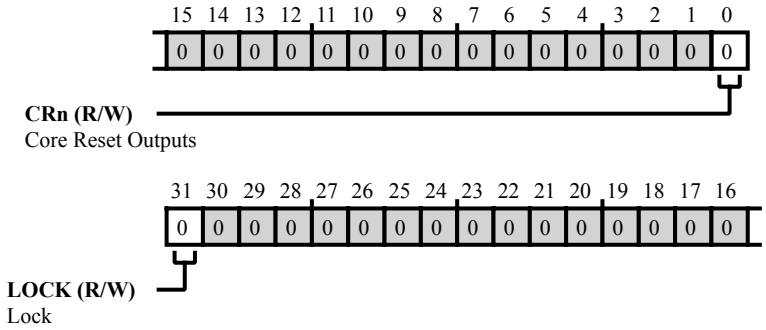

Figure 30-2: RCU_CRCTL Register Diagram

Table 30-7: RCU_CRCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the RCU_CRCTL.LOCK bit is set, the RCU_CRCTL register is read only (locked).	
		0	Unlock
		1	Lock
0 (R/W)	CRN	Core Reset Outputs. The RCU_CRCTL.CRN bits control CRES[0:0] Core Reset signals. RCU_CRES[n] can be individually controlled. It is reset to its default value by Hard Reset or by System Reset RCU_CRES[n] the selected RCU0_CRMSKi[n] bit is cleared.	
		0	RCU_CRES[0:0] Deasserted
		1	RCU_CRES[0:0] Asserted

Core Reset outputs Status Register

The RCU core reset status register (RCU_CRSTAT) contains status bits, indicating which core reset signals have been asserted.

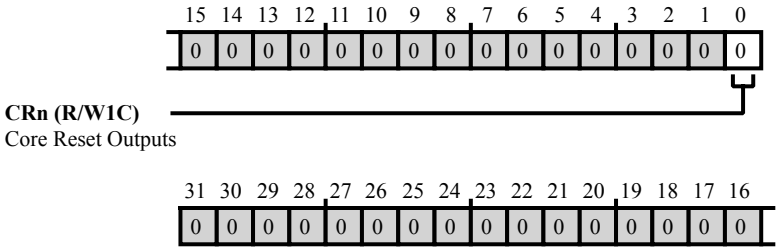

Figure 30-3: RCU_CRSTAT Register Diagram

Table 30-8: RCU_CRSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W1C)	CRN	Core Reset Outputs. The RCU_CRSTAT.CRN bits indicate which cores have been reset since the last time the bit was cleared. Bits masked by CORE_DISABLE_MASK[15:0] are permanently disabled and the corresponding CR bits set. CR bits are sticky, they need to be cleared by software.	
		0	RCU_CRES[0:0] Deasserted CR[n] corresponds to RCU_CRES[n]
		1	RCU_CRES[0:0] were asserted since the last time bits were cleared. CR[n] corresponds to RCU_CRES[n]

Control Register

The RCU control register (RCU_CTL) provides a register lock, enables for the core and system reset requests inputs and control for the Reset Output pin.

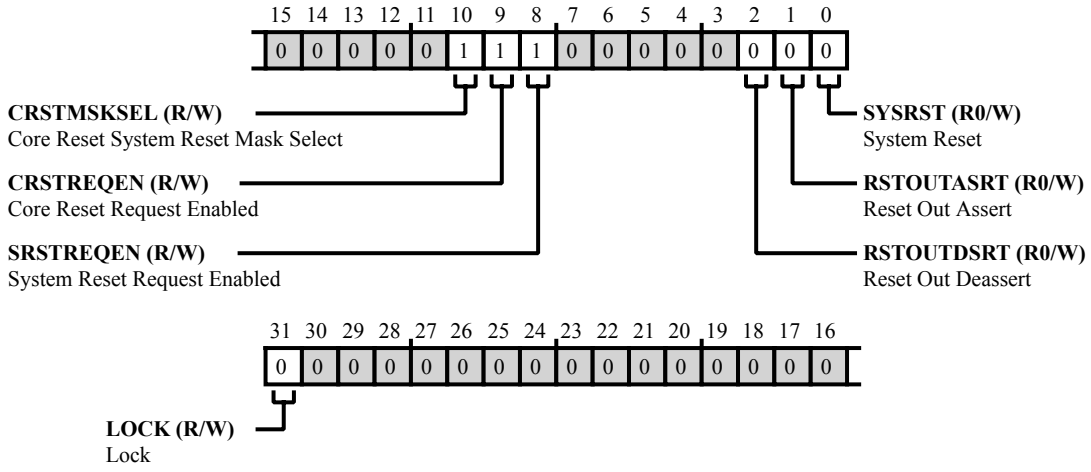

Figure 30-4: RCU_CTL Register Diagram

Table 30-9: RCU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the RCU_CTL.LOCK bit is set, the RCU_CTL register is read only (locked). This bit is cleared by Hard Reset or any System Reset event.	
		0	Unlock
		1	Lock
10 (R/W)	CRSTMSKSEL	Core Reset System Reset Mask Select. The RCU_CTL.CRSTMSKSEL bit selects the Core Reset System Reset mask. This bit is cleared by Hard Reset.	
9 (R/W)	CRSTREQEN	Core Reset Request Enabled. The RCU_CTL.CRSTREQEN bit controls whether the SYSCLK domain source(s) of reset is/are enabled to reset the core(s) when asserted. This bit is cleared by Hard Reset or any System Reset event.	
		0	Disabled
		1	Enabled

Table 30-9: RCU_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	SRSTREQEN	System Reset Request Enabled. The RCU_CTL.SRSTREQEN bit controls whether the SYSCLK domain sources of reset are enabled to do a system reset when asserted. This bit is cleared by Hard Reset.	
		0	Disabled
		1	Enabled
2 (R0/W)	RSTOUTDSRT	Reset Out Deassert. The RCU_CTL.RSTOUTDSRT bit controls de-assertion of the system reset pin.	
		0	No Action
		1	Deassert RSTOUT
1 (R0/W)	RSTOUTASRT	Reset Out Assert. The RCU_CTL.RSTOUTASRT bit controls assertion of the system reset pin.	
		0	No Action
		1	Assert RSTOUT
0 (R0/W)	SYSRST	System Reset. The RCU_CTL.SYSRST bit provides reset for all system units.	
		0	No Action
		1	System Reset

Message Register

The RCU_MSG is a general-purpose register. It is intended to provide flexibility for Boot ROM code and to pass predefined variables to the debugger. Please see the Booting chapter for product-specific details.

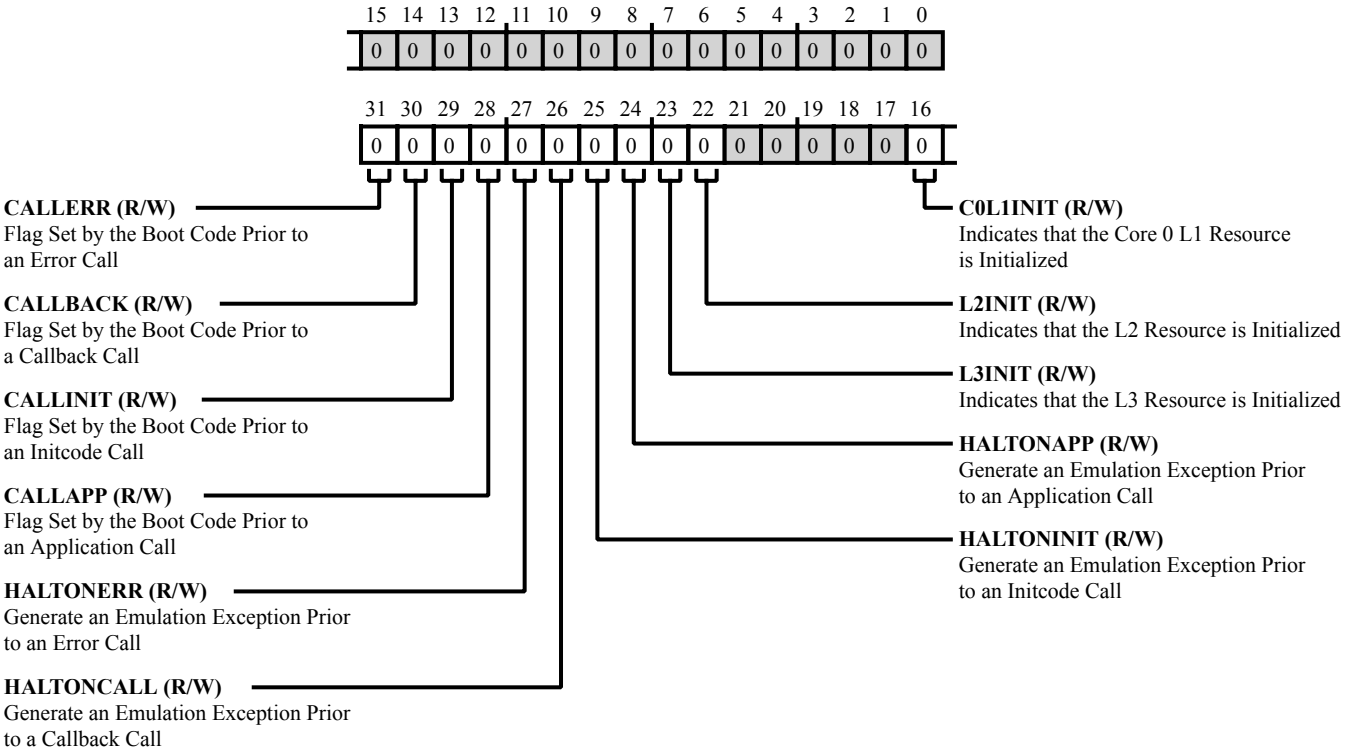

Figure 30-5: RCU_MSG Register Diagram

Table 30-10: RCU_MSG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	CALLERR	Flag Set by the Boot Code Prior to an Error Call. The RCU_MSG.CALLERR bit indicates that a flag has been set by the boot code prior to an error call.
30 (R/W)	CALLBACK	Flag Set by the Boot Code Prior to a Callback Call. The RCU_MSG.CALLBACK bit indicates that a flag has been set by the boot code prior to a callback call.
29 (R/W)	CALLINIT	Flag Set by the Boot Code Prior to an Initcode Call. The RCU_MSG.CALLINIT bit indicates that a flag has been set by the boot code prior to an initcode call.
28 (R/W)	CALLAPP	Flag Set by the Boot Code Prior to an Application Call.

Table 30-10: RCU_MSG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		The RCU_MSG.CALLAPP bit indicates that a flag has been set by the boot code prior to an application call.
27 (R/W)	HALTONERR	Generate an Emulation Exception Prior to an Error Call. The RCU_MSG.HALTONERR bit generates an emulation exception prior to an error call.
26 (R/W)	HALTONCALL	Generate an Emulation Exception Prior to a Callback Call. The RCU_MSG.HALTONCALL bit generates an emulation exception prior to a callback call.
25 (R/W)	HALTONINIT	Generate an Emulation Exception Prior to an Initcode Call. The RCU_MSG.HALTONINIT bit generates an emulation exception prior to an init-code call.
24 (R/W)	HALTONAPP	Generate an Emulation Exception Prior to an Application Call. The RCU_MSG.HALTONAPP bit generates an emulation exception prior to an application call.
23 (R/W)	L3INIT	Indicates that the L3 Resource is Initialized. The RCU_MSG.L3INIT bit indicates that the L3 resource is initialized.
22 (R/W)	L2INIT	Indicates that the L2 Resource is Initialized. The RCU_MSG.L2INIT bit indicates that the L2 resource is initialized.
16 (R/W)	C0L1INIT	Indicates that the Core 0 L1 Resource is Initialized. The RCU_MSG.C0L1INIT bit indicates that the core 0 L1 resource is initialized.

Message Clear Bits Register

The RCU_MSG_CLR register is used to clear bits in RCU_MSG register. Reading this register returns 0x00000000.

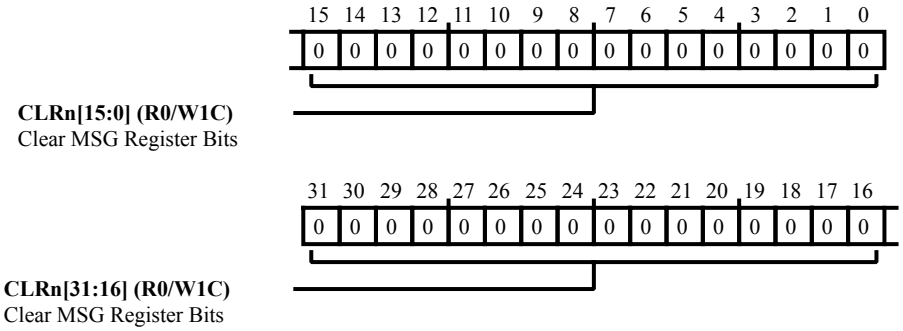

Figure 30-6: RCU_MSG_CLR Register Diagram

Table 30-11: RCU_MSG_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W1C)	CLRN	Clear MSG Register Bits. The RCU_MSG_CLR.CLRN bit resets MSG bit n.

Message Set Bits Register

The RCU_MSG_SET register is used to set bits in RCU_MSG register. Reading this register returns 0x00000000.

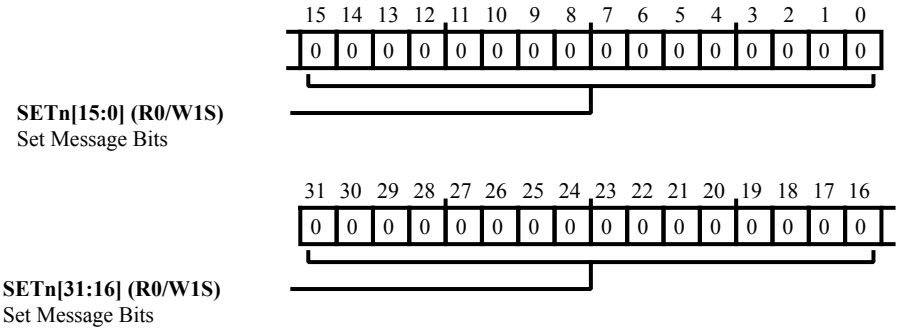

Figure 30-7: RCU_MSG_SET Register Diagram

Table 30-12: RCU_MSG_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W1S)	SETN	Set Message Bits. The RCU_MSG_SET.SETN bit sets MSG bit n.

Revision ID Register

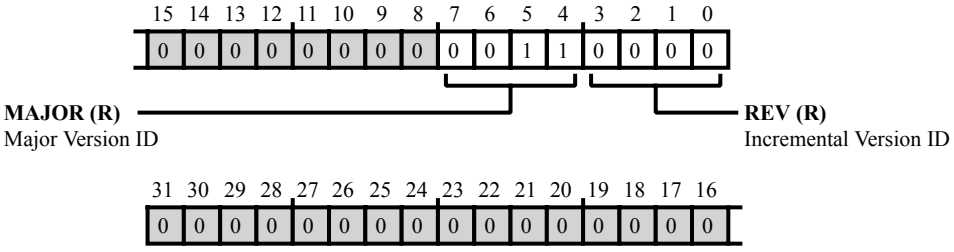

Figure 30-8: RCU_REVID Register Diagram

Table 30-13: RCU_REVID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	MAJOR	Major Version ID.
3:0 (R/NW)	REV	Incremental Version ID.

System Reset Request Status

The RCU system reset request status register (RCU_SRRQSTAT) contains status bits, indicating which system reset request input triggered System Reset. This register is set to its default values by Hard Reset.

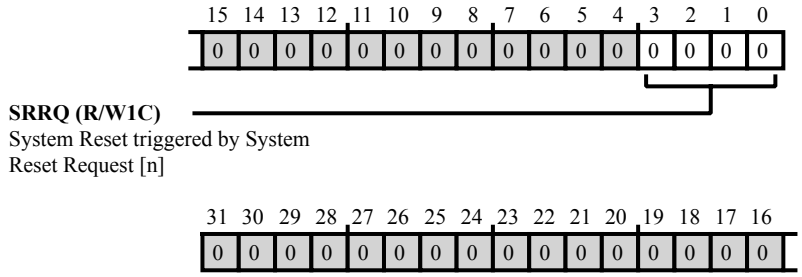

Figure 30-9: RCU_SRRQSTAT Register Diagram

Table 30-14: RCU_SRRQSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0 (R/W1C)	SRRQ	System Reset triggered by System Reset Request [n]. These bits are set by the assertion of the corresponding System Reset Request input and de-asserted by Writing "1" to the bit. This register is cleared by Hard Reset.	
		0	SYSCD_RESETb[3:0] have not been asserted since last Hard Reset or since the bit was cleared
		15	SYSCD_RESETb[3:0] have been asserted since last Hard Reset or since the bit was cleared

Status Register

The RCU status register (RCU_STAT) contains status bits for all RCU reset sources, reset status, and boot mode inputs. Status bits for reset sources are sticky and can cleared by software. Error status bits are cleared by any reset event.

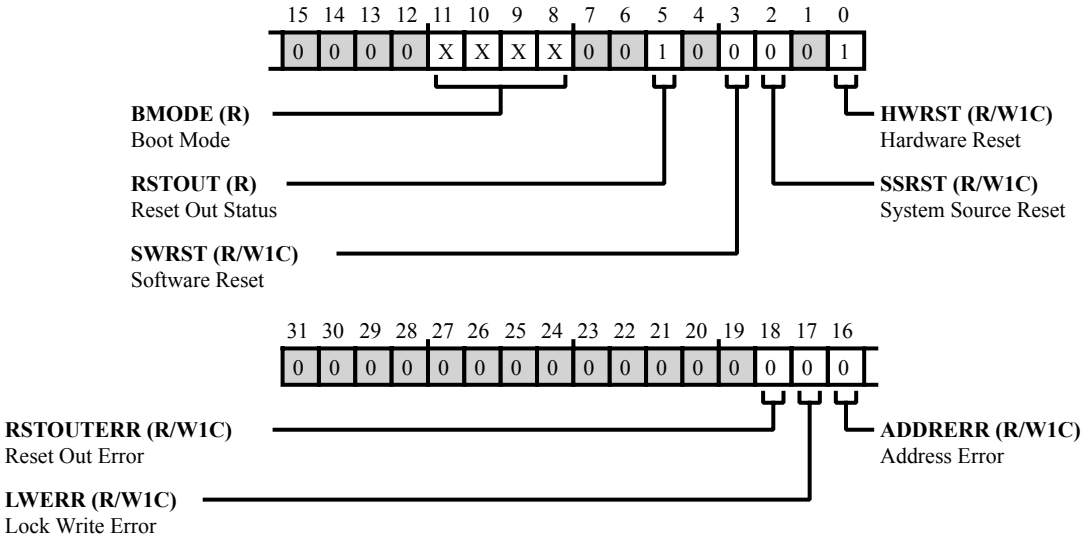

Figure 30-10: RCU_STAT Register Diagram

Table 30-15: RCU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W1C)	RSTOUTERR	Reset Out Error. The RCU_STAT.RSTOUTERR bit indicates (if set) that a write attempted to set the RCU_CTL.RSTOUTASRT and RCU_CTL.RSTOUTDSRT simultaneously. This condition triggers a bus error.	
		0	No Error
		1	Error Occurred
17 (R/W1C)	LWERR	Lock Write Error. The RCU_STAT.LWERR bit indicates (when set) there was an attempted write to an RCU register while the RCU_CTL.LOCK bit was set and the global lock bit is enabled (SPU_CTL.GLCK bit =1). This status bit is sticky; write-1-to-clear	
		0	No Error
		1	Error Occurred

Table 30-15: RCU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W1C)	ADDRERR	Address Error. The RCU_STAT.ADDRERR bit indicates that the RCU generated an address error. This status bit is sticky; write-1-to-clear it.	
		0	No Error
		1	Error Occurred
11:8 (R/NW)	BMODE	Boot Mode. The RCU_STAT.BMODE bits indicate the input on the boot mode pins.	
5 (R/NW)	RSTOUT	Reset Out Status. The RCU_STAT.RSTOUT bit indicates the assertion status of the system reset pin.	
		0	RSTOUT Deasserted
		1	RSTOUT Asserted
3 (R/W1C)	SWRST	Software Reset. The RCU_STAT.SWRST bit indicates that a system reset (which was triggered by software) has occurred since the last time a hardware reset occurred or since the RCU_STAT.SWRST bit was cleared by software.	
		0	Inactive
		1	Reset Occurred
2 (R/W1C)	SSRST	System Source Reset. The RCU_STAT.SSRST bit indicates that a system reset triggered by hardware in the system clock domain, clock A domain, or clock B domain has occurred since the last time a hardware reset occurred or since the RCU_STAT.SSRST bit was cleared by software.	
		0	Inactive
		1	Reset Occurred
0 (R/W1C)	HWRST	Hardware Reset. The RCU_STAT.HWRST bit indicates that a hardware reset has occurred.	
		0	Inactive
		1	Reset Occurred

Software Vector Register 0

The RCU_SVECT0 register contains the default location of the first instruction to execute after a reset.

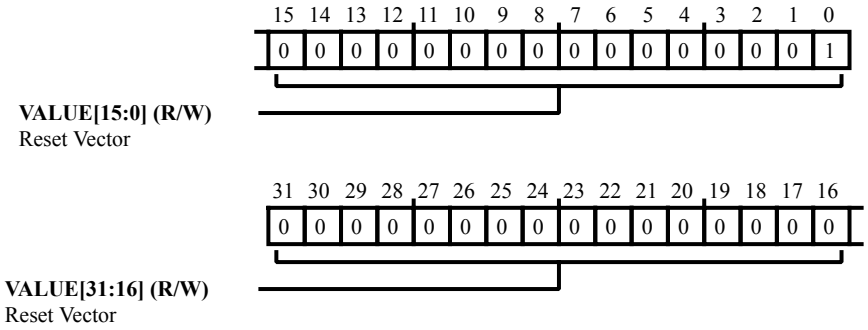

Figure 30-11: RCU_SVECT0 Register Diagram

Table 30-16: RCU_SVECT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Reset Vector.

SVECT Lock Register

The RCU software vector lock register (RCU_SVECT_LCK) provides a register lock and software vector n enable bits for each processor core on the product. This register is set to its default values by Hard Reset or any System Reset event.

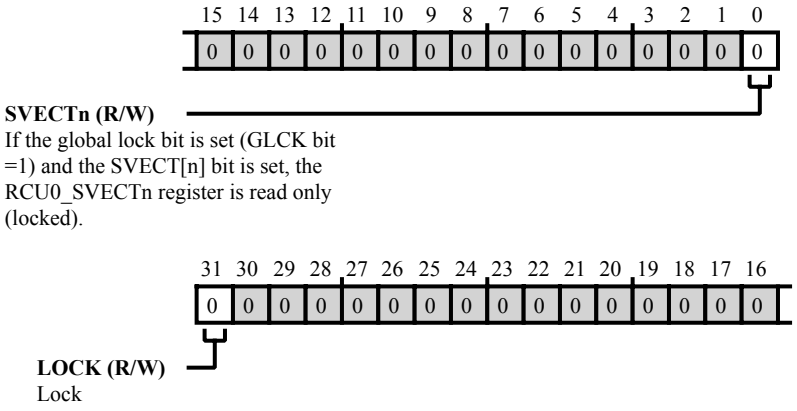

Figure 30-12: RCU_SVECT_LCK Register Diagram

Table 30-17: RCU_SVECT_LCK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the RCU_SVECT_LCK.LOCK bit is set, the RCU_SVECT_LCK register is read only (locked).	
		0	Unlock
		1	Lock
0 (R/W)	SVECTN	If the global lock bit is set (GLCK bit =1) and the SVECT[n] bit is set, the RCU0_SVECTn register is read only (locked)..	

31 Boot ROM and Booting the Processor

Bootstrapping or booting is the series of events that occur when the system applies power to the processor or when the processor enters a hardware reset state. This section gives an in-depth description of these events and how to integrate an application effectively.

On reset, the processor begins fetching instruction from an internal ROM. The boot code contained within the ROM is designed to facilitate loading an application. The boot code can automatically initialize certain peripherals for communication based on a chosen boot mode, then load an application. For more information on what boot modes are available, see the Boot Modes section. The boot code can efficiently load an entire application, code, and data, into appropriate locations after the development tools repackage the application into a boot stream.

A boot stream is an application or data that the boot-loader tool splits into blocks. A 16-byte header in each block provides instruction to the boot code for processing the associated data. The processor can perform several boot functions, depending on the flags set in the header. For more details on what options are available and a description of the stream format, refer to the Boot Loader Stream section.

In addition to supporting loading of a boot stream from a boot source to internal memory for execution, the SPI Master boot mode of the processor also supports the ability execute code from the on-chip SPI flash via the SPI2 interface without the requirement for a formatted boot stream.

NOTE: The processor supports a secure boot mechanism to protect access to the on-chip flash and secure the slave the slave boot modes. The 4KB sector 0 of the on-chip SPI flash is reserved for configuration information relating to secure boot. The boot process, by default, attempts to boot from the beginning of sector 1. If a boot stream is being used the boot stream is expected to reside at the base of sector 1. If the user is not using a boot stream and instead executing from flash then the vector table is expected to reside at the base of sector 1.

Many of the utilities of the boot code are also available to the application. These utilities include features such as copying memory, comparing memory, or loading another boot stream at run-time. The APIs may be used to help ensure that application code is more compatible with future products. For more details on available APIs, see the Callable Kernel API section.

In addition to APIs, the boot code provides the ability to define a custom boot mode. This capability helps when support is not available for a desired boot mode. It allows second stage boot loaders for non supported boot-peripherals to leverage a significant amount of the existing boot ROM functionality.

Boot Loader Stream

A loader stream is a set of formatted blocks containing instructions for the boot kernel, as well as the application and data for loading to the chip. This section details the different aspects of the stream, its blocks, some common use cases, and the ROM functionality.

Each block begins with a block header which contains attributes of the block as well as flags to control its processing by the boot ROM. On power-up or reset, the processor begins executing the on-chip boot ROM. The boot stream is either read from memory or received from a peripheral, depending on the boot mode specified. Each block in the boot stream instructs the boot kernel to perform an action, most commonly to load data to a specified location. For a complete list of actions, refer to Block Types . Some common actions include:

- running code that initializes a peripheral
- processing data then loading it to a location

As the ***Project Flow*** figure illustrates, a utility is required to process the resulting output from the tool chain to create a valid boot stream. This utility can be in the form of a standalone application or script that parses an application image file, elf output file, or text-based file such as Intel hex. It creates a valid boot stream. A flash programmer utility can format a boot stream internally.

A loader stream must always begin with a first block and end with a final block. The loader file contains the boot stream and becomes available to hardware by:

- programming or burning it into non-volatile external memory, or
- sending it through a peripheral during boot time

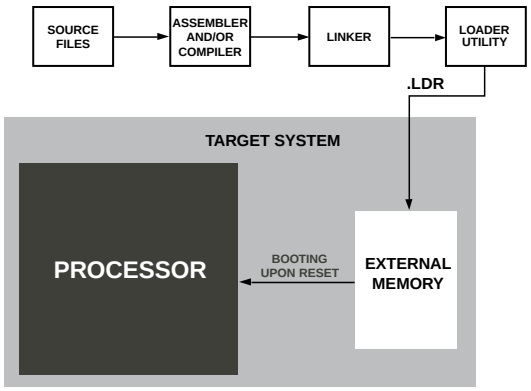

Figure 31-1: Project Flow

The ***Booting Process*** figure shows the parallel or serial boot stream contained in a flash memory device. In host boot scenarios, the non-volatile memory usually connects to the host processor rather than directly to the processor. After reset, the on-chip boot kernel reads and parses the headers and the loader stream is processed block-by-block. Finally, payload data is copied to destination addresses, either in on-chip L1 and L2 memory, or off-chip to SDRAM or SRAM.

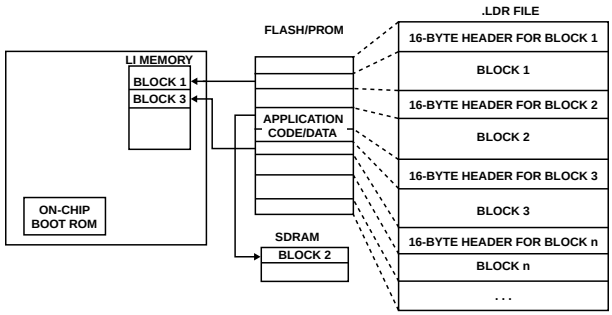

Figure 31-2: Booting Process

In some cases (for example, when the `BFLAG_INDIRECT` flag for any block is set), the boot kernel uses another memory block for intermediate data storage. To avoid conflicts, the utility used must ensure that this region is booted last.

Block Header

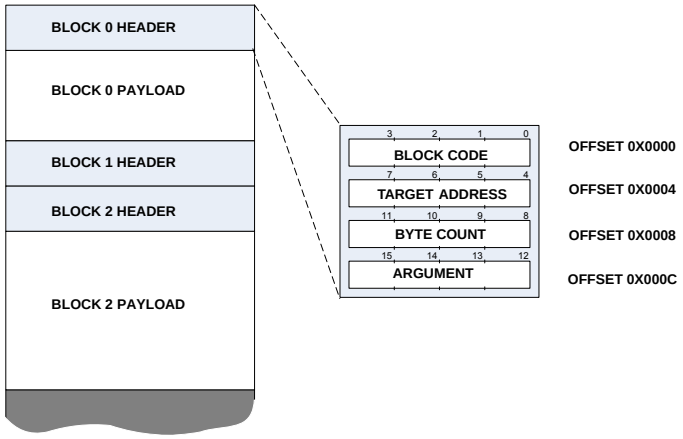

Figure 31-3: Loader Stream Block Structure

A boot stream consists of multiple boot blocks as shown in the figure. A 16-byte block header begins every block. The 16 bytes are functionally grouped into four 32-bit words:

- the block code
- the target address
- the byte count
- the argument field

This section describes the fields in general. The uses can vary depending on the particular block type and boot mode. Refer to the block type descriptions and boot modes for further information.

Block Code

Table 31-1: Block Code flags

Bit	Name	Description
0–3	BCODE	Specific to boot modes (see Boot Modes)
4	BFLAG_SAVE	Saves the memory of this block to off-chip memory in case of power failure . The on-chip boot kernel does not use this flag.
5	BFLAG_AUX	Nests special block types as required by special-purpose second-stage loaders. The on-chip boot kernel does not use this flag.
6	reserved	
8	BFLAG_FILL	Fill the target location with a specified value.
10	BFLAG_CALLBACK	Calls function at the address provided.
11	BFLAG_INIT	Calls function at target address after loading payload to the same address.
12	BFLAG_IGNORE	Block is ignored.
13	BFLAG_INDIRECT	Boots to an intermediate storage place.
14	BFLAG_FIRST	Indicates the block to be the beginning of a new application
15	BFLAG_FINAL	Indicates the last block of a loader stream. Booting will complete after processing the block. This flag does not denote the end of an application in a Multi-Application Boot Streams boot stream.
16–23	HDRCHK	A simple XOR checksum of the other 24 bits in the boot block header.
24–31	HDRSIGN	0xAD

TARGET_ADDRESS

The TARGET_ADDRESS holds the applicable address for the block, (where the code or data is loaded). However, the interpretation of the field differs depending on what specific flags are set in the block code. Refer to the documentation for each block type for details.

The following attributes must be true:

- The target address must be divisible by 4, as the boot kernel uses 32-bit DMA for certain operations.
- The target address must point to valid on-chip or off-chip memory locations.

BYTE_COUNT

The byte count must be divisible by 4, and can also be zero. This 32-bit field generally holds the size of the block. In some cases, it has a different use (such as when BFLAG_FILL is set). See the block types section for information on the variations.

ARGUMENT

The 32-bit field is a user-variable for most block types. The `Initcode` or the callback routine can access this value and use it for optional instructions.

The different block types use the `ARGUMENT` field in various ways. See the block type descriptions for further information.

Block Types

A loader stream is a set of linked blocks and each block is responsible for performing a certain function dependent on the block type. The flags in the block header define a block type. Operations include functions such as loading data, filling a memory region with data, and instructing the kernel to stop processing. This section describes each block type and its usage within a boot loader stream.

Normal Block

The primary function of a block is to load data into a specified location of memory. A normal block instructs the boot kernel to load the data contained in its payload to the location specified in the `TARGET_ADDRESS` field. The `BYTE_COUNT` defines the size of the payload. Once the correct amount of data has been loaded, the kernel moves on to process the next block in the stream.

Table 31-2: Flags

Flag	Required Value	Init
`TARGET_ADDRESS`	Y	Address where payload is loaded (must be valid)
`BYTE_COUNT`	Y	Size of block in bytes

First Block

A first block indicates the start of a boot stream and is always needed at the beginning of the boot stream. When a loader stream contains Multi-Application Boot Streams , a first block occurring within the loader stream indicates the beginning of a new application.

When the kernel processes the first block in a loader stream, the boot kernel uses the `TARGET_ADDRESS` to determine the application entry location. For more details, refer to Boot Termination and Application Execution .

NOTE: A first block cannot be combined with a fill block.

Table 31-3: Flags

Flag	Required Value	Init
`BFLAG_FIRST`	Y	1

Table 31-3: Flags (Continued)

Flag	Required Value	Init
ARGUMENT	Y	Offset to the next application, or first address following loader stream. Commonly referred to as the NEXTDXE field.
TARGET_ADDRESS	Y	When the block is the first block in a loader stream, also defines the start address for the application. If the block is not the first in a loader stream, use the target address as in normal operation.

Final Block

The final block marks the last block in a boot stream. After processing a final block, the boot kernel jumps to the start address of the application. For more information on the definition of the start address, refer to Boot Termination and Application Execution

There is further customization to the kernel behavior. For example, the kernel can be instructed to return from the boot routine rather than jump to the application using initialization codes or the Boot Routine API.

Before the boot kernel passes program control to the application, it does some housekeeping. Most of the registers in use are set to their default state. However, some register values can differ depending on the boot mode. See Boot Modes for more information.

Table 31-4: Flags

Flag	Required Value	Init
BFLAG_FINAL	Y	1

Indirect Block

An indirect block is first loaded to a storage location before being copied to the destination. The following situations motivate this functionality:

- Some boot modes do not use DMA from the boot peripheral. The core is not always able to access some memory locations directly or efficiently. An intermediate load to a different location improves overall efficiency.
- In some booting scenarios, the data in the payload must be operated-on or analyzed before it is fully loaded (such as decryption or checksum calculation). By using an intermediate location, such scenarios are simplified and can be more efficient when loading to off-chip memories (see Callback Block).

In some cases, a boot block does not fit into temporary storage memory. Having a larger buffer can improve boot performance. If an entire block cannot fit into the buffer, it is processed in pieces. Initialization code or callback functions can alter the temporary buffer region, including its location and size, by modifying the pTempBuffer and dTempByteCount variables in the ADI_ROM_BOOT_CONFIG structure.

Table 31-5: Flags

Flag	Required Value	Init
BFLAG_INDIRECT	Y	1
BFLAG_CALLBACK	N	Defines a callback function to operate on intermediate data. These 2 flags are often used together.

Ignore Block

An ignore block is a block that is (in most cases) ignored by the loader stream. Ignore blocks are useful when it is not possible to pass information in another block header. For example, if the first block contains data such as a firmware revision rather than application code, then BFLAG_IGNORE can be set with the correct application start address. This step ensures that the loader stream uses the correct start address. Since this block has no other function, identify it as an ignore block. Then, the kernel does not attempt to process any payload.

Ignore blocks result in the clearing of the following flags, disabling the corresponding blocks from being processed if set along with BFLAG_IGNORE:

- BFLAG_INIT
- BFLAG_CALLBACK
- BFLAG_FINAL
- BFLAG_AUX

NOTE: When BFLAG_IGNORE is set along with BFLAG_FIRST, only the payload associated with the first block is ignored. The application entry point retrieved from the first block is always processed.

Table 31-6: Flags

Flag	Required Value	Init
BFLAG_IGNORE	Y	1
BYTE_COUNT	Y	Size of block to ignore, can be zero

Fill Block

A fill block instructs the boot kernel to perform a 32-bit memory fill of the memory region. Fill blocks help minimize the size of a boot stream when an application contains large arrays of data that need to be initialized upon startup. Zero fill is the most common fill block type however any 32-bit fill value is supported.

Table 31-7: Flags

Flag	Required Value	Init
BFLAG_FILL	Y	1
TARGET_ADDRESS	Y	Address where payload is loaded (must be valid)
BYTE_COUNT	Y	Size of block in bytes (must be multiple of 4)
ARGUMENT	Y	32-bit Fill Value

Init Block

An initialization block instructs the boot kernel to do a function call to the target address after the entire block has loaded. The function called is referred to as the *initialization code (Initcode) routine*. If the Initcode routine has been previously loaded, the block can declare a zero-size and have no payload.

Initcode routines can be used to speed up and customize booting mechanisms exposed by the boot kernel. Traditionally, an Initcode routine is used to setup system PLL, bit rates, wait states, and the external memory controllers. If executed early in the boot process, the boot time can be significantly reduced.

Initcode routines are required to follow the C language calling conventions. The expected C prototype is:

```
void initcode( ADI_ROM_BOOT_CONFIG* pBootStruct)
```

NOTE: When programming in assembly, use a return from subroutine instruction for returns. See the Compiler manual for more information

The struct provided to the Initcode routine by the boot kernel contains various information about the block being processed. It includes header information, locations of temporary block data (for indirect blocks), target address, and byte count. See Booting Data Structures for a full list and details on the provided data.

In the simplest case, an Initcode routine consists of only a single block in which the BFLAG_INIT flag is set. For larger routines, a sequence of blocks can incrementally load the routine, and only the last block sets the BFLAG_INIT flag. In the latter case, the last block has no payload attached, and simply instructs the boot kernel to issue a call to subroutine instruction.

An Initcode routine can be overwritten by a successive block when it is no longer needed. Otherwise, the routine can be called at multiple points during the boot process, and even remain in memory after the application completes booting.

NOTE: The following list provides requirements for Initcode that is written in C or C++.

- Ensure that the Initcode routine does not contain calls to the run-time libraries
- Do not assume that parts of the run-time environment, such as the heap, are fully functional
- Ensure that all run-time components are loaded an initialized before the routine executes

Table 31-8: Flags

Flag	Required Value	Init
BFLAG_INIT	Y	1
TARGET_ADDRESS	Y	Location to load payload data. Call to subroutine issued to the same location.
ARGUMENT	N	Can be used to supply block specific arguments
BYTE_COUNT	Y	Size of payload, can be zero

Callback Block

A callback block instructs the boot kernel to call a pre-registered function upon completion of loading the payload of the block. The purpose of a callback routine is to apply standard processing to the block payload. The callback routine is registered through an Initcode routine prior to loading a block using the routine. Typically, callback routines contain checksum, decryption, decompression or hash algorithms.

To register a callback, create an ● Init Block whose Initcode modifies the pCallBackFunction pointer in the ADI_ROM_BOOT_CONFIG structure. Register a callback routine before processing a callback block.

Since callback routines require access to the payload data of the boot blocks, load the block data before processing. Often an ● Indirect Block block is used in combination with a callback block.

Callback routines are expected to meet the C language calling conventions. The prototype is as follows:

```
            s32
CallBackFunction(ADI_ROM_BOOT_CONFIG* pBootStruct,


                 ADI_ROM_BOOT_BUFFER* pCallbackStruct,


                 s32 dCbFlags)
```

The pBootStruct argument contains the ADI_ROM_BOOT_CONFIG information, and the pCallbackStruct contains the target address and size of the block (can vary when using indirect). The dCbFlags parameter is specifically used when BFLAG_INDIRECT is also used. The CBFLAG_DIRECT flag indicates that the BFLAG_INDIRECT bit is not active and so that the program only calls the callback routine once per block. When the CBFLAG_DIRECT is set, CBFLAG_FIRST and CBFLAG_FINAL are also set.

Callback Block Used in Conjunction with Indirect Block

When a block using a callback routine is also loaded indirectly, there are slight behavior differences. The procedure for loading is:

1. Load data into the temporary buffer defined by `pTempBuffer`.
2. Issue a call to the `pCallBackFunction`.
3. After the callback routine returns, if the return value is zero, the memory DMA copies data to the destination.

When a block does not fit entirely into the temporary buffer, loading is performed similar to indirect blocks. The software calls the callback function after each chunk is loaded into the temporary storage. The `dCbFlags` parameter gives information on the specific iteration.

When a block does not fit entirely into the temporary storage area, the `dCbFlags` tells the callback routine whether it is invoked for the first time (`CBFLAG_FIRST`) or called the last time (`CBFLAG_FINAL`) for a specific block.

When the software invokes DMA to copy the data, it relies on the `pCallbackStruct` structure, not the global `pTempBuffer` , and `dTempByteCount` variables. The callback routine can control the source of the memory DMA by altering the content of the `pCallbackStruct` structure. This alteration can be necessary when the callback routine performs data manipulation such as decompression.

When the software uses an indirect block, the return value of the callback routine determines whether the DMA transfer occurs. If the value is non-zero, then the transfer does not occur.

Table 31-9: Flags

Flag	Required Value	Init
`BFLAG_CALLBACK`	Y	1

Quick Boot Block

There are some booting scenarios in which not all memories are required to be reinitialized during the boot process. For example, if a processor supports wakeup from hibernate, off-chip SRAM may not need to be loaded if it is powered while the processor is in a hibernate state. Dynamic RAM is also not always impacted if it was put into a self-refresh mode before the processor powered down.

The processing of a quick boot block is determined by the state of the `BFLAG_WAKEUP`, `BFLAG_QUICKBOOT` and `BFLAG_IGNORE` flags.

Table 31-10: Quick Boot Block Processing

BFLAG_WAKEUP	BFLAG_QUICKBOOT	BFLAG_IGNORE	Block Processed
0	0	0	Yes
0	0	1	No
0	1	0	Yes

Table 31-10: Quick Boot Block Processing (Continued)

BFLAG_WAKEUP	BFLAG_QUICKBOOT	BFLAG_IGNORE	Block Processed
0	1	1	No
1	0	0	Yes
1	0	1	No
1	1	0	No
1	1	1	Yes

The BFLAG_WAKEUP flag is applied to the entire boot process when calling the boot kernel, Refer to adi_rom_Boot() for further details on the global flags available that can be applied to the boot process.

When the boot process is triggered via a hard reset or a software triggered system reset event the boot kernel is always called with BFLAG_WAKEUP cleared. Processors not supporting the wakeup from hibernate feature would generally not have this quickboot feature available for typical boot events. However, when using the boot API from the user application to boot the processor, the quickboot feature and selective processing of boot blocks is fully available regardless if wakeup from hibernate is supported or not.

NOTE: When BFLAG_WAKEUP is set and a block also has BFLAG_QUICKBOOT set, BFLAG_IGNORE flag is toggled within the boot kernel. This event occurs before the processing of an • Ignore Block and as such also has an influence on the processing of blocks that are disabled as a result of BFLAG_IGNORE being set.

NOTE: If an initcode contains an implementation that requires some operations to be performed for wakeup type events, do not set BFLAG_QUICKBOOT in the initblock, disabling it completely. Instead, ensure BFLAG_QUICKBOOT is clear and perform any conditional processing based on the state of the BFLAG_WAKEUP flag.

Table 31-11: Flags

Flag	Required Value	Init
BFLAG_QUICKBOOT	Y	1

Save Block

A save block saves the payload of the block to off-chip memory. The on-chip boot kernel does not use this flag. The flag can be used to mark blocks that an application wants for storing to non-volatile memory and restoring from non-volatile memory when the application is loaded. It provides a means of doing a context restore after a reboot.

Table 31-12: Flags

Flag	Required Value	Init
BFLAG_SAVE	Y	1

Multi-Application Boot Streams

This section describes loader streams that contain multiple applications.

A boot stream is typically generated from an application file. It is therefore common to refer to loader stream with more than one application multi-application booting. A loader utility often accepts multiple application files as input parameters and generates a contiguous boot image. The subsequent applications are appended to the first.

The loader utility must also update the Boot Loader Stream argument field of all BFLAG_FIRST blocks. The Boot Loader Stream argument field of a BFLAG_FINAL block is called the *next-application* pointer.

The next-application pointer of the first application boot stream points relatively to the start address of the second application boot stream. A multi-application boot image can be seen as a linked list of boot streams. The next-application pointer of the last application boot stream points relatively to the next free address. The ***Multi-application Boot Stream Example*** figure illustrates an example.

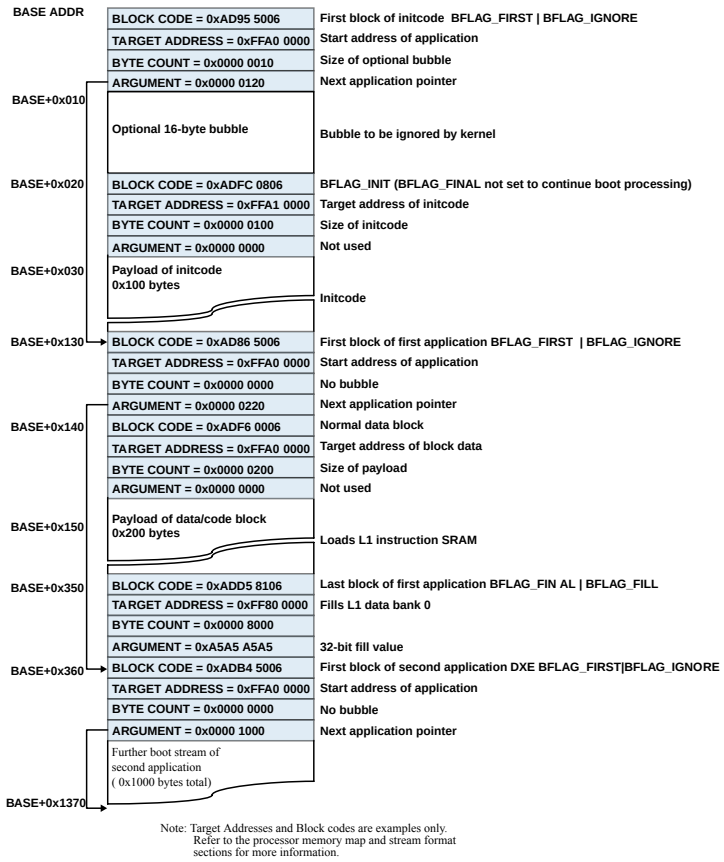

Figure 31-4: Multi-application Boot Stream Example

The ***Multi-application Direct Code Execution Example*** figure shows a linked list of initial block headers. The blocks instruct the boot kernel to terminate immediately and to start code execution at the address provided by the target address field of the individual blocks. There is nothing in the boot code that prevents multi-application streams from mixing regular boot streams and direct code execution blocks.

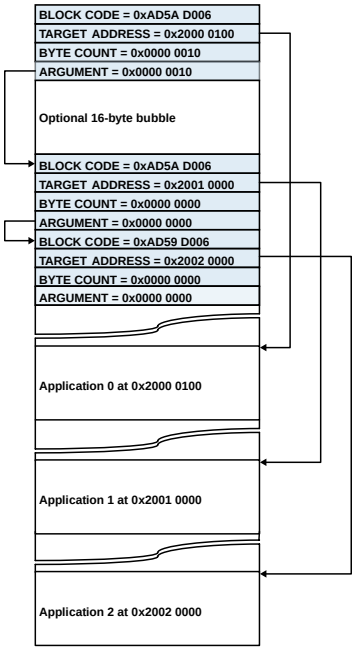

Figure 31-5: Multi-application Direct Code Execution Example

Single-Block Boot Streams

The simplest boot stream consists of a single block header and one contiguous block of instructions and possibly data. When the appropriate flags are set in the block header, the kernel loads the block to the target address, terminates the boot process, and begins executing from the entry address of the application.

The ***Initial Header for Single-Block Stream*** table shows an example of a single-block boot stream header with settings that can be loaded using any boot mode. The `BFLAG_FIRST` and `BFLAG_FINAL` flags are both set at the same time. The desired location of the applications vector table and application size is defined by the target address and byte count.

Table 31-13: Initial Header for Single-Block Stream

Field	Description of Value
BLOCK_CODE	0xAD000000|XORSUM|BFLAG_FINAL|BFLAG_FIRST
TARGET_ADDRESS	Start address of block
BYTE_COUNT	Number of bytes in the block
ARGUMENT	Functions as next-application pointer in multi-application boot streams.

Direct Code Execution

Applications can avoid long booting times and execute code from flash or SDRAM memory with minimal processing by the boot kernel. This feature is called direct code execution.

An initial boot block header is required for the processor to fetch and execute program code from the boot device as early as possible. The block uses safety mechanisms of the block to avoid unpredictable processor behavior when boot memory is not yet programmed with valid data. Safety mechanisms include the header signature and the bytewise XOR checksum. Rather than blindly executing code, the boot kernel first executes the pre-boot routine for system customization. It then loads the first block header and checks it for consistency. If the block header is corrupt, the boot kernel goes into a safe idle state and does not start code execution.

If the initial block header check is good, the boot kernel interrogates the block flags. If the block has the `BFLAG_FINAL` flag set, the boot kernel terminates and executes the sequence as described in the Boot Termination and Application Execution section. To cause the boot kernel to customize the starting address in advance, the first block must also have the `BFLAG_FIRST` flag set. The target address field is then saved as the application start address.

NOTE: When executing from SPI2 flash memory, no boot block header is required if a valid vector is found at the boot address. Refer to the SPI Master Boot Mode.

Table 31-14: Example Direct Code Execution Header

Field	Value	Comments
BLOCK_CODE	0xAD7BD006	0xAD000000|XORSUM|BFLAG_FINAL|BFLAG_FIRST|BFLAG_IGNORE|BCODE
TARGET_ADDRESS	0x20000020	Start address of application code. Provided as an example this would be dependent upon the boot start address.
BYTE_COUNT	0x00000010	Ignores 16 bytes to provide space for control data such as version code and build data. This field is optional and can be zero.
ARGUMENT	0x00000010	Functions as next-application pointer in multi-application boot streams.

CRC32 Protection

This section describes the CRC32 protection provisions.

The boot kernel provides mechanisms to allow verification of each block using a 32-bit CRC. To enable this feature, use the kernel Initcode routine, CRC Initcode. Create an Init Block with the address of this function as the `TARGET_ADDRESS`, and the `ARGUMENT` field containing the CRC32 checksum polynomial. Once this function is called (the function is called when the Initblock is processed), CRC verification is enabled for all blocks except:

- Ignore
- First

NOTE: The ROM provides a CRC Initcode routine, refer the API Initcode section of CRC Initcode. An Init-block pointing to the address of this function, as well as a zero-byte count can be used in the boot stream to instruct the boot process to execute this routine.

SRAM Requirements

The boot process requires some L1 SRAM resources for stack usage. The stack pointer used by the boot rom when exiting from a reset state is 0x2005FFFC. All the resources used by the boot process are created on the stack as and when required. It is recommended that users reserve 0xC00 bytes at the top of the L2 data SRAM space for the stack use and thus not boot content to this region of memory when booting a boot stream. Once control is passed to the user application, the stack pointer may be reset to re-use the stack space used during the boot porcess.

When executing directly from the on-chip flash instead of booting a boot stream the application in flash can immeidately reset the stack pointer to the top of the L2 SRAM and reuse the stack space that was used during the execution of the boot code.

Preboot

Upon a hard reset event or a software controlled system reset, the ROM preboot routine is executed by default. The preboot routine is responsible for inspecting the state of the processor and taking appropriate action. These actions include inspecting the state of the `SYS_BMODEn` pins via the `RCU` to determine the required boot mode and initial configuration of the processor that is required before the actual user application can be booted.

The flow through the preboot phase of the boot process is influenced by the contents of the `TAPC_RCMSG` register. This register contains a number of flags that allow for conditional processing of a number of the preboot operations. This is useful for debug sessions when debug tools may need to bypass certain operations of the preboot phase.

NOTE: Please note that upon power on reset, hard reset and software triggered system reset events the processor is running by default in PLL Bypass mode. The entire boot sequence runs with the PLL in this configuration until user software reconfigures the clocking requirements.

The sequence of operations performed during preboot is:

1. Soft Vector Processing
2. Sample boot mode pin state
3. Stack space memory initialization
4. Configure SPI2 peripheral instance and SPI flash memory
5. Perform a mass erase of the SPI flash if requested via the `TAPC_RCMSG` register
6. Process the security header in SPI flash and load required secure debug keys to the `TAPC`
7. If No-Boot mode was marked as disabled in the security header set the boot mode to SPI2 master boot

8. Initialize RCU_SVECT0 with the default user vector table address
9. Fault configuration
10. Memory initialization
11. Cache configuration
12. Memory initialization synchronization
13. Installation of memory faults
14. No-Boot processing
15. Deassert $\overline{\text{SYS\_RESOUT}}$
16. Execute user application if boot kernel bypass was requested via TAPC_RCMSG register
17. Secure header boot mode disable processing
18. Call the requested boot mode

Soft Vector Processing

The first operation of the boot software whenever a hard reset, software triggered system reset or core reset occurs, is to execute the reset vector in the boot rom. The reset vector contains software to implement a more flexible software vectoring scheme from a reset vector that cannot be altered due to the ROM placement of the reset vector. The implementation allows for core only resets being able to vector to a user defined location after a core only reset.

The soft vectoring implementation checks the contents of the RCU_SVECT0 register. If the contents are equal to the default reset value of 0x00000001 then the boot process continues with the execution of the preboot process. If the value is not equal to 0x00000001 then a BLX instruction is executed to the address that was contained in the register. Refer to the RCU chapter to understand what reset sources result in the default reset setting being applied to the RCU_SVECT0 register.

Sampling Boot Mode Pin State

The boot mode selected by the user is determined by the boot software via the RCU_STAT.BMODE field. The boot process saves the sampled value locally for then determining the boot mode to call at the end of the preboot process.

Users are required to ensure that the boot mode pins are at the desired stable state prior to releasing the processor from reset. If the boot mode strapping changed after the initial sampling of this register takes place the new settings will not take effect to the boot mode called by the preboot software.

Stack Space Memory Initialization

The boot software initializes a region of the stack space before performing any accesses to the stack in order to avoid generation of parity errors.

1024 bytes of initial stack space from address 0x2005FC00 through 0x2005FFFF is initialized by core write operations. This allows the boot software to make use of the stack until the rest of the L1 code and data space is initialized later in the boot cycle.

NOTE: This stack space initialization is always performed and is not conditional based on the `TAPC_RCMSG.NOMEMINIT` bit unlike the later stages of memory initialization.

SPI2 Peripheral Configuration

The processor contains on-chip SPI flash interfaced via the `SPI2` peripheral. This on-chip SPI flash is used to enable the security functionality of the processor. In order to gain access to the security relevant information the `SPI2` peripheral is configured to allow access to the flash.

The `SPI2` peripheral is configured to perform the following operations at an SPI clock dividier value of 0x5. The first step in the process is to reset the flash device this is poerformed by executing the following steps:

1. Issue the `CONTINUOUS MODE RESET` command (0xFFFF) to the SPI flash.
2. Issue the `DEEP POWERDOWN RELEASE` command (0xAB)

Next the device ID of the flash is read in order to confirm that communication with the flash device is functional after the reset sequence. Should the device ID not be read back correctly, the boot process terminates and enters an endless loop in the boot rom error handler.

Finally the boot code then configures the SPI flash for a mode of operation that is optimal when the processors SPI interface is operational at the maximum supported frequency. This involves the following operations:

1. Enable quad mode of operation in the SPI flash if not already enabled
2. Configure the variable read latency of the SPI flash to a value of 8
3. Configure the `SPI2` controller for memory mapped mode of operation. This allows for the core to read directly from the flash, execution of code from the flash as well as using the `MDMA` channels to read from the flash as it is now memory mapped and the SPI peripheral handles all the SPI read operations automatically.

The SPI is configured as follows for the memory mapped mode:

Table 31-15: SPI2 Peripheral Memory Mapped Configuration

Register	Value	Description
`SPI2_MMRDH`	0x05F05BEB	Single pin for command cycles, quad for address and data reads. Merge enabled Tristate after 4 bits, allowing for the mode to be issued after the address cycles Issue mode byte of 0xF0 (continuous read mode disabled) 5 Dummy Bytes. Results in 10 dummy cycles in quad mode, 2 cycles for the mode field and 8 as required by the configured latency Use 4 pins for the address cycles

Table 31-15: SPI2 Peripheral Memory Mapped Configuration (Continued)

Register	Value	Description
		3 address bytes required Use the 0xEB read command (QuadIO).
SPI2_TXCTL	0x00000005	Transmitter enabled with Transmit Transfer Initiate enabled
SPI2_RXCTL	0x00000005	Recevier enabled with Receive Transfer Initiate enabled
SPI2_CTL	0x80240043	Memory Mapped Mode enabled 4 pins to be used for transfers (excluding the command) Hardware controlled slave select Master Mode enabled SPI enabled
SPI2_MMTOP	0x18200000	The maximum addressable SPI memory range
SPI2_CLK	0x00000001	SPI clock frequency equal SCLK/2
SPI2_SLVSEL	0x0000FE02	$\overline{\text{SSEL1}}$ enabled

The SPI flash and the associated controller is now configured and ready to use for accesses to the SPI flash memory.

Flash Mass Erase Request via TAPC_RCMSG

The boot code performs a read of the TAPC_RCMSG register in order to determine if a request to erase the flash is pending. If the TAPC_RCMSG.ERASE_KEY field is set to 0xAD and the TAPC_RCMSG.ERASE bit is set then the boot process will perform a mass erase of the SPI flash. The mass erase is performed in 4KB sectors starting from the highest sector in the flash down to sector 0. This is to ensure that the secure header located at physical address 0x00000000 in the SPI flash remains intact untill all other sectors are erased.

Security Header Processing & Secure Debug Key Loading

The secure header contains keys and some boot customization options for the later stages of the boot process. Before the user can gain access to the part via the debug tools the boot software is required to load the secure debug keys provisioned by the user to the debug unit for comparison against the keys scanned in by the debug tools.

The boot software performs the following operations for reading the secure header from the flash to internal SRAM for verification

1. Read the device ID from the flash to ensure the ID is what is expected by the boot rom
2. Read the secure header from the flash into the SRAM
3. Read the device ID of the flash again to ensure the ID is what is expected by the boot rom

If the flash device ID read in steps 1 or 3 above does not return the expected result the boot rom enters an endless loop in the error handler.

Once the secure header is in SRAM and the ID checks of the flash were successful. The secure header is verified to determine if the contents are all 0xFF. This all 0xFF state indicates a blank flash or that custom security is not required. If this is the case the default secure debug keys are loaded to the TAPC to allow user debug.

If the secure header is not all 0xFF then the following operations are performed:

1. Perform ECC validation of all the ECC protected fields in the secure header
2. Perform validation of the secure header signatures to ensure they are of the required values

Assuming no errors are encountered in the ECC or the signature verification the secure debug key as defined in the secure header is then loaded to the TAPC to allow user debug. If the secure debug key is all 0x00 or all 0xFF then the key is not loaded to the TAPC at all disabling debug access until the user software writes a key for comparison.

Should an error occur in the verification of the ECC or the signature and the boot mode was detected as SPI Master Boot or No Boot then an error handler is entered that continually polls on the TAPC_RCMSG register to allow a Mass Erase request to be performed.

If an error occurred in the verification of the ECC or the signature and the device was configured for a slave boot mode, the boot process continues to allow for a mass erase operation to be issued via the slave boot mode.

Boot Mode 0 Disable

The secure header allows for the disabling of various boot modes. If boot mode 0 is found to be disabled via the secure header then the boot software will continue to boot from the SPI flash instead of remaining in the boot rom.

Default User Vector Table Address

The boot process upon completion uses the value stored in the RCU_SVECT0 register as the expected location of the cores vector table. At the end of the boot process the boot software reads the entry point fo the application from the reset vector in the vector table. By default the boot software sets the contents of the RCU_SVECT0 register to 0x10000000. This is the base address of the L1 instruction SRAM. The boot process itself can request a change to the address stored int his register to allow for flexible relocation of the vector table to location other than the base of the L1 memory.

NOTE: The initialization of the RCU_SVECT0 register is enabled by default and may be optionally bypassed by setting the TAPC_RCMSG.NOVECTINIT bit.

Fault Configuration

The following faults are enabled via the SEC. Please note that only the faults are enabled, the boot process does not install any SEC interrupts.

Table 31-16: Boot ROM Enabled Faults

SEC Fault ID	SEC Fault Name	Description
5	INTR_M4P_BUS_FAULT	Bus Fault

Table 31-16: Boot ROM Enabled Faults (Continued)

SEC Fault ID	SEC Fault Name	Description
6	INTR_M4P_LOCKUP	Lockup Error
8	INTR_WDOG0_EXP	WDOG0 Expire
9	INTR_SEC_ERR	SEC Error
46	INTR_MDMA0_SRC_CRC0_IN_ERR	MDMA0 Source DMA Channel Error
47	INTR_MDMA0_DST_CRC0_OUT_ERR	MDMA0 Destination DMA Channel Error
101	INTR_CRC0_ERR	CRC0 Error
117	INTR_SOFT3	Software Driven Interrupt 3. This is raised in the boot rom error handler.

NOTE: The installation of the fault sources is enabled by default and may be optionally bypassed by setting the TAPC_RCMSG.NOFAULTS bit. Faults are also disabled when the No-Boot boot mode is enabled.

Memory Initialization

The remainder of the L1 instruction and data SRAM are initialized to ensure that the entire parity protected L1 memories can then be read without causing a parity error.

MDMA channel 0 and the CRC peripheral configured for fill mode of operation are used to initialize the L1 instruction memory from address 0x10000000 though 0x1002FFFF. Once the DMA transfer has started the core is then used to simultaneously initialize the remainder of the L1 data memory from address 0x20030000 through 0x2005FBFF.

NOTE: The memory initialization is enabled by default and may be optionally bypassed by setting the TAPC_RCMSG.NOMEMINIT bit.

Cache Configuration

The L1 code cache is configured in order to help accelerate the accesses to the SPI flash, particularly for code execution. The default configuration enabled during preboot sets the M4P0_CACHE_CFG register to 0x0000FC10.

A cache clear request is submitted as a result of this register write operation. The boot software waits until the cache clear operation completes before progressing.

NOTE: The cache configuration is enabled by default and may be optionally bypassed by setting the TAPC_RCMSG.NOCACHE bit.

Memory Initialization Synchronization

Before continuing further with the preboot phase of the boot process the device waits for the MDMA0 channel, which was enabled earlier in the preboot cycle, to complete memory initialization. Once the MDMA operation completes, RCU_MSG.C0L1INIT is set to indicate that the entire L1 memory has been initialized by the boot software.

NOTE: The memory initialization synchronization is enabled by default and may be optionally bypassed by setting the TAPC_RCMSG.NOMEMINIT bit.

Memory Fault Installation

The memory faults are enabled separately after the memory initialization process has completed. The following faults are enabled.

Table 31-17: Boot ROM Enabled Memory Faults

SEC Fault ID	SEC Fault Name	Description
3	INTR_M4P_CORE_SRAM_PERR	SRAM Core Parity Error
4	INTR_M4P_DMA_SRAM_PERR	SRAM DMA Parity Error
7	INTR_M4P_SRAM_PERR_FLT	SRAM Parity Error

NOTE: The installation of the memory fault sources is enabled by default and may be optionally bypassed by setting the TAPC_RCMSG.NOFAULTS bit. Memory faults are also disabled when the No-Boot boot mode is enabled.

No-Boot Processing

No-Boot mode is executed on supported devices when selected by the SYS_BMODE pins. The boot mode is intended as a recovery boot mode or for debug purposes. The core simply executes an endless WFI instruction, thus terminating further execution of the boot process. This allows for a debugger to safely connect to the device and take control assuming the debugger has been granted access rights as defined by the processor security implementation when applicable.

NOTE: Boot mode 0 processing is usually only entered as a result of the boot mode pin sampling resulting in execution of the No-Boot boot mode. This processing can also be optionally enabled by setting the TAPC_RCMSG.NOBOOT bit.

$\overline{\text{SYS\_RESOUT}}$ Deassertion

In order to signal to the system that the processor is now initialized and ready to boot, the boot software deasserts the $\overline{\text{SYS\_RESOUT}}$ pin via RCU_CTL.RSTOUTDSRT.

Bypassing the Boot Kernel

By default, for boot modes other than No-Boot, the boot kernel is called to perform the actual boot of the user application. It is possible to disable the actual call to the boot kernel and instead call a user application that is located in accessible memory.

Bypassing of the boot kernel can be a useful feature for debuggers to allow for execution of the preboot process after loading an application to memory. It would typically be used along with other features such as disabling of memory initialization. The entry address of the application is retrieved from the reset vector of the vector table whose base address must be located in the RCU_SVECT0 register.

Bypassing of the boot kernel is performed by setting the TAPC_RCMSG.NOKERNEL bit. When this bit is set the following operations are performed:

1. Retrieve the base address of the vector table from RCU_SVECT0
2. Retrieve the stack pointer from the first entry in the vector table
3. Verify that the stack pointer is greater than 0x10000000 and less than or equal to 0x2005FFFF. If this operation fails the error handler is called and the boot process terminates
4. Read the application entry point from the second entry in the user vector table
5. Set the LSB of the address to ensure thumb mode
6. Call the user application. The call is performed using a BLX instruction to the application entry point

NOTE: The calling of the boot kernel is enabled by default and may be optionally bypassed by setting the TAPC_RCMSG.NOKERNEL bit. This should only ever be set when software to be executed already resides in the memory.

Disabling of Boot Modes

Prior to calling the boot kernel for the requested boot mode. The security implementation allows for disabling of specific boot modes via the secure header located in the on-chip flash. This allows only for the disabling of the slave boot modes, it is not possible to disable the SPI master boot from the on-chip SPI flash.

The boot mode disable processing is only executed if the secure header was fully verified. Refer to the secure header description for details of the bits used to disable the boot modes.

Calling the Boot Kernel

The boot kernel is called using the single adi_rom_Boot() API.

```
void * adi_rom_Boot( void *pBootAddress, uint32_t dFlags, int32_t dBlockCount,
ROM_BOOT_HOOK_FUNC *pCallHook, uint32_t dBootCommand);
```

The default parameters passed for each boot mode setting are as follows:

Table 31-18: Default Boot ROM API Parameters

Boot Mode	pBootAddress	dFlags	dBlockCount	pCallHook	dBootCommand
0	0x00000000	0x00000000	0x00000000	0x00000359	0x00000000
1	0x18001000	0x00040000	0x00000000	0x00000359	0x11610267
2	0x00000000	0x00040000	0x00000000	0x00000359	0x00000012
3	0x00000000	0x00040000	0x00000000	0x00000359	0x00000013

For details of each boot mode specific `dBootCommand` configuration Refer to the corresponding Boot Modes documentation.

In the case of the boot mode 1 `dBootCommand` value which corresponds to the SPI2 master boot mode. The critical setting is that the boot commands `BCMD_NOAUTO` and `BCMD_NOCFG` flags are set. This prevents the boot mode drivers from reconfiguring the `SPI2` peripheral which would be an unnecessary operation as the peripheral is already configured for access to the on-chip SPI flash.

The `pCallHook` parameter points to an empty function in the boot rom as there are no additional configurations to be performed that are not supported already by the boot mode specific drivers.

NOTE: The `pBootAddress` parameter of boot mode 1 can be altered from the default described by a field in the secure header that is read from the flash.

Boot Modes

The boot kernel provides various built-in support for booting from various peripherals. The ***Boot Modes*** table describes the various boot modes.

In *slave* boot modes, the processor functions as a slave to any host device. In these modes, the host device usually controls the processor `SYS_HWRST` input. Typically, the host applies the reset sequence and waits until the processor is ready to boot, depending on the peripheral in use, and transmits the boot stream data to the processor. Handshake signals are most likely used to signal to the host that the processor is ready to accept more data.

In a *master* boot mode, the processor controls the peripheral and indicates to the peripheral when to transmit the boot stream data.

Table 31-19: Booting Modes

BMODE[1:0]	Boot Source	Description
00	No Boot	The processor does not boot. Rather the boot kernel executes some of the preboot operations then enters and endless `WFI` state.
01	SPI Master Boot	Boot from integrated Flash memory via SPI2. Supports both direct execution from flash or loading of a boot stream.
10	SPI Slave Boot	Boot through the SPI0 peripheral configured as a slave
11	UART Boot	Boot through UART0 peripheral configured as a slave

NOTE: The ADSP-CM40x processor offers security functionality and these features are tightly integrated with various booting modes. Refer to Preboot and the Secure Boot sections for more details to ensure proper operation of various boot modes.

No-Boot Mode

No-boot mode is intended for device-recovery purposes caused by incorrect programming of the boot source memory allowing for target connection through an emulator.

This boot mode is not a slave boot mode, it simply places the processor in a known-safe state to allow access through emulation. The boot code performs the following preboot actions in this boot mode.

1. Soft Vector Processing
2. Sample boot mode pin state
3. Stack space memory initialization
4. Configure SPI2 peripheral instance and SPI flash memory
5. Perform a mass erase of the SPI flash if requested via the TAPC_RCMSG register
6. Process the security header in SPI flash and load required secure debug keys to the TAPC
7. If No-Boot mode was marked as disabled in the security header set the boot mode to SPI2 master boot
8. Initialize RCU_SVECT0 with the default user vector table address
9. Memory initialization
10. Cache configuration
11. Memory initialization synchronization
12. Installation of memory faults
13. No-Boot processing

The boot mode does not perform the following actions:

- Fault configuration
- Deassert $\overline{\text{SYS\_RESOUT}}$
- Invoke the boot kernel.

In the event of the security header in the SPI flash being invalid, the processor will not execute preboot steps 7 and onwards from the above list of preboot operations. Instead the boot software enters an endless loop allowing for a mass operation to be triggered via a write to the TAPC_RCMSG register. This operation may be performed by the debug tools.

NOTE: If boot mode 0 was found to be disabled via the secure boot header the boot kernel is invoked for SPI master boot mode

SPI Master Boot Mode

The SPI master boot routine provides support for booting the processor from SPI flash memories.

This SPI master boot mode boots from SPI flash memory connected to the SPI interface. The SPI supports 8-bit, 16-bit, 24-bit, and 32-bit addressable flash devices. Use of the device auto-detection feature is typically enabled by default allowing for the boot stream itself to instruct updates to the SPI configuration and the read command used allowing for more efficient transactions.

The processor supports booting from the on-chip flash only from a hardware or system based reset event. Booting from external flash devices via another SPI peripheral instance is only supported via the Boot Routine API.

Boot From on-chip SPI Flash device

The SPI peripheral is configured for memory mapped mode of operation allowing for improved boot performance as the SPI peripheral hardware handles the transmission of the read command, address and dummy cycle operations required in order to perform a read operation from the flash. The SPI peripheral also fully control the slave select signal assertion. Once the peripheral is configured for memory mapped mode, flash contents may be read by simple core read operations or by using the Memory DMA (MDMA) channels. Memory mapped mode also provides the ability to execute code directly from the flash with the assistance of cached accesses for improved performance.

When the boot mode is called, a boot address is supplied as an input parameter. By default the processor will boot from memory mapped address 0x18001000, corresponding to absolute address 0x00001000 in the flash.

NOTE: The first 4KB sector of flash is reserved for storage of the Secure Header which must be located at absolute address 0x00000000 in the flash if custom security is to be enabled.

When booting from the on-chip flash the user is provided with two booting options:

- Boot an application from a formatted boot loader stream as described in Boot Loader Stream
- Execute an application directly from the flash

In order to determine if code execution or loading of a formatted boot stream is required the first 16 bytes from the boot address are read and verified to determine if they are a valid boot block header. If a valid boot block header is found then the boot process continues processing the boot stream. All block headers and payloads are fetched using MDMA1.

In the event a valid block header was not present execution from the flash is then to be expected. The first 8 bytes stored at the boot address are required to contain the first two entries of the core vector table. The first entry in the core vector table is the stack pointer and the second the entry address of the user application.

Prior to executing from the application entry point a basic validation is performed on the stack pointer. In order to execute code from the flash the stack pointer must be in the range 0x10000000 through 0x1005FFFF or 0x20000000 through 0x2005FFFF. A stack pointer outside of these ranges results in termination of the boot process. A valid stack pointer results in the base address of the vector table being copied to the `RCU_SVECT0` register and the user application executed.

NOTE: It is recommended when using the boot API to boot via SPI2, the peripheral be configured in the manner that supports code execution prior to calling the API. The user should then call the boot mode with the address required ensuring that `BFLAG_NOAUTO` and `BFLAG_NOCFG` are set to avoid the boot software re-configuring the SPI controller.

Boot From External SPI Flash Devices

The SPI boot mode supports 8- bit, 16-bit, 24-bit, and 32-bit addressable flash devices.

The SPI master boot routine uses the peripheral DMA or MDMA channels to receive the boot stream. The DMA channel used is dependent upon support for the SPI peripheral and if the boot mode was called via the API requesting usage of the memory mapped functionality or the standard peripheral based functionality.

When auto-device detection is enabled, the SPI memory is initially read using the standard 0x03 SPI read command with a reduced clocking frequency for maximum compatibility. The first nibble of the boot stream is then used to reconfigure the SPI interface and possible the SPI flash. Refer to ● SPI Device Detection Routine.

For booting, the SPI memory is connected as shown in the ***SPI Memory Connections*** figure.

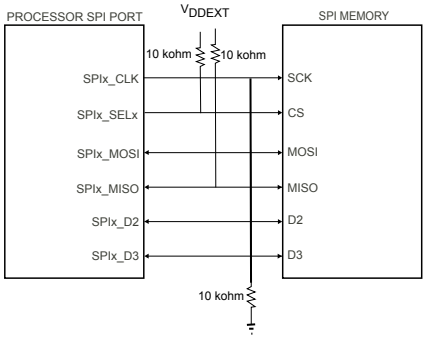

Figure 31-6: SPI Memory Connections

The pull-up resistor on the slave select signal ensures that the memory is deselected when the pin is in a high-impedance mode such as during reset.

Initialization codes are allowed to manipulate the `dBootCommand` variable in the `ADI_ROM_BOOT_CONFIG` structure to extend the boot mechanism to a second SPI memory connected to another slave select pin. Updating the field that specifies the slave select signal for use allows the boot process to manage larger boot streams than are able to fit in a single SPI device.

NOTE: If modifying the slave select signal used during the boot process, configure the pin multiplexing to enable the correct functionality for the pin. Once the boot process has proceeded past the configuration function and the boot process has actually started, the boot kernel will not perform any further pin multiplexing operations.

For SPI master boot mode, the SPI transfer size is configured for 8-bit mode and the `SPI_CTL.CPOL` and `SPI_CTL.CPHA` bits are set to zero so the SPI flash operates in SPI clock mode 0. The boot kernel does not allow SPI hardware to control the `SPI_SEL[n]` pin in the peripheral based boot mode. Instead, software toggles this pin

via the writes to SPI_SLVSEL. Initialization codes are allowed to manipulate the uwSsel variable in the ADI_ROM_BOOT_CONFIG structure to extend the boot mechanism to a second SPI memory connected to another slave select pin.

SPI Device Detection Routine

Since the boot mode supports booting from various SPI memories, the boot kernel automatically detects what type of memory is connected. To determine whether the SPI memory device requires an 8, 16, 24, or 32-bit addressing scheme, the boot kernel performs a device detection sequence prior to booting. The SPI_MISO signal requires a pull-up resistor. The routine relies on the fact that memories do not drive their data outputs unless the right number of address bytes are received.

Initially, the boot kernel transmits the read command on the SPI_MOSI line. Once the command has been sent, the boot kernel proceeds to transmit a single address byte and waits until the receive FIFO indicates that the buffer is no longer empty. The first received byte is discarded. The boot code then proceeds to issue another address byte while simultaneously receiving a byte. The process continues until a non-0xFF or 0x00 byte is received or until the full 4 address bytes is sent without any valid data being returned.

The receiving of a non-0x00 or 0xFF byte tells the boot code whether the memory device requires 8, 16, 24, 32 address bits. The lower nibble of the received byte is then used to further customize the boot mode. This nibble is referred to as the SPIMCODE. The boot code applies settings to the SPI peripheral according to the ***SPIMCODE Descriptions***.

If the received value equals 0x00 or 0xFF, it is assumed that the memory device has not driven its data output thus, another zero byte is transmitted and the received data is tested again.

If the value still equals 0xFF, device detection continues. Device detection aborts immediately when a byte different than 0xFF is received. The boot process continues with normal boot operation and it re-issues a command to re-read from address 0. Two read sequences load the first block header. Separate read sequences load further block headers and block payload fields.

The ***SPI Device Detection Principle*** figure illustrates how individual devices behave.

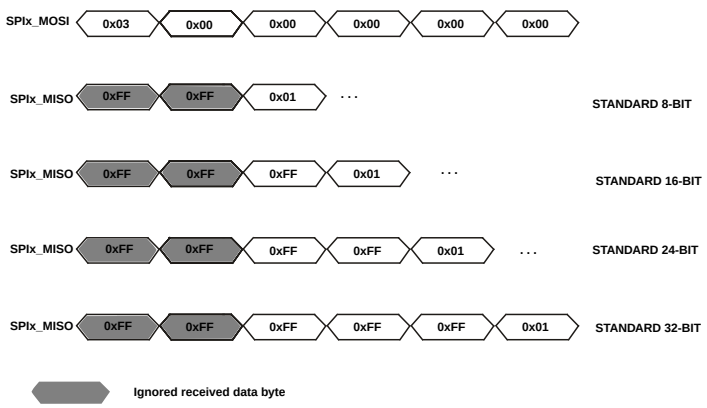

Figure 31-7: SPI Device Detection Principle

SPIMCODE Descriptions

Table 31-20: SPIMCODE Descriptions for SPI Master Memory Mapped Boot Mode

SPIMCODE	Mode	Command	Dummy Bytes	Data Lines	Address Lines	SPI Clock	Purpose
0x0	STANDARD	0x03	0	1	1	SCLK0/32	Single-bit SPI mode. Not supported for device auto-detection.
0x1	QIOR	0xEB	5	4	4	SCLK0/2	Quad data and address with `SPI_CTL.FMODE` enabled for full cycle access. This only supports devices that support quad transactions from reset and do not require enabling of quad mode support. SPI mode byte value of 0xF0.
0x2	QOR	0x6B	5	4	1	SCLK0/2	Quad data with `SPI_CTL.FMODE` enabled for full cycle access. This only supports devices that support quad transactions from reset and do not require enabling of quad mode support. SPI mode byte value of 0xF0.
0x3	DIOR	0xBB	2	2	2	SCLK0/2	Dual data and address with `SPI_CTL.FMODE` enabled for full cycle access. SPI mode byte value of 0xF0.
0x4	DOR	0x3B	2	2	1	SCLK0/2	Dual data with `SPI_CTL.FMODE` enabled for full cycle access. SPI mode byte value of 0xF0.
0x5	FAST-READ	0x0B	1	1	1	SCLK0/2	Single bit SPI mode fast read with `SPI_CTL.FMODE` enabled for full cycle access. SPI mode byte value of 0xF0.
0x6	STANDARD	0x03	0	1	1	SCLK0/4	Single-bit SPI mode.
0x7	QIOR	0xEB	5	4	4	SCLK0/2	Quad data and address with `SPI_CTL.FMODE` enabled for full cycle access. This only supports devices that support quad transactions from reset and do not require enabling of quad mode support. SPI mode byte value of 0xA5.

Table 31-20: SPIMCODE Descriptions for SPI Master Memory Mapped Boot Mode (Continued)

SPIMCODE	Mode	Command	Dummy Bytes	Data Lines	Address Lines	SPI Clock	Purpose
0x8	QOR	0x6B	1	4	1	SCLK0/2	Quad data with `SPI_CTL.FMODE` enabled for full cycle access. This only supports devices that support quad transactions from reset and do not require enabling of quad mode support. SPI mode byte value of 0xA5.
0x9	DIOR	0xBB	1	2	2	SCLK0/2	Dual data and address with `SPI_CTL.FMODE` enabled for full cycle access. SPI mode byte value of 0xA5.
0xA	DOR	0x3B	1	2	1	SCLK0/2	Dual data with `SPI_CTL.FMODE` enabled for full cycle access. SPI mode byte value of 0xA5.
0xB	FAST-READ	0x0B	1	1	1	SCLK0/2	Single bit SPI mode fast read with `SPI_CTL.FMODE` enabled for full cycle access. SPI mode byte value of 0xA5.
0xC	QIOR	0xEB	3	4	1	SCLK0/2	Quad data using quad mode enable method 2. `SPI_CTL.FMODE` is enabled for full cycle access. SPI mode byte value of 0x00.
0xD	QIOR	0xEB	3	4	4	SCLK0/2	Quad data and address using quad mode enable method 1. `SPI_CTL.FMODE` is enabled for full cycle access. SPI mode byte value of 0x00.
0xE	QIOR	0xEB	3	4	4	SCLK0/2	Quad data and address using quad mode enable method 3. `SPI_CTL.FMODE` is enabled for full cycle access. SPI mode byte value of 0x00. Not supported for device auto-detection.
0xF	FAST-READ	0x0B	0	1	1	SCLK0/32	Single bit SPI mode fast read.

NOTE: The SPI mode byte referred to is the first byte transmitted after the address cycles and is used to control the continuous read mode functionality in which the next read operation is not required to issue a command cycle. Continuous read mode is not supported during the boot process.

Table 31-21: SPIMCODE Descriptions for SPI Master Peripheral Boot Mode

SPIMCODE	Mode	Command	Dummy Bytes	Data Lines	Address Lines	SPI Clock	Purpose
0x0	STANDARD	0x03	0	1	1	SCLK0/32	Single-bit SPI mode. Not supported for device auto-detection.
0x1	STANDARD	0x03	0	1	1	SCLK0/32	Single-bit SPI mode
0x2	STANDARD	0x03	0	1	1	SCLK0/4	Single-bit SPI mode
0x3	FAST READ	0x0B	1	1	1	SCLK0/2	Single bit SPI mode fast read
0x4	FAST READ	0x0B	1	1	1	SCLK0/2	Single bit SPI mode fast read with `SPI_CTL.FMODE` enabled for full cycle access.
0x5	STANDARD	0x03	0	1	1	SCLK0/3	Single bit SPI mode with`SPI_CTL.FMODE` is enabled for full cycle access.
0x6	FAST READ	0x0B	1	2	1	SCLK0/1	Single bit SPI mode fast read with `SPI_CTL.FMODE` enabled for full cycle access.
0x7	RAPID-S	0x1B	2	2	2	SCLK0/1	Single bit RAPID-S read `SPI_CTL.FMODE` is enabled for full cycle access.
0x8	DOR	0x3B	1	2	1	SCLK0/2	Dual bit data. `SPI_CTL.FMODE` is enabled for full cycle access.
0x9	DIOR	0xBB	1	2	2	SCLK0/2	Dual data and address. `SPI_CTL.FMODE` is enabled for full cycle access.
0xA	QOR READ	0x6B	1	4	1	SCLK0/2	Quad bit data mode using quad enable method 1 with `SPI_CTL.FMODE` is enabled for full cycle access.
0xB	QIOR READ	0xEB	3	4	4	SCLK0/2	Quad data and address using quad enable method 1 with `SPI_CTL.FMODE` is enabled for full cycle access.
0xC	QOR READ	0x6B	1	4	1	SCLK0/2	Quad data using quad mode enable method 2. `SPI_CTL.FMODE` is enabled for full cycle access
0xD	QIOR READ	0xEB	3	4	4	SCLK0/2	Quad data and address using quad mode enable method 2. `SPI_CTL.FMODE` is enabled for full cycle access

Table 31-21: SPIMCODE Descriptions for SPI Master Peripheral Boot Mode (Continued)

SPIMCODE	Mode	Command	Dummy Bytes	Data Lines	Address Lines	SPI Clock	Purpose
0xE	QIOR READ	0xEB	3	4	4	SCLK0/2	Quad data and address using quad mode enable method 3. `SPI_CTL.FMODE` is enabled for full cycle access
0xF	FAST-READ	0x0B	0	1	1	SCLK0/32	Single bit SPI mode fast read. Not supported for device auto-detection.

NOTE: The SPI mode byte issued for all the SPI Master peripheral based configurations is 0x00

Quad Mode Enable Method 1

1. Issue the Read Status Register command (0x05) and read in the value of the status register
2. Issue the Read Configuration Register command (0x3F) and read in the configurations register value
3. Issue the Write Enable command (0x06)
4. Issue the Read Status Register command (0x05) to verify the write latch status is set
5. Set bit 7 of the read configuration register value to enable quad mode
6. Issue the Write Register command (0x3E) and write the value of the configuration register
7. Issue the Read Status Register command (0x05) until the device is ready and write latch is cleared
8. Issue the Read Configuration Register command (0x3F) to verify quad mode is enabled

Quad Mode Enable Method 2

1. Issue the Read Status Register command (0x05) and read in the value of the status register
2. Issue the Read Configuration Register command (0x35) and read in the value of the configuration register
3. Issue the Write Enable command (0x06)
4. Issue the Read Status Register command (0x05) to verify the write latch status is set
5. Set bit 1 of the read configuration register value to enable quad mode
6. Issue the Write Register command (0x01) and write the value of the status register and the updated value of the configuration register
7. Issue the Read Status Register command (0x05) until the device is ready

Quad Mode Enable Method 3

1. Issue the Read Status Register command (0x05) and read in the value of the status register
2. Issue the Write Enable command (0x06)
3. Issue the Read Status Register command (0x05) to verify the write latch status is set
4. Set bit 6 of the read status register value to enable quad mode
5. Issue the Write Status command (0x01) and write the value of the status register back to enable quad mode
6. Issue the Read Status Register command (0x05) until the device is ready

Run-time API

The following table provides descriptions of the `adi_rom_Boot()` `dBootCommand` parameter.

Table 31-22: dBootCommand for SPI Master Boot Bit Descriptions

Bit No. (Access)	Bit Name	Description/Enumeration	
31:28	ROM_BCMD_SPIM_SPEED	SPI clock divider to be used. The value written to the SPI Peripherals clock divider register.	
26:25	ROM_BCMD_SPIM_IOPROT	SPI I/O protocol. If multiple I/O pins are required, then I/O mode is enabled on the flash device. The number of data bits required for the data transfer determines the I/O mode enabled. This field is only applied if NOAUTO is set to 1. Otherwise, the protocol for enabling the additional pins is defined through the BCODE value supplied.	
		0x0	No protocol required to implement multiple I/O
		0x1	Enable quad functionality using protocol 1
		0x2	Enable quad functionality using protocol 2
		0x3	Enable quad functionality using protocol 3
24:22	ROM_BCMD_SPIM_DUMMY	Number of dummy bytes to be issued. Specifies the number of dummy bytes to be issued after the address bytes are issued for the required read command.	
		0x0	Do not issue dummy bytes
		0x1	Issue 1 dummy byte
		0x2	Issue 2 dummy bytes
		0x3	Issue 3 dummy bytes
		0x4	Issue 4 dummy bytes
		0x5	Issue 5 dummy bytes
		0x6	Issue 6 dummy bytes

Table 31-22: dBootCommand for SPI Master Boot Bit Descriptions (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0x7	Issue 7 dummy bytes
21:20	ROM_BCMD_SPIM_ADDR	Number of address bytes to be issued. Specifies the number of address bytes that are required to be issued to the SPI flash for the required read command.	
		0x0	Issue 1 address byte
		0x1	Issue 2 address bytes
		0x2	Issue 3 address bytes
		0x3	Issue 4 address bytes
19:16	ROM_BCMD_SPIM_BCODE	Boot mode-specific code. Specifies the boot mode-specific code that can further customize and control the boot process.	
14:12	ROM_BCMD_SPIM_SSEL	SPI Slave select signal. Specifies the SPI slave select signal to be used to enable the SPI Flash. Not all slave selects are available for each SPI port. Refer to the product data sheet for further details.	
		0x0	SEL1
		0x1	SEL2
		0x2	SEL3
		0x3	SEL4
		0x4	SEL5
		0x5	SEL6
		0x6	SEL7
		0x7	Reserved
11:8	ROM_BCMD_DEVENUM	Device enumeration. Specifies the SPI device to use.	
		0x0	SPI0
		0x1	SPI1
		0x2	SPI2
		0x3 - 0xF	Reserved
6	ROM_BCMD_NOAUTO	Automatic device detection disable. When set disables automatic device detection and uses the setting provided in the other fields of this register to configure the boot mode.	
5	ROM_BCMD_NOCFG	Device configuration disable.	

Table 31-22: dBootCommand for SPI Master Boot Bit Descriptions (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		When set, this bit disables device configuration. Device configuration includes reconfiguration of the peripherals MMR registers and device pin muxing.	
4	ROM_BCMD_ HOST	Host boot mode enable. When set, this bit indicates that SPI slave boot is to be performed. Otherwise, use the master boot mode.	
3:0	ROM_BCMD_ DEVICE	Boot source device. Specifies the device to boot from.	
		0x2	SPI
		0x7	Memory Mapped SPI

NOTE: All bits in the above table that are not defined must be set to zero.

SPI Slave Boot Mode

When using SPI slave mode boot, the processor consumes boot data from an external SPI host device. This mode supports single, dual, and quad-bit modes. The boot kernel always starts in single bit mode and can be changed using the appropriate command. The following figures show the hardware configuration for the modes. As in all slave boot modes, the host device controls the `SYS_HWRST` input of the processor.

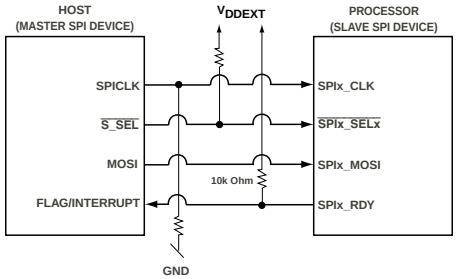

Figure 31-8: Connection Between Host (SPI Master) and Processor (SPI Slave)

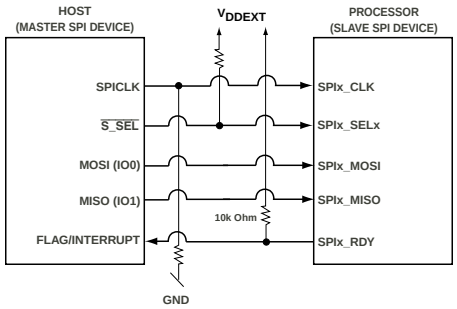

Figure 31-9: Connection Between Host (SPI Master) and Processor (SPI Slave) DIOM

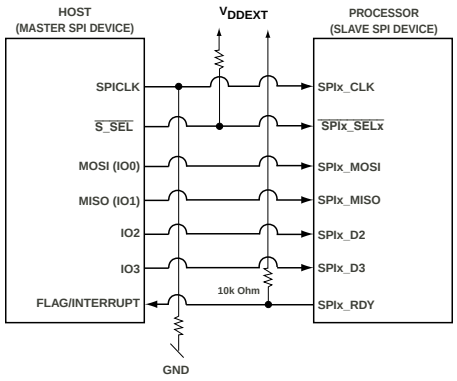

Figure 31-10: Connection Between Host (SPI Master) and Processor (SPI Slave) QSPI

The host drives the SPI clock and is responsible for timing. The host must provide an active-low chip select signal that connects to the SPIx_SS input of the processor. It can toggle with each byte transferred or remain low during the entire procedure. 8-bit data is expected and 16-bit mode is not supported.

In SPI slave boot mode, the boot kernel sets the SPI_CTL.CPHA bit and clears the SPI_CTL.CPOL bit in the SPI_CTL register. Therefore the SPI_MISO pin is latched on the falling edge of the SPI_MOSI pin.

The SPI slave processor detects the correct boot mode from the host SPI device by reading the first byte sent, defined as SPICMD. The ***SPICMD Descriptions*** table describes the available codes. These additional bytes must be sent prior to transmitting the data to configure the SPI device accordingly.

The table describes two cases, one in which the host starts in single bit mode, and one in which the host starts in a mode other than single bit.

Table 31-23: SPICMD Descriptions

SPICMD	Description
If host Starts in Single bit Mode	
0x3	Keep single-bit mode
0x7	Switch to dual-bit mode
0xB	Switch to quad-bit mode
If host device starts in DIOM or QSPI	
0xAA,0xBF	Switch to dual-bit mode
0xEE,0xEE,0xFE,0xFF	Switch to quad-bit mode

In SPI slave boot mode, SPIx_RDY functionality is critical. The SPIx_RDY output is used for back pressure and requires a pulling resistor. The boot code requires the SPIx_RDY signal function as active-low. The host is only permitted to transfer data when SPIx_RDY is in the active state. This functionality allows the processor to hold off the host while the processor is in reset or executing the pre-boot and processor initialization sequences. The SPI is configured to de-assert SPIx_RDY when the receive FIFO is filled to 75% or more. The ***SPI Program Flow on the Host Side*** figure illustrates the required program flow on the host side.

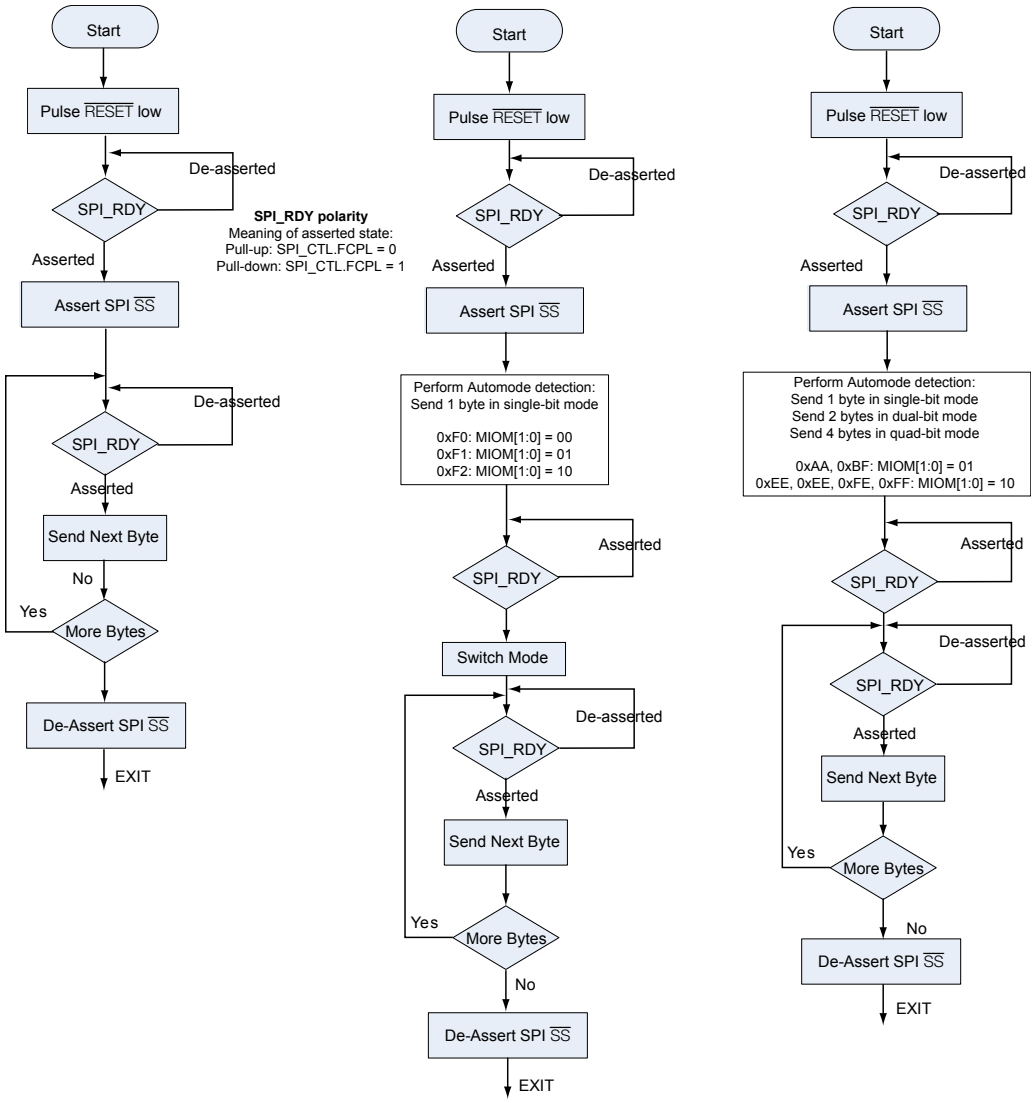

Figure 31-11: SPI Program Flow on the Host Side

Run-time API

The SPI Slave Boot mode can be called through the Boot Routine API function at run-time. Initiating a boot through the run-time API allows for extra customization such as disabling automatic device configuration or specifying a different SPI device other than the default.

When `ROM_BCMD_NOCFG` flag is specified, it is necessary to program pin multiplexing and other SPI configuration as required, while keeping the `SPI_CTL.EN` bit cleared.

The `ROM_BCMD_NOAUTO` flag can suppress auto mode detection. In that case, the desired configuration must be passed through the `ROM_BCMD_SPIS_BCODE` bit field, even if the `ROM_BCMD_NOCFG` flag is set.

The ***dBootCommand*** figure describes the fields possible for customization through the `dBootCmd` parameter.

The following table provides descriptions of the `adi_rom_Boot()` `dBootCommand` parameter.

Table 31-24: dBootCommand for SPI Slave Boot Bit Descriptions

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16	ROM_BCMD_SPIS_BCODE	Boot Mode Specific BCODE. Specifies the boot mode-specific code that can further customize and control the boot process.	
		00xxb	Single bit SPI bus
		01xxb	Dual SPI bus
		10xxb	Quad SPI bus
		11xxb	Reserved
11:8	ROM_BCMD_DEVENUM	Device enumeration. Specifies the SPI device to use.	
		0x0	SPI0
		0x1	SPI1
		0x2 - 0xF	Reserved
6	ROM_BCMD_NOAUTO	Automatic device detection disable. When set disables automatic device detection and uses the setting provided in the other fields of this register to configure the boot mode.	
5	ROM_BCMD_NOCFG	Device configuration disable. When set, this bit disables device configuration. Device configuration includes reconfiguration of the peripherals MMR registers and device pin muxing.	
4	ROM_BCMD_HOST	Host boot mode enable. When set, this bit indicates that SPI slave boot is to be performed. Otherwise, use the master boot mode.	
3:0	ROM_BCMD_DEVICE	Boot source device. Specifies the device to boot from.	
		0x2	SPI

NOTE: All bits in the above table that are not defined must be set to zero.

UART Slave Boot Mode

When using UART slave mode boot, the processor receives boot data from a UART host device connected to the UART interface. The device connected to UART0 is initially detected using an autobaud detection sequence. After finishing the UART slave boot process, all control and status registers of the used resources are restored.

Further customization, such as disabling autobaud detection, and changing the device, use the boot routine API.

During the boot operation, the host device usually relies on the `RTS` output of the UART device. At boot time, the processor does not evaluate RTS signals driven by host. Since the RTS is in a high impedance state when the processor is in reset, or while executing a pre-boot, an external pull-up resistor to `VDDEXT` is recommended. The

Connection Between Host and Processor figure shows the interconnection required for booting. The figure does not show physical line drivers and level shifters that are typically required to meet the individual UART-compatible standards.

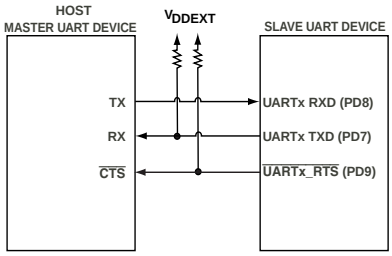

Figure 31-12: Connection Between Host and Processor

When the UART is enabled, the RTS goes immediately low, encouraging the host to send the first boot stream data as shown in the ***Host Relying on RTS*** figure. For half-duplex UART connections, the host must avoid this action. The host must wait until it has received the 4 bytes from the slave processor before sending any data.

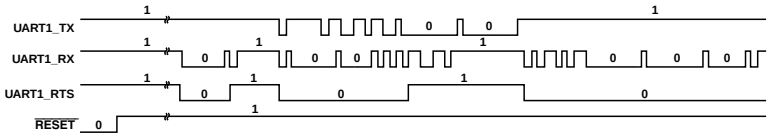

Figure 31-13: Host Relying on RTS

When the boot kernel is processing fill or Initcode blocks, it can require extra processing time and must delay the host from sending more data. This request is signaled using the RTS output.

The ***Host Relying on RTS*** figure shows RTS timing when an extended Initcode routine executes. Since code execution is distracting from the data loading, the host device must be prevented from sending more data. The timing of the RTS depends on the state of the RFRT bit in the UART control register (UART_CTL). This bit is cleared during UART slave boot mode when RTS is de-asserted, the UART receive FIFO contains 4 or more data words, and another start bit is detected.

Autobaud Detection

The kernel supports autobaud detection using the '@' character as data. The host is expected to have its clock set to a rate supported in the UART.

To determine the bit rate when performing autobaud detection, use the following steps:

1. The boot kernel expects an '@' character (0x40, eight bits data, one start bit, one stop bit, no parity bit) on the UART RXD input.
2. The EDBO and UART_CLK register is cleared.
3. The boot kernel acknowledges, and the host then downloads the boot stream. The acknowledgment consists of 4 bytes: 0xBF, UART_CLK [15:8], UART_CLK [7:0], 0x00.
4. The host is requested to not send further bytes until it has received the complete acknowledge string.

5. Once the 0x00 byte is received, the host can send the entire boot stream.

The host knows the total byte count of the boot stream, but it is not required to know the content of the boot stream.

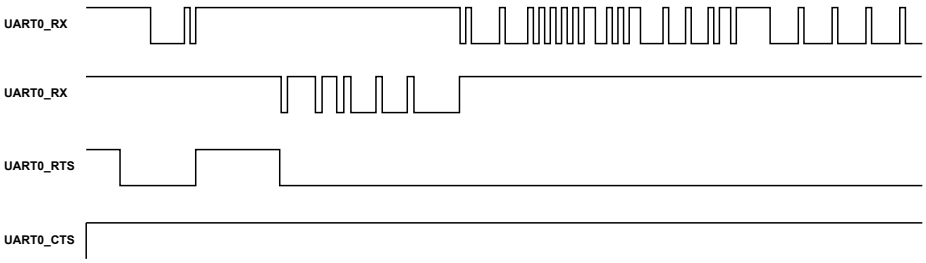

Figure 31-14: UART Autobaud Detection Waveform

The ***UART Autobaud Detection Waveform*** figure provides timing information for UART booting. After the bit rate is known, the UART is enabled and the kernel transmits the 4 acknowledge bytes.

Run-time API

The UART slave boot mode can be called through the boot routine API function at run time. The run-time API allows for more customization. Both autobaud detection and device configuration can be disabled, and a device other than the default UART0 can be specified.

If `ROM_BCMD_NOCFG` flag is specified, it is the programs responsibility to configure pin multiplexing as required.

Autobaud detection can be suppressed using the `ROM_BCMD_NOAUTO` flag. In this case, the desired configuration can be passed through the `ROM_BCMD_UART_CLK` bit field. If the `ROM_BCMD_UART_CLK` bit field is zero, `UART_CLK` is evaluated. If a value of 0xFFFF was present, the default error routine of the boot kernel is called and the booting process is aborted. Otherwise, the value in `UART_CLK` remains untouched.

The following table provides descriptions of the `adi_rom_Boot()` `dBootCommand` parameter.

Table 31-25: dBootCommand for UART Slave Boot Bit Descriptions

Bit No. (Access)	Bit Name	Description/Enumeration
31:16	ROM_BCMD_ UART_CLK	UART Clock Divider. When set to zero this field is ignored.
15	ROM_BCMD_ UART_EDBO	UART Clock Divider Mode When set enables EDBO functionality.

Table 31-25: dBootCommand for UART Slave Boot Bit Descriptions (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11:8	ROM_BCMD_ DEVENUM	Device enumeration. Specifies the SPI device to use.	
		0x0	UART0
		0x1	UART1
		0x2	UART2
		0x3 - 0xF	Reserved
6	ROM_BCMD_ NOAUTO	Automatic device detection disable. When set disables automatic device detection and uses the setting provided in the other fields of this register to configure the boot mode.	
5	ROM_BCMD_ NOCFG	Device configuration disable. When set, this bit disables device configuration. Device configuration includes reconfiguration of the peripherals MMR registers and device pin muxing.	
4	ROM_BCMD_ HOST	Host boot mode enable. When set, this bit indicates that SPI slave boot is to be performed. Otherwise, use the master boot mode.	
3:0	ROM_BCMD_ DEVICE	Boot source device. Specifies the device to boot from.	
		0x3	UART

NOTE: All bits in the above table that are not defined must be set to zero.

Security

The ADSP-CM40x series processor offers security to protect the content (Intellectual Property) located in the SoC flash memory. The functionality is user configurable to assist in the application and product development. It is an approach to help safeguard sensitive data and IP without using cryptographic functions. Although it does not provide any decryption functionality or any other features using cryptographic means, the collective features help provide a high level of integrity and confidentiality.

Security Features

Security features of the ADSP-CM40x include:

- User controlled security enable.
- Disable of debug by default upon power up or after reset, to prevent the debugger to gain access to flash contents.

- Enable of secure debug via a key entry through JTAG and SWD ports after a successful key match.
- Disable individual boot modes regardless of which boot mode is sampled via the SYS_BMODE pins, only allowing booting from the internal SPI flash memory.
- Protected slave boot modes via key.
- Flash recovery via mass erase of the flash memory.
- ECC protected Secure Block Header.

Security Functional Description

The following sections provide information on security enable, secure debug and header, secure booting and bulk erase of SoC flash.

Security Definitions

To make the best use of the security functions, it is useful to understand the following terms.

Invalid Secure Header

Header at flash memory location 0x0000_0000 (at memory location of 0x1800_0000) is invalid if the header is not in the format in the ***Secure Header Structure Definition*** table or the ECC values for different header fields are not correct.

Valid Secure Header

Header at flash memory location 0x0000_0000 (i.e. at memory location of 0x1800_0000) is a valid secure header, if the header is exactly in the format given in the ***Secure Header Structure Definition*** table. Also the ECC values should be correct for different fields of the header like key, signature, reserved, flag and boot address fields.

Enabling Security

Programs can enable or disable security features depending upon the header configured at flash memory location 0x0000_0000 (at memory location 0x1800_0000).

Various security features are active depending on the following header configurations:

- Invalid Secure header, debug keys will not be loaded to the TAPC as the content cannot be trusted thus requiring a mass erase of the flash to recover
- Flash Secure header is in erased state, default keys are loaded to the TAPC to allow debug access using the default keys. Slave boot modes are not protected at all
- Valid secure header with default keys, default keys are loaded to the TAPC to allow debug access uisng the default keys. Slave boot modes are not protected at all

By default, the SoC flash memory is blank, so the security features are not active and the default key is required to be used for debug access. If a valid Secure Header is found with a key other than the default key, security is applied according to the parameters defined in the Security Header.

Secure Header

The Secure Header is a structure defined by the user found in the flash memory at location 0x0000_0000. The boot kernel residing in the ROM handles the majority of the security functions in the processor guided by what is configured in the Secure Header. Upon booting, the boot kernel looks at the Secure Header in the flash memory to determine how security is handled. The ***Secure Header Structure Definition*** table shows the Secure Header structure.

Table 31-26: Secure Header Structure Definition

Name	Description	Size (Bytes)	SPI Memory Mapped Address
signature0	Secure boot header signature0	4	0x18000000
reserved0	Reserved	4	0x18000004
key0	Value to be loaded to TAPC.SDBGKEY0	4	0x18000008
key1	Value to be loaded to TAPC.SDBGKEY1	4	0x1800000c
key2	Value to be loaded to TAPC.SDBGKEY2	4	0x18000010
key3	Value to be loaded to TAPC.SDBGKEY3	4	0x18000014
flags	Flags to define boot mode disable and default boot mode	4	0x18000018
bootAddress	The default address to use to boot from or where the vector table is stored if executing from flash	4	0x1800001c
ecc0	ECC parity bits for signature0 and reserved0 fields	1	0x18000020
ecc1	ECC parity bits for key0 and key1 fields	1	0x18000021
ecc2	ECC parity bits for key2 and key3 fields	1	0x18000022
ecc3	ECC parity bits for flags and bootAddress fields	1	0x18000023
ecc4	ECC parity bits for signature1 and signature2 fields	1	0x18000024
reserved1	Reserved	3	0x18000025
signature1	Secure boot header signature1	4	0x18000028
signature2	Secure boot header signautre2	4	0x1800002c

Secure Debug

In order to help protect the IP stored on the flash, debug is disabled by default upon power up or after reset. This prevents just anyone from attaching a debugger and gaining access to the contents stored in the flash memory.

Default Debug Key

If the Secure Header is blank, it is assumed that security is not needed and debug is enabled by passing a Default Debug Key. The Default Debug Key also needs to be passed if the Secure Header is a valid one with Default keys programmed in the flash. The Default Debug Key is mentioned below

Default Debug Key
0x8392C1FE18839DF8D2041DD7A4B8E4A5

The boot kernel loads the Default Debug Key into the Secure Debug Key register of the TAP controller. The following Default key is loaded into the Secure Debug Key register.

```
TAPC.SDBGKEY0 = 0xA4B8E4A5;
TAPC.SDBGKEY1 = 0xD2041DD7;
TAPC.SDBGKEY2 = 0x18839DF8;
TAPC.SDBGKEY3 = 0x8392C1FE;
```

The key from the Secure Debug Key registers of the TAP controller are compared with the one passed over the debugger, the debugger is given access only if they match.

To gain access with a debugger to a device where secure header in flash is in erased state, the default key must be passed.

User Debug Key

If the flash has a valid Secure Header with custom user key, debug is enabled by passing the user key. The boot kernel copies the user key from secure header in flash to the Secure Debug Key registers of the TAP controller. The debugger is given access only if the key stored in the Secure Debug Key register match with the one passed over the debugger.

To gain access with a debugger to a device where the secure header is a valid, the user key (the same as that programmed in secure header) must be passed to the debugger. The debugger can enter the key through the JTAG port as well as the SWD. For key entry through the JTAG port, the debugger should write to the USER_KEY instruction over JTAG port and then enter the keys. For key entry through the SWD, the debugger should write the key to the USERKEYCMPx registers.

Boot Mode Control

Another security measure provides the ability to control which boot modes are allowed or disallowed via the Secure Header. If untrusted or non-secure boot modes were allowed and enabled, a malicious user may boot via one of these modes and enable entry paths to gain access to the IP in the flash memory.

When the processor first comes out of reset, the SYS_BMODE[1:0] pins are sampled and the boot mode is indicated in the RCU_STAT register. In order to help secure sensitive information or IP that is in the flash memory, the program should disable alternative slave boot modes or boot mode 0 (no boot). While these boot modes are also protected by a key entry mechanism, provision is provided to allow programs to selectively disable that part of the boot process and default to SPI master boot mode.

The FLAGS parameter in the Secure Header allows the program to set which boot modes are enabled or disabled. This overrides whichever boot mode was sampled from the SYS_BMODE[1:0] pins. In order to help protect the content in the SPI flash,the program should disable all alternative boot modes, especially slave boot modes such as UART to prevent a malicious or unintended application from being loaded.

The ***Boot Mode Disable Bits*** table describes the bits in the FLAGS parameter for disabling boot modes.

Table 31-27: Boot Mode Disable Bits

Bit	Boot Mode Disabled When Set
0	No boot/Idle
2	SPI Slave Boot
3	UART Boot

This table mimics the Boot Modes table specified for the SYS_BMODE[1:0] pin setting that can be found in the ADSP-CM40x datasheet. These bits override the SYS_BMODE[1:0] pin settings and disable that particular boot mode. The other difference is that Bit 1 and the corresponding SYS_BMODE[1:0] = 01 are different. The FLAGS parameter does not allow boot mode 1, Flash Boot to be disabled.

The bits that are not specified in the table above are reserved and have no affect on boot mode disabling.

SPI Master Boot Mode

To perform SPI master boot, the processor does not require any key to be sent. If the Secure Header is valid, the boot kernel boots the application from the location configured in the bootAddress field. The bootAddress field defines the IVT location of the application to be booted. In case of an invalid header, the boot kernel waits for the erase command to be issued and does not boot the application. If the flash header is in an erased state, the boot kernel jumps to the 0x1800_1000 location in flash to boot the application (default behavior).

Secure Slave Boot Mode

In one instance, a program may want to allow a slave boot mode in order to perform a firmware update. In another instance, the flash may have become corrupted and can't boot correctly. The key in the Secure Header is employed in this case as well as for allowing Secure Debug.

If the slave boot modes are not flagged by configuring FLAGS parameter, the boot kernel will perform the following operations depending upon the secure header.

Valid Secure Header with User Key

In case the flash has a valid Secure Header with User key, in order to perform secure slave booting the host will have to send the command 0x086A043F, followed by the User key and subsequently the loader stream.

For example, if the `SYS_BMODE[1:0]` pins were set to b'11, for UART Boot, and if bit 3 of the FLAGS parameter in the Secure Header was not set, UART booting would proceed as follows.

1. Processor: Autobaud Detect, and wait for command.
2. Host: Send Command ***0x086A043F*** (least significant byte first)
3. Host: Send Key (least significant byte of key0 first all the way to the most significant byte of key3).
4. Processor: Compare Key sent from Host to Key found in Secure Header.
    - If the keys do not match in their entirety, the boot terminates.
    - If the comparison is successful resulting in a match, the boot process continues to load the loader stream.

This also applies to the SPI Slave boot. If security is enabled, the SPI Slave boot needs to pass as described above for UART boot.

Erased Secure Header/Valid Secure Header with Default key:

If the flash Secure Header is in an erased state or the Secure Header has a Default key, security features are not applied to the device. In this case slave boot is performed in a non-secured way without the requirement for the host to send the key command and the 128-bit default key.

Invalid Secure Header

If an invalid Secure Header is present, the boot kernel does not execute the loader stream. The boot kernel waits for the flash erase command to be issued by the host. As soon as the host sends an erase command, the boot kernel erases the entire flash to remove the invalid header. For more information about the bulk erase command, refer to Bulk Erase of the SoC Flash Memory.

Mass Erase of the SoC Flash Memory

One of the safeguards that the security features on the ADSP-CM40x processor provides is the ability to perform an erase of the entire SoC flash memory. Although the SPI flash memory device supports a bulk erase command, the boot kernel instead loops through every sector starting with the last sector and erasing each one in turn. This way, the Secure Header remains intact until the very end. Otherwise, it may be possible to clear the Secure Header before the sensitive information is erased resulting in security being disabled while sensitive data or IP remain present on the flash.

NOTE: Due to the mass erase operation being performed on a sector by sector basis. The mass erase functionality for recovery can take anything up to 4 minutes to complete. Upon completion of the mass erase when performed via the one of the slave boot modes, the default error handler will be called and a fault is raised.

This fault may be used as indication that the mass erase operation has completed. In a majority of circumstances this will take significantly less than 4 minutes however host software should maintain a 4-minute time out to be certain that the erase operation has completed.

The mass erase operation can be impacted by the locked state of blocks in the SoC flash. If user software has locked down blocks then a power cycle may be required before the mass erase operation will complete as expected.

Mass erase can be performed via the following methods:

1. Erase using slave boot modes
2. Erase using the debugger

Erase Using Slave Boot Modes

The flash can be erased by sending the erase command via the host in UART and SPI slave boot modes. The erase command is issued through the peripheral interface prior to the loader stream. When the boot code recognizes the command as an "Erase Flash" command, a bulk erase is performed. For example, if the `SYS_BMODE[1:0]` pins were set to b'11 for UART Boot, and if bit 3 of the `FLAGS` parameter in the Secure Header was not set, flash erasing would proceed as follows:

1. Processor: Autobaud Detect, and wait for command.
2. Host: Send Command 0x48D0B3BA (least significant byte first)

   This tells the boot kernel to erase the entire SoC flash memory. Once completed, the boot kernel enters into an error handler and requires a reset. With the flash erased, there is no sensitive information or a valid Secure Header. The processor boots without any security enabled.

For valid secure header with default keys, since security features are not active, bulk erase through command does not work. In this case the application can erase the flash by connecting a debugger.

Erase Using the Debugger

In order for the debugger to request a bulk erase, it must set the `TAPC.RCMSG[31:24]` to 0xAD as well as setting bit 15 of the register. In case of invalid header, the boot kernel does not execute the application but waits for the Bulk erase command to be issued by the host or the debugger. So with an invalid header bulk erase can be performed by the debugger by configuring `TAPC.RCMSG` bit field. After configuring the `TAPC.RCMSG` bit field, some time will elapse for the mass erase operation to be completed.

This register is preserved across a system reset event so that once the system is reset again, the boot kernel can see this setting and perform the erase operation. In case of valid header, since the boot kernel starts executing the application it does not wait for the erase command. However for bulk erase operation to take place with a valid header, it is needed to configure the `TAPC.RCMSG` bit field and then issue a system reset so that the boot kernel can recognize the erase command.

Security Programming Model

In order to enable security and its related features, the Secure Header must be programmed. By default, the SoC flash memory is not programmed so security is not enabled. This lets the application program the SoC flash with the Secure Header along with the application or any other data content.

When programming the Secure Header, program the configurable parameters like the key, boot address and the flag to determine which boot modes should be disabled. The signatures must be programmed with fixed values. The signatures indicate to the boot kernel the beginning and end of the Secure Header.

The values of the signatures are

1. signature0 = 0x16C80DBD
2. signature1 = 0xA4287F8A
3. signature2 = 0x14B234F2

Once signatures and other security related parameters are in place, the error correction codes (ECC) are calculated and inserted into the Secure Header. The ECC is Hamming 7264 method, same as that used in the boot kernel. ECC can be calculated with the help of the ECC generation utility included with the board support package or through the bootInfo API.

The following table provides an overview of the various security features.

Table 31-28: Security Feature Overview

Type of Access	Erased Header/ Valid Header with default key	Valid Header with User Key	Invalid Header
Debug Access	Boot kernel writes Default key in Secure Debug Key register Debugger can get access by sending the default key	User programmed key from Flash written to Secure Debug Key register by the boot kernel Debugger can get access by sending the custom user key	Boot kernel waits for the erase command to be issued Debugger cannot get access
SPI Master Boot	Key is not required to boot For valid header bootAddress field defines where to boot from, whereas for erased header it boots from 0x00001000 location Boot kernel still loads the default key in Secure Debug Key register	Key is not required to boot bootAddress field defines where to boot from User programmed key from Flash written to Secure Debug Key register by the boot kernel	Boot kernel waits for the erase command to be issued, hence does not boot the application
Slave Boot	No need for any key to boot from SPI or UART host Boot kernel will still load default key	User programmed key from Flash written to Secure Debug Key register by the boot kernel User key should be send by the host before the boot stream	Boot kernel waits for the erase command to be issued, hence does not boot the application

Table 31-28: Security Feature Overview (Continued)

Type of Access	Erased Header/ Valid Header with default key	Valid Header with User Key	Invalid Header
		Boot kernel examines the user key, it allows only if it matches	
Bulk Erase through Slave boot	Not applicable	Send Erase command over the host for slave boot mode	Send Erase command over the host for slave boot mode
Bulk Erase through TAPC.RCMSG register	Write to `TAPC.RCMSG` register and issue a system reset so as to boot again	Write to `TAPC.RCMSG` register and issue a system reset so as to boot again	Boot kernel waits for the erase command to be issued Write to `TAPC.RCMSG` register to perform a bulk erase

Boot ROM Programming Model

This section describes the programming model for booting the processor. The programming model includes booting functions, API calls, and data structures.

Callable Kernel API

This section describes the kernel API available at run-time.

The boot code stored in ROM exposes several functions that can be used during run-time or within Initcode or callback routines. All functions meet the C run-time calling conventions. Address provided for the documented API routines have the least significant bit set to meet the Thumb execution mode calling requirements.

adi_rom_Boot()

The boot routine provides access to boot an application at run-time through a supported peripheral. The boot routine often also provides the ability for further customization over the equivalent boot mode.

The boot routine can be used for any second-stage boot for supported Boot Modes . It provides options to boot from all device enumerations of a supported peripheral and in the case of SPI master boot modes using any slave select signal. Often any auto-configuration or detection of the device can also be disabled. Each boot mode defines its down `dBootCommand`. (See individual boot modes for a description of this parameter). The `dBootCommand` parameter is what provides a bulk of the peripheral configuration information and select the boot mode to use. The `*pCallHook` argument is used when `ROM_BFLAG_HOOK` is set and allows for users to install callback functions during the installation of a boot mode. When not required this parameter should be `NULL`.

Name	`adi_rom_Boot()`	-
PP Define	`FUNC_ROM_BOOT`	-
ROM Address	0x00000111	-

Prototype	`void * adi_rom_Boot( void *pBootAddress, uint32_t dFlags, int32_t dBlockCount, ROM_BOOT_HOOK_FUNC *pCallHook, uint32_t dBootCommand);`	-
Argument	`void *` pBootAddress	Pointer to boot stream. Typically only used for master boot modes and is used to initialize the boot structures `pSource` parameter.
Argument	`uint32_t` dFlags	Global flags to be applied to the boot process. Refer to the ***dFlags Description*** table
Argument	`int32_t` dBlockCount	Number of blocks to load. When 0 will continue until the final block in the boot stream is processed.
Argument	`ROM_BOOT_HOOK_FUNC *` pCallHook	Pointer to a hook function to be called during boot mode registration. Refer to Kernel Hook Function for more information.
Argument	`uint32_t` dBootCommand	The boot command providing the boot mode to call and boot mode specific peripheral configuration. Refer to run-time API subsections of each boot modes documentation for details.
Return Value	`void *` Pointer to next location in the boot source to be processed	Points to the next location in the boot source. This could be the next free location in the boot source or it could point to another application boot stream if multi-application boot streams are being used.
Stack Requirements	Requires a valid stack to be in place prior to the call	-

Global dFlags Description

Table 31-29: dFlags Description

Flag	Description
`ROM_BFLAG_NORESTORE`	Do not execute the cleanup routines
`ROM_BFLAG_RETURN`	Return to the function that called the boot rom and do not hand-over to the newly loaded application
`ROM_BFLAG_NEXTDXE`	Parse stream via Next DXE pointer to navigate multi-application boot streams

Table 31-29: dFlags Description (Continued)

Flag	Description
ROM_BFLAG_WAKEUP	Enable quick boot processing
ROM_BFLAG_SLAVE	Boot mode is a slave mode (this is typically set by the boot mode drivers)
ROM_BFLAG_HEADER	pLoadFunction call for block header (should be treated as read only for users as it is maintained by the boot kernel)
ROM_BFLAG_NOFIRSTHEADER	Do not load first header from boot source. Users can set this if they have prefetched the first block header and stored it in the location where the bootkernel can find it for processing. Users would typically not set this. It is often used during device auto-detection.
ROM_BFLAG_PAGEMODE	Enable page mode processing for fetching fixed size blocks of data to intermediate storage for further processing as opposed to fetching block headers then payloads from the boot source.
ROM_BFLAG_HOOK	Call hook routine during boot mode configuration

Boot Routine Example

```
 //
SPI Master Boot
uint32_t
dBootCmd =  (1<<  BITP_ROM_BCMD_SPIM_SPEED)    |

                   (7<<  BITP_ROM_BCMD_SPIM_IOPROT)   |

                   (0<<  BITP_ROM_BCMD_SPIM_DUMMY)    |

                   (0<<  BITP_ROM_BCMD_SPIM_ADDR)     |

                   (0xD << BITP_ROM_BCMD_SPIM_BCODE)  |

                   (0  << BITP_ROM_BCMD_SPIM_SSEL)    |

                   (1  << BITP_ROM_BCMD_SPIM_DEVENUM) |

                   (0  << BITP_ROM_BCMD_SPIM_NOAUTO)  |

                   (0  << BITP_ROM_BCMD_SPIM_HOST)    |

                   (ENUM_ROM_BCMD_DEVICE_SPI)        |

                   (0  << BITP_ROM_BCMD_SPIM_NOCFG  ));
adi_rom_Boot(0,0,0,0,dBootCmd);
```

adi_rom_BootKernel()

The boot kernel API can be used to implement custom boot modes.

Name	adi_rom_BootKernel()	-
PP Define	FUNC_ROM_BOOTKERNEL	-
ROM Address	0x00000109	-
Prototype	void adi_rom_BootKernel(ADI_ROM_BOOT_CONFIG *pBootStruct)	-
Argument	pBootStruct	Pointer to boot config struct
Return Value	End address of the stream processed	-
Stack Requirements	Requires a valid stack to be in place prior to the call	-

The boot kernel API performs the core processing of a boot stream. Typically, the ROM initializes a peripheral to be booted from based on the sampled boot mode pins pin and any given setting for the particular boot mode. This API exposes the core stream processing to allow custom boot modes. Rather than relying on the built-in boot mode functionality of the boot ROM kernel hooks can be installed and the kernel uses this custom functionality instead.

Prior to calling the function, the adi_rom_BootKernel() function, the pLoadFunction member of ADI_ROM_BOOT_CONFIGmust be defined. The load function copies data from one location to another based on settings in the ADI_ROM_BOOT_CONFIG structure that is passed as a parameter to the load function. The kernel validates headers, updates the destination address and size of the block, and deals with any special blocks appropriately.

NOTE: In order to make the boot implementation more flexible. The load function that is registered with the kernel must update the pSource member of ADI_ROM_BOOT_CONFIG to point to the next item of data to be fetched prior to returning to the boot kernel. This allows for a peripherals load function to fully control where the kernel will fetch blocks of data on the next load operation. This can be useful for example if the boot process is required to traverse a file structure on a removable storage device.

adi_rom_MemCopy()

The memory copy function provides a convenient facility to copy memory from one location to another using the MDMA channels. The source address, destination address and byte count must all be multiples of 4 bytes. The byte count can be any 32-bit value including zero. The DMA channel is disabled after completion, but the DMA status is not cleared so that in the event of an error the DMA can be debugged.

Name	adi_rom_MemCopy()	-
PP Define	FUNC_ROM_MEMCPY	-
ROM Address	0x00000131	-
Prototype	ROM_DMA_RESULT adi_rom_MemCopy(void * pDst, void * pSrc, uint32_t ulByteCount);	-

Argument	`void * pDst`	Destination address
Argument	`void * pSrc`	Source address
Argument	`uint32_t ulByteCount`	Byte Count
Argument	`uint32_t DmaId`	MDMA Channel ID. Valid values of 0 and 1 are supported.
Return Value	`ROM_DMA_RESULT`	
Stack Requirements	Requires a valid stack to be in place prior to the call	-

enum ROM_DMA_RESULT

Enum Type Declaration: `ROM_DMA_RESULT`

Return results for the DMA driver.

The DMA driver currently only supports a basic success or failure

Table 31-30: ROM_DMA_RESULT Enumeration

Enumerator	Description
`ROM_DMA_SUCCESS`	Success
`ROM_DMA_FAILURE`	General Failure

adi_rom_MemCompare()

Memory Compare compares a specified region of memory against a provide 32-bit reference value. The byte count can be any 32-bit value that is divisible by four, including zero. The source address in which to start comparing the data must also be 32-bit aligned. The memory compare operation uses the source MDMA channel and the MDMA channels CRC peripheral for the compare operation. The CRC peripheral is configured for memory compare mode of operation. The DMA channel is disabled after completion, but the DMA status is not cleared so that in the event of an error the DMA can be debugged.

Name	`adi_rom_MemCompare()`	-
PP Define	`FUNC_ROM_MEMCMP`	-
ROM Address	0x00000151	-
Prototype	`ROM_DMA_RESULT adi_rom_MemCompare(void * pSrc, uint32_t nCmpValue, uint32_t nByteCount, uint32_t DmaId );`	-
Argument	`void * pSrc`	Source address to read the data from to compare

Argument	`void *` nCmpValue	32-bit value to compare against
Argument	`uint32_t` nByteCount	Byte Count
Argument	`uint32_t` DmaId	MDMA Channel ID. Valid values of 0 and 1 are supported.
Return Value	`ROM_DMA_RESULT`	
Stack Requirements	Requires a valid stack to be in place prior to the call	-

enum ROM_DMA_RESULT

Enum Type Declaration: `ROM_DMA_RESULT`

Return results for the DMA driver.

The DMA driver currently only supports a basic success or failure

Table 31-31: ROM_DMA_RESULT Enumeration

Enumerator	Description
`ROM_DMA_SUCCESS`	Success
`ROM_DMA_FAILURE`	General Failure

adi_rom_MemCRC()

Memory CRC uses the source MDMA channel to read data and feed it into the CRC peripheral in order to build up a CRC32 checksum of the data. The CRC Look-up table must have been initialized prior to the call. This can be performed via the adi_rom_Crc32Poly() function. The expected CRC32 checksum is supplied as a parameter to compare against. The byte count can be any 32-bit value that is divisible by four, including zero. The DMA channel is disabled after completion, but the DMA status is not cleared so that in the event of an error the DMA can be debugged.

Name	`adi_rom_MemCRC()`	-
PP Define	`FUNC_ROM_MEMCRC`	-
ROM Address	0x00000139	-
Prototype	`ROM_DMA_RESULT adi_rom_MemCompare( void * pSrc, uint32_t nCrcChk, uint32_t nByteCount, uint32_t DmaId );`	-
Argument	`void *` pSrc	Source address of data to run the CRC32 operation on

Argument	`void *` nCrcChk	Expected CRC32 Checksum Result
Argument	`uint32_t` nByteCount	Byte Count
Argument	`uint32_t` DmaId	MDMA Channel ID. Valid values of 0 and 1 are supported.
Return Value	`ROM_DMA_RESULT`	
Stack Requirements	Requires a valid stack to be in place prior to the call	-

enum ROM_DMA_RESULT

Enum Type Declaration: `ROM_DMA_RESULT`

Return results for the DMA driver.

The DMA driver currently only supports a basic success or failure

Table 31-32: ROM_DMA_RESULT Enumeration

Enumerator	Description
`ROM_DMA_SUCCESS`	Success
`ROM_DMA_FAILURE`	General Failure

adi_rom_MemFill()

Memory Fill fills a specified region of memory with a 32-bit value provided. The byte count can be any 32-bit value that is divisible by four, including zero. The DMA channel is disabled after completion, but the DMA status is not cleared so that in the event of an error the DMA can be debugged. The CRC peripheral is used for the fill operation along with the MDMA channels destination DMA channel.

Name	`adi_rom_MemFill()`	-
PP Define	`FUNC_ROM_MEMFILL`	-
ROM Address	0x00000149	-
Prototype	`ROM_DMA_RESULT adi_rom_MemFill(void * pDst, uint32_t nFillVal, uint32_t nByteCount, uint32_t DmaId );`	-
Argument	`void *` pDst	Destination address
Argument	`void *` nFillValue	32-bit value to use for the fill operation
Argument	`uint32_t` nByteCount	Byte Count

Argument	`uint32_t` DmaId	MDMA Channel ID. Valid values of 0 and 1 are supported.
Return Value	`ROM_DMA_RESULT`	
Stack Requirements	Requires a valid stack to be in place prior to the call	-

enum ROM_DMA_RESULT

Enum Type Declaration: `ROM_DMA_RESULT`

Return results for the DMA driver.

The DMA driver currently only supports a basic success or failure

Table 31-33: ROM_DMA_RESULT Enumeration

Enumerator	Description
`ROM_DMA_SUCCESS`	Success
`ROM_DMA_FAILURE`	General Failure

adi_rom_Crc32Init()

The CRC Initcode function extracts the polynomial from the argument field of the block header (see Boot Loader Stream) and calls the adi_rom_CRC32Poly() function to initialize the CRC LUT. The address of the CRC callback function in the boot rom is then registered with the boot kernel via the boot structures pCrcFunction member.

Name	`adi_rom_Crc32Init()`	-
PP Define	`FUNC_ROM_CRC32INITCODE`	-
ROM Address	0x00000119	-
Prototype	`void adi_rom_Crc32Init( ADI_ROM_BOOT_CONFIG *pBootStruct);`	-
Argument	`ADI_ROM_BOOT_CONFIG` pBootStruct	Pointer to the `ADI_ROM_BOOT_CONFIG` structure
Return Value	`ROM_DMA_RESULT`	-
Stack Requirements	A valid stack is required to be in place before the call	-

enum ROM_DMA_RESULT

Enum Type Declaration: `ROM_DMA_RESULT`

Return results for the DMA driver.

The DMA driver currently only supports a basic success or failure

Table 31-34: ROM_DMA_RESULT Enumeration

Enumerator	Description
ROM_DMA_SUCCESS	Success
ROM_DMA_FAILURE	General Failure

adi_rom_Crc32Poly()

Initializes the CRC Look-up table using the supplied CRC polynomial. The routine loads the supplied CRC polynomial to the CRC peripheral and starts a LUT generation operation. The CRC peripheral is configured for Memory Scan Compute and Compare mode during the process. Byte Mirroring, Bit Mirroring, Polynomial Mirroring and Result Mirroring are all enabled for the CRC operations with auto clear to zero. Users need to take this configuration into consideration when supplying the polynomial to use for the CRC operations.

Name	adi_rom_Crc32Poly()	-
PP Define	FUNC_ROM_CRC32POLY	-
ROM Address	0x00000141	-
Prototype	ROM_DMA_RESULT adi_rom_Crc32Poly(uint32_t nPolynomial, uint32_t DmaId);	-
Argument	uint32_tnPolynomial	CRC32 Polynomial
Argument	uint32_t DmaId	MDMA Channel ID. Valid values of 0 and 1 are supported.
Return Value	ROM_DMA_RESULT	0 on success
Stack Requirements	A valid stack is required to be in place before the call	-

enum ROM_DMA_RESULT

Enum Type Declaration: ROM_DMA_RESULT

Return results for the DMA driver.

The DMA driver currently only supports a basic success or failure

Table 31-35: ROM_DMA_RESULT Enumeration

Enumerator	Description
ROM_DMA_SUCCESS	Success
ROM_DMA_FAILURE	General Failure

adi_rom_GetId()

Returns the 128-bit unique chip id.

Name	adi_rom_GetId()	-
PP Define	FUNC_ROM_GETID	-
ROM Address	0x00000179	-
Prototype	uint32_t adi_rom_GetId(U128_Typedef * Id, uint32_t reserved);	-
Argument	U128_Typedef * Id	Pointer to a 128-bit data structure to hold the ID.
Argument	uint32_t Reserved	Reserved
Return Value	uint32_t	Bits3:0 of the result contains the ECC results for the lower 64-bits of the ID. Bits 7:4 contain the ECC results for the upper 64-bits of the ID.
Stack Requirements	A valid stack is required to be in place before the call	-

Table 31-36: Error Return Codes

PP Define	Description
ROM_ECC_ERROR_INVS (0x3)	Invalid syndrome or checksum
ROM_ECC_ERROR_2BIT (0x2)	Uncorrectable 2-bit error
ROM_ECC_ERROR_1BIT (0x1)	1-bit error, has been corrected
ROM_ECC_ERROR_NONE (0x0)	No error

adi_rom_Ecc()

Proves ECC parity validation or generation for a 64-bit value. The function supports Hamming 7264 method which uses an 8-bit checksum for supporting ECC operations. If the return value equals ROM_ECC_ERROR_1BIT, source data is corrected in place.

Name	adi_rom_Ecc()	-
PP Define	FUNC_ROM_ECC	-
ROM Address	0x00000159	-
Prototype	ROM_DMA_RESULT adi_rom_Ecc(U64_TypeDef * pData, uint32_t ChkSum, ROM_ECC_OPTION eccOpt);	-
Argument	U64_TypeDef * pData	Pointer to 64-bit data
Argument	uint32_t ChkSum	8-bit checksum stored int the 8 lsb when performing encode operations

Argument	ROM_ECC_OPTION eccOpt	ROM_ECC_ENCODE=encode, ROM_ECC_DECODE=decode
Return Value	uint32_t result	The result contains the 8-bit checksum in the 8 lsb when eccOpt=ROM_ECC_ENCODE The return result contains an error code when eccOpt=ROM_ECC_DECODE. An error code of 0x00 indicates no error, an error code of 0x01 indicates a correctable error occurred. 0x02 and 0x03 indicate uncorrectable errors.

Table 31-37: ROM_ECC_OPTION Enumeration

PP Define	Description
ROM_ECC_ENCODE	Encode mode
ROM_ECC_DECODE	Decode mode

Table 31-38: Error Return Codes

PP Define	Description
ROM_ECC_ERROR_INVS (0x3)	Invalid syndrome or checksum
ROM_ECC_ERROR_2BIT (0x2)	Uncorrectable 2-bit error
ROM_ECC_ERROR_1BIT (0x1)	1-bit error, has been corrected
ROM_ECC_ERROR_NONE (0x0)	No error

adi_rom_GetAddress()

The Get Address routine can be used to access various look-up tables stored in the ROM. The function returns the address of the lookup table specified by the enumerator provided. Use this function rather than directly addressing tables to improve compatibility with future parts and silicon revisions.

Name	adi_rom_GetAddress()	-
PP Define	FUNC_ROM_GETADDR	-
ROM Address	0x00000101	-
Prototype	void * adi_rom_GetAddress(ROM_GETADDR_VALUE value);	-
Argument	ROM_GETADDR_VALUE value	Defines which data object to retrieve the address of

enum ROM_GETADDR_VALUE

Enum Type Declaration: `ROM_GETADDR_VALUE`

Values that can be passed to the adi_rom_GetAddress API.

The boot rom contains a number of data tables used during the boot process. The adi_rom_GetAddress routine can be used to retrieve the address of these items from the boot rom. The enumerations here specify the item to find the address of.

Table 31-39: ROM_GETADDR_VALUE Enumeration

Enumerator	Description
`ROM_GETADDR_CONSTANTS`	The constants table that contains rom version information
`ROM_GETADDR_BMODE`	The boot mode look up table containing all the default values when preboot calls adi_rom_Boot
`ROM_GETADDR_SOCFLASHBMODE`	The boot mode look up table containing all the default values for the on-chip flash
`ROM_GETADDR_ECCSYN`	ECC syndrome tables
`ROM_GETADDR_SPIXIPLUT`	The 16 entry SPI Master boot table containing all the default SPI configuration for each bcode when using memory mapped spi boot
`ROM_GETADDR_SPILUT`	The 16 entry SPI Master boot table containing all the default SPI configuration for each bcode when using peripheral spi boot

Page Mode

For the benefit of page oriented boot source devices, the boot kernel provides support for page operations. Page mode optimizes memory reads for block organized devices by always reading a page, rather than reading data on demand. The same temporary buffer used by the indirect blocks is used in page mode. The size of the buffer is defined by the `dTempByteCount` variable in the structure. The page size of the physical source device is defined by the `dPageByteCount` variable.

In order to improve performance double buffering is implemented. Two buffers are used, and while one buffer is active, DMA is used to load the next data into the second buffer. Two buffers of the size given in the configuration struct will be used.

The `pTempSource` variable points to the source address of the data that is currently in the temp buffer. This variable is an internal variable of the boot kernel. However, at any time programs can set this variable to -1 to force the kernel to re fetch source data into the temp buffer. The following items also pertain to Page mode.

- `dPageByteCount` must be a power-of-2 value (default is 4)
- `dTempByteCount` must be the same as or a multiple of `dPageByteCount` (default is 512)
- `pTempBuffer` does not have special alignment requirements. However, alignment of 32 may speed up DMA operations.

- In page mode, both block payload data and block headers are loaded through the same mechanism.

Changing Settings at Run Time

Programs can change the `pTempBuffer`, `dTempByteCount`, and `dPageByteCount` variables at any time, even within initialization codes or callbacks. Whenever the settings change the kernel continues to operate on the old settings until the content of the former temporary buffer has been entirely processed. The new settings only become active for the next load operation. The kernel can be forced to immediately switch to the new settings by setting `pTempSource` to a value of -1. Note that doing so requires the kernel to re fetch data that had been loaded earlier. Under normal conditions `pTempSource` should not be altered.

The following Initcode Routine example illustrates on how page mode can be activated in any boot mode.

```
#
The config input is the boot config datastructure
#
change PLL controller first */
initPLL()
#
enable page mode operation
set
config.dFlags.BITM_ROM_BFLAG_PAGEMODE to true;
#
set temp buffer to any unused address,
#
make sure it is not overwritten by the boot stream
#
ideally pointing to an un-initialized section, such as the stack
or heap
set
config.pTempBuffer to 0xABCDABCD;
#
update the temp byte count
#
config.pSource points to the current source address
#
config.pNextDxe points to the first address after the DXE
set
config.dTempByteCount to config.pNextDxe - config.pSource
```

Error Handler

This section describes the default error handler for the ROM and how users may provide their own custom error handler.

The default error handler eventually puts the core into a WFI state. Users can register their own custom error handler via the use of an Init Block (see Block Types) to modify the error function point in the `ADI_ROM_BOOT_CONFIG` structure. The error handler has access to the entire boot structure and receives the instruction address that triggered the error.

The expected prototype is:

```
void
ErrorFunction(ADI_ROM_BOOT_CONFIG *
pBootStruct, void *pFailingAddress);
```

The default error routine performs the following actions in order:

1. Assign the local variable `ErrorAddress` the address of the error
2. Assign the local variable `pBootLocal` with `pBoot`
3. Raise software interrupt 3. This action results in the fault being generated only due to the fault configuration performed during preboot.
4. Enter a `while(1);` loop

Users may attach with debug tools and retrieve the pointer to the boot structure and the failing address if the error handler is entered.

Boot Kernel API

The kernel provides a mechanism to provide a custom boot mode while still taking advantage of the rest of the booting framework. A custom boot mode could provide support for a peripheral that is not supported for boot by the ROM, or it could support one of the same peripherals differently. The same boot loader utility and other kernel features are available.

A boot mode, as perceived by the boot code, is a collection of four functions.

- Initialization - initialize the boot source
- Configuration - configure the boot source
- Load - read from the boot source
- Cleanup - called after booting

Of these four functions the boot kernel is only ever aware and has a requirement to support the Load function. It is this function that is responsible for the fetching of the boot stream from the boot peripheral.

To install a custom boot mode:

- create a first stage boot application to define a Load function
- use the `adi_rom_BootKernel()` API to call the boot kernel once the boot peripheral and pinmuxing has configured. Ensure all the fields of the boot structure are configured accordingly prior to performing the call.

The boot mode can use `pModeData` to preserve data across the different function calls if required.

All functions have the following prototype:

```
void
apiFunction(ADI_ROM_BOOT_CONFIG*
pBootStruct);
```

Load Function

The load function is required to read data from the source into the specified destination, according to the parameters given through the configuration struct parameter. The structure provides all of the required information read from the block header, or specified by the kernel to read the block header. The load function often makes use of the supplied DMA APIs in order to simplify the load function implementation.

As the kernel processes the stream, it calls the load function to request data. Initially, the request is for the header, then the kernel requests according to the block flags it parses. The load function must only read from the device, and write where requested.

Relevant fields for the load function can be (not limited to): `uwDataWidth`, `pSource`, `dByteCount`, `pDestination`, `loadType`.

Custom load functions must meet the following requirements.

- Protect against `dByteCount` values of zero
- Use multiple DMA units if `dByteCount` is greater than 65536 and the peripheral does not support byte count transfers greater than 65536
- The `pSource` and `pDestination` pointers must be properly updated after loading.

In slave boot modes, the boot kernel uses the address of the `dArgument` field in the `pHeader` block as the destination for the required dummy DMAs when payload data is consumed from `ROM_BFLAG_IGNORE` blocks. If the load function requires access to the `ARGUMENT` word of the block, it should be read early in the function.

Initialization/ Configuration Function

The initialization and configuration functions are called in sequence when calling a boot operation using an already supported boot peripheral via the `adi_rom_Boot()` API. These functions are used to configure the boot peripheral prior to calling the boot kernel. Both functions are called in sequence separated only by a call to a user-defined hook function. This hook function is useful when using built-in boot modes to further customize their functionality. The initialization and configuration functions are responsible for applying any required settings to any devices in use. For example, pin multiplexing may need to be applied, and any data or pointers that are used by the load function must be initialized. The specific actions depend on the device and functionality used.

Cleanup Function

The cleanup function is called after the entire boot stream is read, and the kernel has completed its boot mode-specific function. This is only performed when using the `adi_rom_Boot()` API. Resetting of any status registers, or device parameters is done to prepare the environment for the execution of the newly loaded application.

Kernel Hook Function

The kernel provides multiple hooks through the `adi_rom_Boot()` hook function parameter. By using this feature, it is possible to alter the state of the processor, and the course the boot kernel takes.

The hook function must adhere to the following prototype:

```
int32_t
hookFunction(ADI_ROM_BOOT_CONFIG* pBootconfig, ROM_HOOK_CALL_CAUSE);
```

By modifying settings in the ADI_ROM_BOOT_CONFIG structure, many alterations of the boot process can be achieved. Much of the same functionality that is available in an Init Block can be provided through the hook function, with even more flexibility for customization. The hook function is called once after executing the boot modes Init routine then once again after executing the boot modes Config routine. A flag passed to the hook function allows software to determine at which point the call took place to allow for conditional processing to occur at different stages of the setup phase.

Return Value

The hook function must return a zero value in order for normal booting to continue. A non-zero return value will cause the ROM to skip over loading of any data and immediately transfer control according to Boot Termination and Application Execution .

Hook Call Cause

When the hook function is called, a parameter is passed indicating why the hook function was called.

enum ROM_HOOK_CALL_CAUSE

Enum Type Declaration: ROM_HOOK_CALL_CAUSE

Specifies the cause of a call to a hook function.

Hook functions can be called after a boot modes Init or Config functions is executed. This allows user to add custom code to alter the configuration that may have been applied by the code in the boot rom. This cause of the call is passed to the hook function to allow for conditional processing to be performed depending on whether the call was made after the execution of the Init function or execution of the Config function.

Table 31-40: ROM_HOOK_CALL_CAUSE Enumeration

Enumerator	Description
ROM_HOOK_CALL_INIT_COMPLETE	Hook function called as a result of Init function completion
ROM_HOOK_CALL_CONFIG_COMPLETE	Hook function called as a result of Config function completion

Kernel Return Feature

The kernel provides a feature to bypass calling of the loaded application, and to simply return instead. This can be useful when using the adi_rom_Boot() API. The kernel returns the next address after the last loaded application block.

To enable this feature, set the ROM_BFLAG_RETURN flag in the dFlags field when calling the API. Upon completion of loading the new boot stream the application to the existing application.

Boot Termination and Application Execution

When the boot kernel completes the processing of the boot stream, a sequence of events is required to then pass control to the loaded application.

When the boot process is complete, the boot code must pass control to the loaded application. The first block of a boot stream, which is marked with the BFLAG_FIRST flag, contains a pointer to the vector table in the TARGET_ADDRESS field of the boot block. This value is written to the RCU0_SVECT0 register when the flag is processed.

The No Return feature can suppress a call to the application.

The boot kernel reads the value of the RCU0_SVECT0 register. This register points to the location of the vector table for the application. The first two entries of the vector table contain the stack pointer and the reset vector address where code execution must originate. The boot kernel does not update VTOR or SP_main. The set-up code of the application updates the vector table and the stack pointer. This functionality allows for the boot stack to be preserved both for debug -purposes and in the event a user application wishes to return to the boot kernel. Not updating VTOR ensures that the default handlers for the interrupts that are enabled during boot code execution remain in place. The interrupts remain in place until the users application reconfigures all interrupt functionality and handlers for their application.

IMPORTANT: The stack pointer must be between 0x10000000-0x1005FFFF and 0x20000000-0x2005FFFF, within the SRAM memory space. A stack pointer of any other value results in a call to the error handler routine.

If the stack for the boot kernel is not required to be preserved, then the users run-time setup can initialize SP_main back to the top of the data memory region. This operation makes more stack resources available.

Booting Data Structures

The programming model for booting the processor uses the data structures defined in this section.

struct ADI_ROM_BOOT_HEADER

Structure Type Declaration: ADI_ROM_BOOT_HEADER

Boot Block Header.

Boot block headers control the loading process of the boot stream. The block header is 16-bytes in size consisting of four members. The block header has a basic form of integrity checking by requiring a fixed 0xAD signature being present in bits 31:24 of the dBlockCode, in addition to the entire block header having an 8-bit XOR checksum present in bit 23:16 of the dBlockCode member.

All block header TargetAddress values and byte counts are required to be a multiple of 4-bytes.

The dArgument field contents vary depending on the block type being processed. For full details Refer to the Boot Loader Stream section for each of the supported block types.

Table 31-41: ADI_ROM_BOOT_HEADER Members

Type	Name	Description
int32_t	dBlockCode	Contains the bcode, boot flags, an 8-bit checksum and a signature byte.
void *	pTargetAddress	Destination Address of Payload
int32_t	dByteCount	Byte Count of the Payload
int32_t	dArgument	Argument functionality varies depending on operation

dBlockCode

Contains the bcode, boot flags, an 8-bit checksum and a signature byte.

Bits 3:0 contain the bcode that is used for example by the SPI master boot mode to reconfigure the SPI peripheral

Bits 15:4 contain the block specific boot flags instructing the kernel how to process the block

Bits 23:16 8-bit XOR checksum of the 16-byte block header

Bits 31:24 must contain the 0xAD signature

pTargetAddress

Destination Address of Payload

dByteCount

Byte Count of the Payload

dArgument

Argument functionality varies depending on operation

struct ADI_ROM_BOOT_SECHDR

Structure Type Declaration: ADI_ROM_BOOT_SECHDR

Secure Header storage to be located at absolute address 0x00000000 in the SPI flash.

The secure header contains a number of items related to the security functionality of the product. The secure header is integrity protected by fixed signatures at the head and base of the object and the entire structure is ECC protected. Each 8-bit ECC parameter protects a 64-bit field of the secure header structure. The adi_rom_boot_Ecc() routine can be used to calculate and verify the ECC functionality.

Table 31-42: ADI_ROM_BOOT_SECHDR Members

Type	Name	Description
uint32_t	signature0	Signature 0, must be 0x16C80DBD

Table 31-42: ADI_ROM_BOOT_SECHDR Members (Continued)

Type	Name	Description
uint8_t	reserved0	Reserved and should be programmed to all zeros
ADI_ROM_BOOT_SECHDR_KEY	userkey	The 128-bit User Key to secure Debug access and access to the slave boot modes.
uint32_t	flags	Various boot flags to further customize the secure boot process
uint32_t	boot_address	The address to boot from in the SPI flash or the location of the base address of the vector table
uint8_t	ecc0	ECC data for signature and reserved0 fields
uint8_t	ecc1	ECC data for userkey.key0 and userkey.key1
uint8_t	ecc2	ECC data for userkey.key2 and userkey.key3
uint8_t	ecc3	ECC data for flags and boot_address
uint8_t	ecc4	ECC data for signature1 and signature2
uint8_t	reserved1	Reserved, should be programmed to all zeros
uint32_t	signature1	Signature 1, must be 0xA4287F8A
uint32_t	signature2	Signature 2, must be 0x14B234F2

signature0

Signature 0, must be 0x16C80DBD

reserved0

Reserved and should be programmed to all zeros

userkey

The 128-bit User Key to secure Debug access and access to the slave boot modes.

The default 128-bit user key to access the device with debug tools when the secure header is blank is:

SDBGKEY[31:0] = 0xA4B8E4A5

SDBGKEY[63:32] = 0xD2041DD7

SDBGKEY[95:64] = 0x18839DF8

SDBGKEY[127:92] = 0x8392C1FE

flags

Various boot flags to further customize the secure boot process

boot_address

The address to boot from in the SPI flash or the location of the base address of the vector table

ecc0

ECC data for signature and reserved0 fields

ecc1

ECC data for userkey.key0 and userkey.key1

ecc2

ECC data for userkey.key2 and userkey.key3

ecc3

ECC data for flags and boot_address

ecc4

ECC data for signature1 and signature2

reserved1

Reserved, should be programmed to all zeros

signature1

Signature 1, must be 0xA4287F8A

signature2

Signature 2, must be 0x14B234F2

struct ADI_ROM_BOOT_SECHDR_FLAGS

Structure Type Declaration: `ADI_ROM_BOOT_SECHDR_FLAGS`

Secure Header flags for various security related operations.

It is possible for the user to disable each of the boot modes via this parameter in the secure header. If one of the slave boot modes was disabled then the processor will continue to boot only form the on-chip flash.

Table 31-43: ADI_ROM_BOOT_SECHDR_FLAGS Members

Type	Name	Description
uint32_t	bmode0_dis: 1 (bitfield)	Disable boot mode 0
uint32_t	bmode1_dis: 1 (bitfield)	Disable boot mode 1
uint32_t	bmode2_dis: 1 (bitfield)	Disable boot mode 2
uint32_t	bmode3_dis: 1 (bitfield)	Disable boot mode 3
uint32_t	reserved: 28 (bitfield)	Reserved

bmode0_dis

Disable boot mode 0

bmode1_dis

Disable boot mode 1

bmode2_dis

Disable boot mode 2

bmode3_dis

Disable boot mode 3

reserved

Reserved

struct ADI_ROM_BOOT_SECHDR_KEY

Structure Type Declaration: ADI_ROM_BOOT_SECHDR_KEY

128-bit Security Key.

In order to gain access to the debug interface or to enable slave boot modes a 128-bit key is required. The key is stored in the secure header in the on-chip flash. The boot code after verifying the integrity of the secure header in the flash loads the 128-bit to the TAPC in order to allow debug access. The key is also used for protecting the slave boot modes.

Table 31-44: ADI_ROM_BOOT_SECHDR_KEY Members

Type	Name	Description
uint32_t	key0	Bits 31:0 of the 128-bit key
uint32_t	key1	Bits 63:32 of the 128-bit key
uint32_t	key2	Bits 95:64 of the 128-bit key
uint32_t	key3	Bits 127:96 of the 128-bit key

key0

Bits 31:0 of the 128-bit key

key1

Bits 63:32 of the 128-bit key

key2

Bits 95:64 of the 128-bit key

key3

Bits 127:96 of the 128-bit key

struct ADI_ROM_BOOT_SPI_FUNC

Structure Type Declaration: ADI_ROM_BOOT_SPI_FUNC

The SPI Master Boot Mode Function Pointer Structure.

This structure allows calls to different SPI driver routines in order to reset the device, read the ID, configure the MultiIO capabilities and configuration of dummy cycles.

The generic SPI boot function can also leverage this functionality. The boot code currently supports 3 different methods of enabling quad mode in order to support a large number of flash manufacturers. The pMultiIOEnFunc pointer can be initialized depending on what routine needs to be called.

Table 31-45: ADI_ROM_BOOT_SPI_FUNC Members

Type	Name	Description
ROM_BOOT_SPI_RESET_FUNC *	pResetFunc	Pointer to the reset function
ROM_BOOT_SPI_READID_FUNC *	pReadIdFunc	Pointer to the Read ID function

Table 31-45: ADI_ROM_BOOT_SPI_FUNC Members (Continued)

Type	Name	Description
ROM_BOOT_SPI_CONFIG_FUNC *	pConfigFunc	Pointer to the Configuration function

pResetFunc

Pointer to the reset function

pReadIdFunc

Pointer to the Read ID function

pConfigFunc

Pointer to the Configuration function

struct ADI_ROM_BOOT_SPI_ID

Structure Type Declaration: ADI_ROM_BOOT_SPI_ID

SPI Flash Device ID.

Used for storage of the manufacturer and device ID. It also provides some limited storage for a unique ID.

Table 31-46: ADI_ROM_BOOT_SPI_ID Members

Type	Name	Description
uint8_t	nManId	Manufacturer ID
uint16_t	nDevId	Device ID
uint16_t	nUniqueIdMaxSize	Maximum number of bytes supported by the nUniqueId field
uint16_t	nUniqueIdSize	Size of the unique ID to fetch
uint8_t	nUniqueId	Storage space for the unique ID

nManId

Manufacturer ID

nDevId

Device ID

nUniqueIdMaxSize

Maximum number of bytes supported by the nUniqueId field

nUniqueIdSize

Size of the unique ID to fetch

nUniqueId

Storage space for the unique ID

struct ADI_ROM_BOOT_UART

Structure Type Declaration: `ADI_ROM_BOOT_UART`

UART Slave Boot Mode Specific Structure.

Table 31-47: ADI_ROM_BOOT_UART Members

Type	Name	Description
`uint32_t`	`nFlags`	Stores the timer ID that is required to be used by the UART peripheral for autobaud detection

nFlags

Stores the timer ID that is required to be used by the UART peripheral for autobaud detection

struct ADI_ROM_CONSTANTS_TYPE

Structure Type Declaration: `ADI_ROM_CONSTANTS_TYPE`

The Boot ROM constants containing version information.

Table 31-48: ADI_ROM_CONSTANTS_TYPE Members

Type	Name	Description
`uint32_t`	`revision`	ROM Revision Code
`uint32_t`	`xorchk`	ROM 32-bit XOR Checksum of the Boot ROM
`uint32_t`	`zero`	All Zero Value
`uint32_t`	`ones`	All Ones value
`uint32_t`	`date`	ROM Date Code
`uint32_t`	`time`	ROM Time Code
`uint32_t`	`reserved0`	Reserved for future use

revision

ROM Revision Code

xorchk

ROM 32-bit XOR Checksum of the Boot ROM

zero

All Zero Value

ones

All Ones value

date

ROM Date Code

time

ROM Time Code

reserved0

Reserved for future use

struct U64_TypeDef

Structure Type Declaration: `U64_TypeDef`

Used to store 64-bits of data for use with the adi_rom_Ecc() function.

The boot code contains routines for calculating ECC parity data that is used to support the Security Header integrity protection scheme as well as the reading of the device unique ID.

Table 31-49: U64_TypeDef Members

Type	Name	Description
`unsigned long`	`h`	Upper 32 MSB bits of the 64-bit data
`unsigned long`	`l`	Lower 32 LSB bits of the 64-bit data

h

Upper 32 MSB bits of the 64-bit data

l

Lower 32 LSB bits of the 64-bit data

struct ADI_ROM_BOOT_BUFFER

Structure Type Declaration: `ADI_ROM_BOOT_BUFFER`

A simple buffer object used by the boot kernel.

The buffer object simply contains a pointer to the buffer content and the size of the buffer.

Table 31-50: ADI_ROM_BOOT_BUFFER Members

Type	Name	Description
`void *`	`pBuffer`	Pointer to the buffer
`int32_t`	`dByteCount`	Size of the buffer

pBuffer

Pointer to the buffer

dByteCount

Size of the buffer

struct ADI_ROM_BOOT_CONFIG

Structure Type Declaration: `ADI_ROM_BOOT_CONFIG`

Boot configuration contains the entire context for the boot process.

The ADI_ROM_BOOT_CONFIG structure provides access to a majority of the boot kernels functionality. The structure is passed to all user customizable functions allowing users to heavily customize and manipulate the boot process if required. For general booting scenarios the structure is primarily only required for debug situations in the event of determining the cause of a boot failure.

Table 31-51: ADI_ROM_BOOT_CONFIG Members

Type	Name	Description
`void *`	`pSource`	Current source address in the boot process.
`void *`	`pDestination`	Destination address for the requested data.
`uint32_t volatile *`	`pControlRegister`	Address of the boot peripherals Control Register.
`uint32_t volatile *`	`pPeripheralBase`	Pointer to the base address MMR address of the boot peripheral.
`uint32_t volatile *`	`pAuxControlRegister`	Pointer to the control register of an auxiliary peripheral.

Table 31-51: ADI_ROM_BOOT_CONFIG Members (Continued)

Type	Name	Description
uint32_t volatile *	pAuxPeripheralBase	Pointer to the base MMR address of an auxiliary peripheral.
ADI_DMA_ TypeDef *	pDmaBaseRegister	Pointer to the base MMR address of the boot peripherals DMA controller.
uint32_t volatile *	pSecControlRegister	Base address of the SEC SSI instance associated with the boot peripheral.
int32_t	dControlValue	Value to be loaded to the boot peripherals control register.
int32_t	dByteCount	The number bytes being requested by the boot kernel.
int32_t	dFlags	Flags used to instruct the boot kernel how to process a particular block.
uint16_t	uwDataWidth	The data width of the boot source.
uint16_t	uwSrcModifyMult	The source modifier for the boot source.
uint16_t	uwDstModifyMult	The destination modifier for the boot destination.
uint16_t	uwUserShort	Free to use by the user
int32_t	dUserLong	Free to use by the user
int32_t	dReserved	Reserved for future use
void *	pModeData	Pointer to boot mode specific data structure.
int32_t	dBootCommand	The Boot Command value supplied for the boot process
void *	pNextDxe	Pointer to the next DXE in the boot stream or the first free location after the boot stream
ROM_BOOT_ ERROR_FUNC	pErrorFunction	Pointer to a user defined Error handler.
ROM_BOOT_ MODE_LOAD_ FUNC	pLoadFunction	Pointer to the boot modes Load function.
ROM_BOOT_ CALLBACK_ FUNC	pCallBackFunction	Pointer to a user supplied callback function.
ROM_BOOT_ CALLBACK_ FUNC	pCrcFunction	Pointer to a CRC callback function.
void *	pReserved	Reserved
ADI_ROM_ BOOT_ HEADER	pHeader	Pointer to a location where the 16-byte block headers are stored.

Table 31-51: ADI_ROM_BOOT_CONFIG Members (Continued)

Type	Name	Description
ADI_ROM_BOOT_SECHDR	pSecHdr	Pointer to the Secure Header.
ROM_SECHDR_RESULT	sechdr_result	Secure Header verification result.
uint8_t	flash_config	Is used to specify the SPI flash device present on the device
uint8_t	reserved1	Reserved
void *	pTempBuffer	Pointer to the Intermediate Temp Buffer.
int32_t	dTempByteCount	Size of our temporary buffer in bytes
void *	pTempSource	Current location within the temp buffer that is being processed
int32_t	dPageByteCount	Page Size for when Page mode is enabled.
uint32_t	ulBlockCount	The number of blocks that the boot kernel has processed
uint32_t	ulBlockCurrent	The block number the boot kernel is currently operating on
void *	pLogBuffer	Pointer to the log buffer
void *	pLogCurrent	The current position within the log buffer
int32_t	dLogByteCount	The size of the log buffer

pSource

Current source address in the boot process.

The source address is updated during execution of the Load routine for a given boot mode and is not maintained by the boot kernel. This allows for a more highly customized boot process allowing user supplied load functions to divert the boot flow if required.

pDestination

Destination address for the requested data.

The load function for the boot mode is required to transfer the requested number of bytes to the address specified by this parameter. The parameter is used for all data requests whether they be requests for 16-byte block header of for associated payloads.

pControlRegister

Address of the boot peripherals Control Register.

Boot drivers can use this parameter to specify the address of the peripherals main control register. It may be used in combination with the dControlValue field which can specify a value to be loaded to the peripherals control register. The use of this field is optional depending on the boot driver implementation.

pPeripheralBase

Pointer to the base address MMR address of the boot peripheral.

This pointer is cast to the structure type required to access the peripheral MMR registers. This allows for the boot drivers to be developed such that they function across all enumerations of a supported peripheral. All MMR accesses to the boot peripheral are performed via this pointer.

pAuxControlRegister

Pointer to the control register of an auxiliary peripheral.

Some peripherals require the use of an auxiliary peripheral. For example UART boot modes use the core timers for autobaud detection. The use of this parameter is not compulsory and is boot mode specific.

pAuxPeripheralBase

Pointer to the base MMR address of an auxiliary peripheral.

Some peripherals require the use of an auxiliary peripheral. For example UART boot modes use the core timers for autobaud detection. The use of this parameter is not compulsory and is boot mode specific.

pDmaBaseRegister

Pointer to the base MMR address of the boot peripherals DMA controller.

Used by the DMA routines to access the DMA channel associated with a boot peripheral

pSecControlRegister

Base address of the SEC SSI instance associated with the boot peripheral.

While initialized by the boot drivers this value is not currently used during boot

dControlValue

Value to be loaded to the boot peripherals control register.

This is not a compulsory parameter, however boot mode drivers can make use of this if they are required to store context that needs to be reloaded during the Load function. The register in which the value is to be written can be specified via pControlRegister.

dByteCount

The number bytes being requested by the boot kernel.

Load functions must transfer the requested number of bytes from the boot source to the destination address.

dFlags

Flags used to instruct the boot kernel how to process a particular block.

The contents of this register are a combination of the global flags passed via the adi_rom_Boot() function as well as the boot flags specific to the current boot block header that is being processed.

uwDataWidth

The data width of the boot source.

Typically most peripherals are configured for byte mode of operation. A value of 0x0000 is used to indicate byte mode. This is used for calculating the required PSize for the peripheral DMA operation

uwSrcModifyMult

The source modifier for the boot source.

Typically this would always be set to 1 for byte mode of operation. The value is used to update the pSource pointer to point to the next data to fetch from the boot source.

uwDstModifyMult

The destination modifier for the boot destination.

Typically this would always be set to 1 for byte mode of operation. The value is used to update the destination modifier in the DMA operation.

uwUserShort

Free to use by the user

dUserLong

Free to use by the user

dReserved

Reserved for future use

pModeData

Pointer to boot mode specific data structure.

This pointer is cast to the type of structure that is specific to the boot mode allowing the boot driver routines access to the boot mode specific structures. When a boot mode structure is created in memory, the address of the structure is written to this member.

dBootCommand

The Boot Command value supplied for the boot process

pNextDxe

Pointer to the next DXE in the boot stream or the first free location after the boot stream

pErrorFunction

Pointer to a user defined Error handler.

When NULL the default error handler will be called. Users can update this pointer via hook routines or init codes to register their own error handler. When not a NULL pointer the boot rom error handler will make a call to the address located in this register.

pLoadFunction

Pointer to the boot modes Load function.

The boot kernel makes a call to the address located in this member every time it requires more data from the boot source. Users can update this pointer to install a customized Load function for advanced boot techniques and for support of second stage boot loaders for unsupported boot peripherals

pCallBackFunction

Pointer to a user supplied callback function.

The boot kernel makes a call to the address located in this member when the ROM_BFLAG_CALLBACK flag is set. Users must initialize this pointer via init codes or hook functions to install the callback function.

pCrcFunction

Pointer to a CRC callback function.

The boot kernel makes a call to the address located in this member after every block header payload is processed. This pointer is initialized by the boot codes adi_rom_CrcInit() function. When the kernel detects that pointer is not a NULL pointer it will make a call to the address after every payload is processed in order to verify the content via the CRC peripheral.

pReserved

Reserved

pHeader

Pointer to a location where the 16-byte block headers are stored.

When the boot kernel request a 16-byte block header, the kernel sets the pDestination member to the contents of this member.

pSecHdr

Pointer to the Secure Header.

The boot process fetches the secure header from the SPI flash and stored it in internal SRAM for further processing. This value specifies the location of the secure header so the slave boot modes can access the keys for slave boot mode authentication.

sechdr_result

Secure Header verification result.

The secure header goes through a verification process in order to verify the integrity of the secure header and the result of that verification is stored in this member.

flash_config

Is used to specify the SPI flash device present on the device

reserved1

Reserved

pTempBuffer

Pointer to the Intermediate Temp Buffer.

The Temp Buffer is used as an intermediate storage location for indirect block processing and also for using page mode accesses. Boot data is loaded from the boot source into this buffer when these features are enabled then the boot kernel processes the data from the TempBuffer

dTempByteCount

Size of our temporary buffer in bytes

pTempSource

Current location within the temp buffer that is being processed

dPageByteCount

Page Size for when Page mode is enabled.

Page mode is intended to help support block based devices that do not support reading of an arbitrary number of bytes. If a boot peripheral has restrictions on the minimum size of the number of bytes that can be read then page mode can be enabled and the size of the page request is stored in this member

ulBlockCount

The number of blocks that the boot kernel has processed

ulBlockCurrent

The block number the boot kernel is currently operating on

pLogBuffer

Pointer to the log buffer

pLogCurrent

The current position within the log buffer

dLogByteCount

The size of the log buffer

struct ADI_ROM_BOOT_LOCAL

Structure Type Declaration: `ADI_ROM_BOOT_LOCAL`

The Boot Kernels Local Structure.

A local structure created by the boot kernel on the stack to save context during various boot operations

Table 31-52: ADI_ROM_BOOT_LOCAL Members

Type	Name	Description
`void *`	`pCallbackBuffer`	Pointer to the callback buffer. keep this always as the first entry
`int32_t`	`dCallbackByteCount`	The byte count of the payload being processed by the callback
`void *`	`pCurrTempBuffer`	Pointer to the current buffer
`int32_t`	`dCurrTempByteCount`	The current size of the temp buffer
`int32_t`	`dCurrPageByteCount`	The current size of a page for block based devices
`int32_t`	`dGlobalFlags`	Global flags to be applied to every block processed by the boot kernel
`int32_t`	`dReserved`	Reserved for future use

pCallbackBuffer

Pointer to the callback buffer. keep this always as the first entry

dCallbackByteCount

The byte count of the payload being processed by the callback

pCurrTempBuffer

Pointer to the current buffer

dCurrTempByteCount

The current size of the temp buffer

dCurrPageByteCount

The current size of a page for block based devices

dGlobalFlags

Global flags to be applied to every block processed by the boot kernel

dReserved

Reserved for future use

struct ADI_ROM_BOOT_REGISTRY

Structure Type Declaration: `ADI_ROM_BOOT_REGISTRY`

Boot Mode Registration.

Each supported boot mode has a registration function that is called. This function is responsible for registering the addresses of the Init function, Config function and the Cleanup function for the particular boot mode. The boot code then calls the Init and Config functions prior to calling the boot kernel in order to set up the boot peripheral. The cleanup function is called after the boot process and prior to handing over to the newly booted application.

Table 31-53: ADI_ROM_BOOT_REGISTRY Members

Type	Name	Description
`uint32_t`	`dStructSize`	The size of the boot mode specific structures
`ROM_BOOT_MODE_INIT_FUNC`	`pInitFunction`	Pointer to the boot modes Init Function

Table 31-53: ADI_ROM_BOOT_REGISTRY Members (Continued)

Type	Name	Description
ROM_BOOT_MODE_CONFIG_FUNC	pConfigFunction	Pointer to the boot modes Config Function
ROM_BOOT_MODE_CLEANUP_FUNC	pCleanUpFunction	Pointer to the boot modes Cleanup Function
void *	pReserved	Reserved for future use
int32_t	dReserved	Reserved for future use

dStructSize

The size of the boot mode specific structures

pInitFunction

Pointer to the boot modes Init Function

pConfigFunction

Pointer to the boot modes Config Function

pCleanUpFunction

Pointer to the boot modes Cleanup Function

pReserved

Reserved for future use

dReserved

Reserved for future use

struct ADI_ROM_BOOT_SPI

Structure Type Declaration: ADI_ROM_BOOT_SPI

The SPI Master Boot Mode Specific Structure.

Table 31-54: ADI_ROM_BOOT_SPI Members

Type	Name	Description
uint8_t	ubReadCommand	Read command to use to access the SPI device
uint8_t	ubDummyBytes	Number of dummy bytes to issue after the read command
uint8_t	ubAddressBytes	Number of address bytes required to access the device
uint8_t	ubDataBits	Additional control bits
uint16_t	uwClkLower	The SPI clock divider value to be used
uint16_t	uReserved0	reserved
uint32_t	nTxCtl	The value to be written to the upper 16 bits of the SPI_CTL register for address and dummy cycles
uint32_t	nRxCtl	The value to be written to the upper 16 bits of the SPI_CTL register for a data receive operation
uint32_t	nCmdCtl	The value to be written to the upper 16 bits of the SPI_CTL register for read command cycles
ROM_BOOT_SPIM_IO_ENABLE_FUNC	pMIOEnFunction	Pointer to the routine that enables quad mode on the flash device
uint8_t	nDummy	The Dummy Byte value to be used
uint8_t	nFlags	Currently this byte should always be 0 and is reserved for future use
uint16_t	uReserved2	Reserved
void *	pXIPAddress	The memory mapped address to boot from
ADI_ROM_BOOT_SPI_FUNC	functions	Device specific driver routines
ADI_ROM_BOOT_SPI_ID	id	The electronic signature of the device

ubReadCommand

Read command to use to access the SPI device

ubDummyBytes

Number of dummy bytes to issue after the read command

ubAddressBytes

Number of address bytes required to access the device

ubDataBits

Additional control bits

uwClkLower

The SPI clock divider value to be used

uReserved0

reserved

nTxCtl

The value to be written to the upper 16 bits of the SPI_CTL register for address and dummy cycles

nRxCtl

The value to be written to the upper 16 bits of the SPI_CTL register for a data receive operation

nCmdCtl

The value to be written to the upper 16 bits of the SPI_CTL register for read command cycles

pMIOEnFunction

Pointer to the routine that enables quad mode on the flash device

nDummy

The Dummy Byte value to be used

nFlags

Currently this byte should always be 0 and is reserved for future use

uReserved2

Reserved

pXIPAddress

The memory mapped address to boot from

functions

Device specific driver routines

id

The electronic signature of the device

enum ROM_BMODE_RESULT

Enum Type Declaration: `ROM_BMODE_RESULT`

Return results for the boot mode specific driver routines.

Functions such as the boot modes Register,Init, Config, Load and Cleanup routines should return this type of result

Table 31-55: ROM_BMODE_RESULT Enumeration

Enumerator	Description
`ROM_BMODE_SUCCESS`	Success
`ROM_BMODE_FAILURE`	General Failure
`ROM_BMODE_SPI_SWITCH`	Boot alternate SPI port as on-chip flash not supported
`ROM_BMODE_ILLEGAL_DEVNUM`	Unsupported device enumeration
`ROM_BMODE_EXIT`	Exit the boot process and return

enum ROM_DMA_RESULT

Enum Type Declaration: `ROM_DMA_RESULT`

Return results for the DMA driver.

The DMA driver currently only supports a basic success or failure

Table 31-56: ROM_DMA_RESULT Enumeration

Enumerator	Description
`ROM_DMA_SUCCESS`	Success
`ROM_DMA_FAILURE`	General Failure

enum ROM_KERNEL_RESULT

Enum Type Declaration: `ROM_KERNEL_RESULT`

Return results for various kernel operations.

Functions used by the boot kernel for general processing use this return type

Table 31-57: ROM_KERNEL_RESULT Enumeration

Enumerator	Description
`ROM_KERNEL_SUCCESS`	Success
`ROM_KERNEL_HDR_SIGNFAIL`	Invalid header sign in block header
`ROM_KERNEL_HDR_CHKSUMFAIL`	Invalid checksum in block header
`ROM_KERNEL_FAILURE`	General Failure

enum ROM_SECHDR_RESULT

Enum Type Declaration: `ROM_SECHDR_RESULT`

Return results for the secure header validation.

When processing the secure header to check the integrity the function can return one of the values listed

Table 31-58: ROM_SECHDR_RESULT Enumeration

Enumerator	Description
`ROM_SECHDR_SUCCESS`	Valid Secure Header
`ROM_SECHDR_ECC0_ERR`	ECC0 uncorrectable error detected in the block header
`ROM_SECHDR_ECC1_ERR`	ECC1 uncorrectable error detected in the block header
`ROM_SECHDR_ECC2_ERR`	ECC2 uncorrectable error detected in the block header
`ROM_SECHDR_ECC3_ERR`	ECC3 uncorrectable error detected in the block header
`ROM_SECHDR_ECC4_ERR`	ECC4 uncorrectable error detected in the block header
`ROM_SECHDR_SIG_ERR`	Invalid header signature in block header
`ROM_SECHDR_KEY_DISABLED`	Security header was found to be disabled due to reading all zeros or all ones
`ROM_SECHDR_FAILURE`	General Failure

enum ROM_GETADDR_VALUE

Enum Type Declaration: `ROM_GETADDR_VALUE`

Values that can be passed to the adi_rom_GetAddress API.

The boot rom contains a number of data tables used during the boot process. The adi_rom_GetAddress routine can be used to retrieve the address of these items from the boot rom. The enumerations here specify the item to find the address of.

Table 31-59: ROM_GETADDR_VALUE Enumeration

Enumerator	Description
`ROM_GETADDR_CONSTANTS`	The constants table that contains rom version information
`ROM_GETADDR_BMODE`	The boot mode look up table containing all the default values when preboot calls adi_rom_Boot
`ROM_GETADDR_SOCFLASHBMODE`	The boot mode look up table containing all the default values for the on-chip flash
`ROM_GETADDR_ECCSYN`	ECC syndrome tables
`ROM_GETADDR_SPIXIPLUT`	The 16 entry SPI Master boot table containing all the default SPI configuration for each bcode when using memory mapped spi boot

Table 31-59: ROM_GETADDR_VALUE Enumeration (Continued)

Enumerator	Description
ROM_GETADDR_SPILUT	The 16 entry SPI Master boot table containing all the default SPI configuration for each bcode when using peripheral spi boot

enum ROM_HOOK_CALL_CAUSE

Enum Type Declaration: ROM_HOOK_CALL_CAUSE

Specifies the cause of a call to a hook function.

Hook functions can be called after a boot modes Init or Config functions is executed. This allows user to add custom code to alter the configuration that may have been applied by the code in the boot rom. This cause of the call is passed to the hook function to allow for conditional processing to be performed depending on whether the call was made after the execution of the Init function or execution of the Config function.

Table 31-60: ROM_HOOK_CALL_CAUSE Enumeration

Enumerator	Description
ROM_HOOK_CALL_INIT_COMPLETE	Hook function called as a result of Init function completion
ROM_HOOK_CALL_CONFIG_COMPLETE	Hook function called as a result of Config function completion

32 System Watchpoint Unit (SWU)

The system watchpoint unit (SWU) is a single module used for transaction monitoring. The SWU is attached to each system slave through the system crossbar interface and provides ports for all address channel signals for the system crossbar. The SWU does not have ports for the read/write data channel signals or the low-power interface signals.

Each SWU contains four match groups of registers with associated hardware. These four SWU match groups operate independently, but share common event (interrupt and trigger) outputs. Each match group can monitor either the write or read address channel and can operate in either watchpoint mode or bandwidth mode.

SWU Features

The system watchpoint unit has the following features.

- Four independent match groups for each SWU
- Each match group can operate in either bandwidth mode or watchpoint mode

SWU Functional Description

This section describes the function of the SWU match block, interface block, and MMR block.

ADSP-CM40x SWU Register List

The System Watchpoint Unit (SWU) provides debug and development support through flexible transaction level and bandwidth monitoring and associated event triggering. The SWU can generate events based on monitoring transactions at the system slaves through watchpoint-match groups. The SWU also provides watchpoint event status reporting, a global lock, and processor reset capability. A set of registers governs SWU operations. For more information on SWU functionality, see the SWU register descriptions.

Table 32-1: ADSP-CM40x SWU Register List

Name	Description
SWU_CNT[n]	Count Register n
SWU_CTL[n]	Control Register n

Table 32-1: ADSP-CM40x SWU Register List (Continued)

Name	Description
SWU_CUR[n]	Current Register n
SWU_GCTL	Global Control Register
SWU_GSTAT	Global Status Register
SWU_HIST[n]	Bandwidth History Register n
SWU_ID[n]	ID Register n
SWU_LA[n]	Lower Address Register n
SWU_TARG[n]	Target Register n
SWU_UA[n]	Upper Address Register n

ADSP-CM40x SWU Interrupt List

Table 32-2: ADSP-CM40x SWU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
118	SWU0_EVT	SWU0 Event	None	
119	SWU1_EVT	SWU1 Event	None	
120	SWU2_EVT	SWU2 Event	None	
121	SWU3_EVT	SWU3 Event	None	
122	SWU4_EVT	SWU4 Event	None	

ADSP-CM40x SWU Trigger List

Table 32-3: ADSP-CM40x SWU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
60	SWU0_EVT	SWU0 Event	None
61	SWU1_EVT	SWU1 Event	None
62	SWU2_EVT	SWU2 Event	None
63	SWU3_EVT	SWU3 Event	None
64	SWU4_EVT	SWU4 Event	None

Table 32-4: ADSP-CM40x SWU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
43	SWU0_EN	SWU0 Event	None

Table 32-4: ADSP-CM40x SWU Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
44	SWU1_EN	SWU1 Event	None
45	SWU2_EN	SWU2 Event	None
46	SWU3_EN	SWU3 Event	None
47	SWU4_EN	SWU4 Event	None

SWU Definitions

Watchpoint Mode

Mode in which transactions are recognized on an exact match. Actions can be configured to be taken after a specified number of matches have occurred.

Bandwidth Mode

Mode in which transactions are recognized and counted inside sampling window.

SWU Architectural Concepts

The information in this section provides basic module design concepts.

SWU Flow Diagram

The *SWU Logical Flow* diagram shows the logical program flow of the SWU.

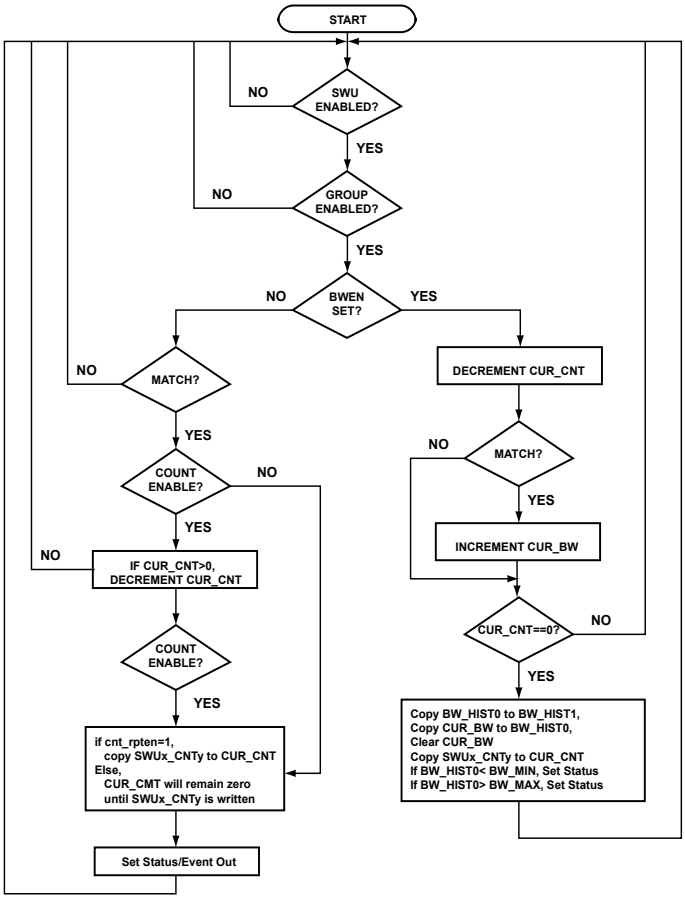

Figure 32-1: SWU Logical Flow

SWU-to-SCB Interface

The SWU system crossbar interface block latches all transactions on the system crossbar read and write address channels when the `SWU_GCTL.EN` register enable bit is set.

SWU Block Diagram

The *System Watchpoint Unit Top-Level Block Diagram* figure shows the SWU block diagram.

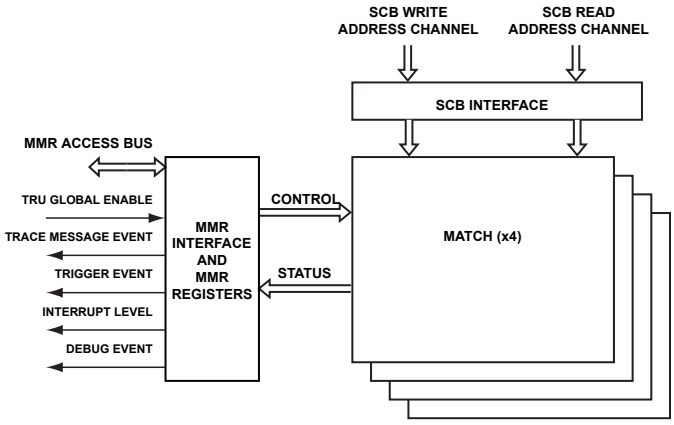

Figure 32-2: System Watchpoint Unit Top-Level Block Diagram

System Crossbar Block

The SWU system crossbar (SCB) latches all transactions on the SCB read and write address channels when the `SWU_GCTL.EN` bit is set.

MMR Block

The SWU MMR block contains the peripheral bus interface and the SWU MMR registers. It also merges all interrupts and events from each match block into common outputs.

SWU Operating Modes

There are two operating modes supported by the SWU: bandwidth mode and watchpoint mode.

Bandwidth Mode

In bandwidth mode, the SWU module counts transactions which match the properties specified in the `SWU_CTL[n]` register during a sampling window determined by the respective `SWU_CNT[n]` register. At the end of the sampling window, the SWU stores results in the `SWU_HIST[n]` register. If the sampled bandwidth falls outside a programmed range, then the programmed action occurs.

Watchpoint Mode

In watchpoint mode, if the `SWU_CTL[n].CNTEN` bit is set, the SWU module decrements the `SWU_CUR[n]` register for each match, until it equals zero, at which point any programmed actions occur. The `SWU_CUR[n]` register is then reloaded from the `SWU_CNT[n]` register (if the `SWU_CTL[n].CNTRPTEN` bit is set), and the cycle repeats. If the `SWU_CTL[n].CNTRPTEN` bit is not set, any programmed actions happen on every match.

Match Block

There are four match blocks for each SWU. Each SWU match block can monitor either the read or write address channel, selected by the `SWU_CTL[n].DIR` bit. The SWU match block can operate in either watchpoint or bandwidth mode, as selected by the `SWU_CTL[n].BWEN` bit.

In either mode, the SWU match block can be programmed to match based on address (exact, inclusive or exclusive range), ID (with masking), security, and lock type. All enabled matches are AND'ed together to determine a match.

SWU Event Control

The SWU can generate the following events when a match occurs and when the event is enabled by configuring the proper bits in the control register.

1. Trace Message
2. Trigger
3. Interrupt
4. Debug

SWU Interrupts

All interrupts and events from each match block are merged into common outputs.

SWU Status and Errors

SWU status and errors are reported in the SWU_GSTAT register. The SWU records an address error when a write or read attempt is made to the MMR address space of the SWU and the register does not exist. This error is the only one the SWU records. The register contains bits that perform the following functions.

- Indicate whether a particular match group sampled a transaction that is below a minimum target or above a maximum target in bandwidth mode.
- Indicate whether a watchpoint match occurred for each match group.
- Indicate whether an interrupt was triggered due to a match event from one of the match groups.

Triggers

The SWU can be either a trigger master or a trigger slave depending on the trigger routing unit (TRU) configuration. As a trigger master, programs must set the SWU_CTL[n].TRGEN bit so that when a match condition is met, a trigger event is generated. Each SWU in the system can also be a trigger slave when mapped as one in the TRU.

When the SWU is a slave, a trigger event activates the SWU by automatically setting the SWU_GCTL.EN bit. Since the SWU can be automatically enabled through a trigger event, programs must pre-configure the SWU before enabling the TRU. Furthermore, although a trigger event can enable the SWU as a slave, to disable the SWU, programs must manually clear the SWU_GCTL.EN bit.

SWU Programming Model

The SWU is used by programming the appropriate registers. Each control register is used to configure various aspects such as

- The direction of monitoring (reads or writes),
- Whether SWU uses bandwidth mode or watchpoint mode
- The setup of events that are triggered when a condition is met while monitoring using the SWU

Supplemental registers such as the lower and upper address boundaries are also configured before enabling.

Once the SWU has been enabled and monitoring conditions are met, events are generated if the SWU was configured to do so.

The global status register can be read to observe the status of the units.

SWU Mode Configuration

The following sections show the steps for configuring the SWU into bandwidth mode and watchpoint mode.

Configuring the SWU for Bandwidth Mode

In bandwidth mode, the SWU counts transactions which match during a sampling window. At the end of the sampling window, the SWU stores the results. An action can be taken if the sampled bandwidth goes above or falls below a programmed range.

1. Configure the `SWU_CTL[n].DIR` bit to test the match on writes or reads.
2. Configure the `SWU_CTL[n].ACMPM` bits to address comparisons, exact match, matches inside a range or matches outside a range.
3. If ID comparison is desired, set the `SWU_CTL[n].IDCMPEN` bit.
4. Set the `SWU_CTL[n].BLENINC` bit to increment by burst length or clear it to increment by 1.
5. Configure the `SWU_CTL[n].MAXACT` and `SWU_CTL[n].MINACT` bits to enable actions taken when the bandwidth goes above the maximum, or falls below the minimum, respectively.
6. Set the `SWU_CTL[n].BWEN=1` to enable bandwidth mode.
7. Program the lower address register, `SWU_LA[n]`, and upper address register, `SWU_UA[n]`, to define the memory range for comparison.
8. If ID comparison is enabled, program the ID register, `SWU_ID[n]`.
9. Program the count register, `SWU_CNT[n]`, with the number of clock cycles for which the SWU counts the number of matches.
10. If the SWU is set to respond when the bandwidth measurement underflows or overflows, program the min and max values into the `SWU_TARG[n]` register.
11. Enable the SWU

The SWU counts the number of matches in a pre-defined number of clock cycles as programmed. As an option, it can define lower and upper limits. If the matches fall outside the limits, an action can be taken.

Configuring the SWU for Watchpoint Mode

In watchpoint mode, the SWU can trigger a programmed action after every match or after a number of matches. This sequence can be automatically reset.

1. Set the SWU_CTL[n].DIR bit to test the match on writes or reads.
2. Configure the SWU_CTL[n].ACMPM bits for address comparisons, exact match, matches inside a range or matches outside a range.
3. If ID comparison is desired, set the SWU_CTL[n].IDCMPEN.
4. Set the SWU_CTL[n].CNTEN bit to enable the events to be triggered when the count decrements to zero.
5. If needed, set the SWU_CTL[n].CNTRPTEN bit to automatically reload the counter after it has decremented to zero to start another match sequence.
6. Clear the SWU_CTL[n].BWEN = 0 to configure watchpoint mode.
7. Configure the lower address register, SWU_LA[n], and upper address register, SWU_UA[n], to define the memory range for comparison.
8. If ID comparison is enabled, configure the ID register, SWU_ID[n].
9. Configure the count register, SWU_CNT[n], to determine how many matches occur before the watchpoint group responds.
10. Enable the SWU.

The SWU detects and counts down the number of match occurrences. When the counter expires, an action is taken.

ADSP-CM40x SWU Register Descriptions

System Watchpoint Unit (SWU) contains the following registers.

Table 32-5: ADSP-CM40x SWU Register List

Name	Description
SWU_CNT[n]	Count Register n
SWU_CTL[n]	Control Register n
SWU_CUR[n]	Current Register n
SWU_GCTL	Global Control Register
SWU_GSTAT	Global Status Register
SWU_HIST[n]	Bandwidth History Register n
SWU_ID[n]	ID Register n
SWU_LA[n]	Lower Address Register n

Table 32-5: ADSP-CM40x SWU Register List (Continued)

Name	Description
SWU_TARG[n]	Target Register n
SWU_UA[n]	Upper Address Register n

Count Register n

The SWU count registers (SWU_CNT[n]) contain a 16-bit count field (SWU_CNT[n].COUNT) whose usage differs depending on the mode of the watchpoint group. In bandwidth mode, the SWU_CNT[n].COUNT field value defines the number of clock cycles in a bandwidth period. In watchpoint mode, when the cycle count is enabled, the SWU_CNT[n].COUNT field value determines how many matches occur before the watchpoint group takes action.

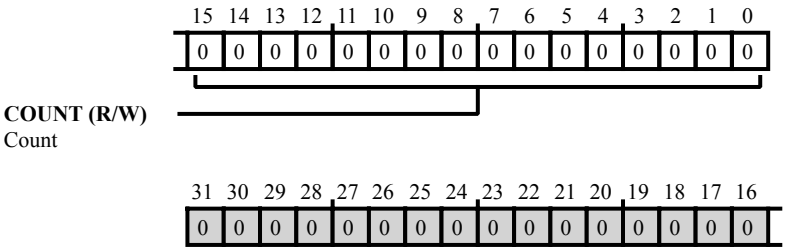

Figure 32-3: SWU_CNT[n] Register Diagram

Table 32-6: SWU_CNT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	COUNT	Count.

Control Register n

The SWU control registers (SWU_CTL[n]) contain watchpoint attribute controls for all four watchpoint groups. These controls include enabling watchpoints, selecting the transaction direction for match, selecting address comparison mode, enabling ID comparison, enabling security comparison, enabling locked comparison, enabling cycle count, enabling count repeat, enabling debug events, enabling interrupts, enabling triggers, enabling trace messages, enabling bandwidth mode, selecting the burst length increment, and enabling bandwidth underflow and overflow detection.

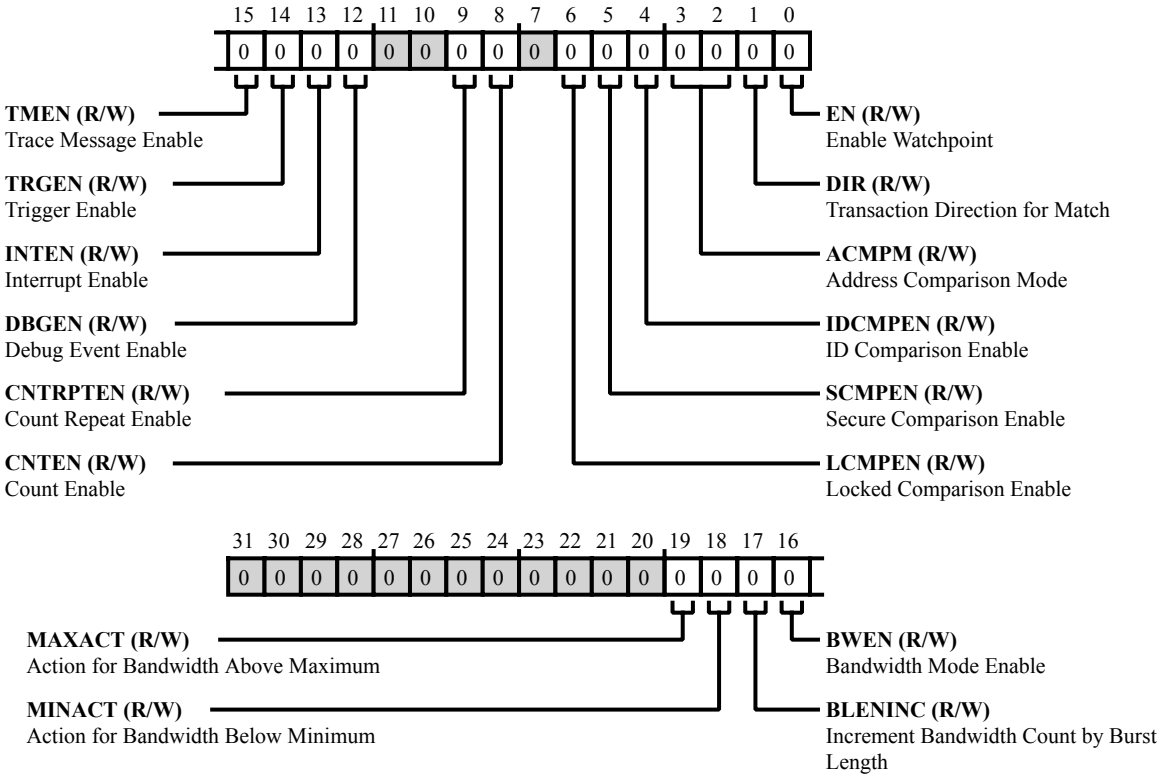

Figure 32-4: SWU_CTL[n] Register Diagram

Table 32-7: SWU_CTL[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W)	MAXACT	Action for Bandwidth Above Maximum. Each SWU_CTL[n].MAXACT bit determines whether a watchpoint group takes action on bandwidth overflow. This feature is only valid in bandwidth mode.	
		0	No Action
		1	Take Action

Table 32-7: SWU_CTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	MINACT	Action for Bandwidth Below Minimum. Each SWU_CTL[n].MINACT bit determines whether a watchpoint group takes action on bandwidth underflow. This feature is only valid in bandwidth mode.	
		0	No Action
		1	Take Action
17 (R/W)	BLENINC	Increment Bandwidth Count by Burst Length. Each SWU_CTL[n].BLENINC bit controls how a watchpoint group's bandwidth count is incremented in the SWU_CUR[n] register's SWU_CUR[n].CURBW field. If the SWU_CTL[n].BLENINC bit is cleared (= 0), the SWU increments the bandwidth count by 1 for each matching transaction. If the SWU_CTL[n].BLENINC bit is set (=1), the SWU increments the bandwidth count by the burst length of the transaction for each matching transaction. This feature is only valid for bandwidth mode (SWU_CTL[n].BWEN bit == 1). Note that if the address range match is enabled (SWU_CTL[n].ACMPM bits) and if any address of a burst falls within the address range, the SWU_CUR[n].CURBW field is incremented by the burst length even if some of the burst address fall outside of the range. Also, note that the burst size of the transaction is not included in the increment, only the burst length of the transaction. This increment operation provides an approximate (not exact) number of bus cycles consumed during the bandwidth.	
		0	Increment by 1
		1	Burst Length Increment for Bandwidth Count
16 (R/W)	BWEN	Bandwidth Mode Enable. Each SWU_CTL[n].BWEN bit controls whether a watchpoint group operates in watchpoint mode or bandwidth mode. In watchpoint mode, the SWU_CTL[n].CNTEN and (optionally) SWU_CTL[n].CNTRPTEN registers control usage of the cycle count for watchpoint group operations. In bandwidth mode, the SWU_CTL[n].BLENINC, SWU_TARG[n], and SWU_HIST[n] registers control usage of watchpoint matches for watchpoint group operations.	
		0	Watchpoint Mode
		1	Bandwidth Mode
15 (R/W)	TMEN	Trace Message Enable. Each SWU_CTL[n].TMEN bit controls whether a match for a watchpoint group generates a trace message event. This feature is valid in both bandwidth and watchpoint modes.	
		0	Disable
		1	Enable

Table 32-7: SWU_CTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	TRGEN	Trigger Enable. Each SWU_CTL[n].TRGEN bit controls whether a match for a watchpoint group generates a trigger event. This feature is valid in both bandwidth and watchpoint modes.	
		0	Disable
		1	Enable
13 (R/W)	INTEN	Interrupt Enable. Each SWU_CTL[n].INTEN bit controls whether a match for a watchpoint group generates an interrupt. This feature is valid in both bandwidth and watchpoint modes.	
		0	Disable
		1	Enable
12 (R/W)	DBGEN	Debug Event Enable. Each SWU_CTL[n].DBGEN bit controls debug event comparison for a watchpoint group, permitting matches based on debug status.	
		0	Disable
		1	Enable
9 (R/W)	CNTRPTEN	Count Repeat Enable. Each SWU_CTL[n].CNTRPTEN bit controls whether the watchpoint group's cycle count is reloaded and repeated after cycle countdown. If the SWU_CTL[n] register's SWU_CTL[n].CNTRPTEN bit is set, the SWU_CUR[n] register's SWU_CUR[n].CURCNT field is reloaded from SWU_CNT[n] register's SWU_CNT[n].COUNT field, and the countdown starts again. If SWU_CTL[n].CNTRPTEN bit is cleared, the expired count remains zero, and no further events are signalled. (See the SWU_CTL[n].CNTEN bit description for information regarding the countdown setup.)	
		0	Disable
		1	Enable
8 (R/W)	CNTEN	Count Enable. Each SWU_CTL[n].CNTEN bit controls whether the cycle count in the watchpoint group's SWU_CNT[n] register is decremented each cycle until it reaches zero. This feature is only valid in watchpoint mode (SWU_CTL[n].BWEN bit == 0).When the count reaches zero, any enabled watchpoint events are triggered. (See the SWU_CTL[n].CNTRPTEN bit description for optional actions at that may occur at the end of the countdown.)	
		0	Disable
		1	Enable

Table 32-7: SWU_CTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	LCMPEN	Locked Comparison Enable. Each SWU_CTL[n].LCMPEN bit controls locked comparison operation of an SWU watchpoint group, permitting matches based on lock status.	
		0	Match on all transaction
		1	Match only locked transactions
5 (R/W)	SCMPEN	Secure Comparison Enable. Each SWU_CTL[n].SCMPEN bit controls secure transaction comparison operation of an SWU watchpoint group, permitting matches based on transaction security.	
		0	Match on all transaction
		1	Match only secure transactions
4 (R/W)	IDCMPEN	ID Comparison Enable. Each SWU_CTL[n].IDCMPEN bit controls the ID comparison operation of an SWU watchpoint group. The ID match is based on comparison with the value in the SWU_ID[n] register.	
3:2 (R/W)	ACMPM	Address Comparison Mode. Each set of SWU_CTL[n].ACMPM bits control the address comparison operation of an SWU watchpoint group. The address within range for comparison is defined as (SWU_LA[n] register <= address < SWU_UA[n] register). The address outside range for comparison is defined as (address < SWU_LA[n]) or (SWU_UA[n] <= address).	
		0	No address comparison
		1	Exact match on LAn
		2	Match on address within range
		3	Match on address outside range
1 (R/W)	DIR	Transaction Direction for Match. Each SWU_CTL[n].DIR bit determines whether the SWU check reads or writes for watchpoint matches.	
		0	Match on reads only
		1	Match on writes only

Table 32-7: SWU_CTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Enable Watchpoint. Each `SWU_CTL[n].EN` bit controls the operation of one SWU watchpoint group. Clearing the `SWU_CTL[n].EN` bit halts the execution of watchpoint or bandwidth tracking operations in the watchpoint group without resetting status or configuration registers. Setting the `SWU_CTL[n].EN` bit enables the SWU watchpoint group to begin or resume operation with the current configuration and status.	
		0	Disable
		1	Enable

Current Register n

The SWU current register (SWU_CUR[n]) operation varies depending whether the watchpoint group is in bandwidth mode or watchpoint mode. In both modes, the watchpoint count begins when the SWU loads the register's SWU_CUR[n].CURCNT field from the SWU_CNT[n] register's SWU_CNT[n].COUNT field when the watchpoint count is enabled (SWU_CTL[n] register, SWU_CTL[n].CNTEN bit =1).

In bandwidth mode, the current count field (SWU_CUR[n].CURCNT) contains the cycle count remaining within the current watchpoint period. The SWU decrements this value every cycle until the count reaches zero. At that point, the SWU reloads the SWU_CUR[n].CURCNT field from SWU_CNT[n] register's SWU_CNT[n].COUNT field. In bandwidth mode, the current bandwidth field (SWU_CUR[n].CURBW) contains the count of watchpoint matches (bandwidth) accummulated in the current watchpoint period.

In watchpoint mode, the current count field (SWU_CUR[n].CURCNT) contains the watchpoint match count remaining within the current watchpoint period. The SWU decrements this value with every watchpoint match until the count reaches zero. At that point, the SWU reloads the SWU_CUR[n].CURCNT field from SWU_CNT[n] register's SWU_CNT[n].COUNT field if the SWU_CTL[n] register's SWU_CTL[n].CNTRPTEN bit is set (=1). In watchpoint mode, the current bandwidth field (SWU_CUR[n].CURBW) is undefined.

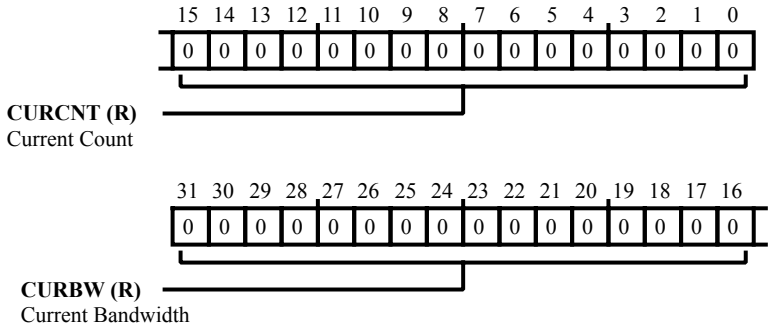

Figure 32-5: SWU_CUR[n] Register Diagram

Table 32-8: SWU_CUR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	CURBW	Current Bandwidth.
15:0 (R/NW)	CURCNT	Current Count.

Global Control Register

The SWU global control register (SWU_GCTL) provides SWU reset and enable.

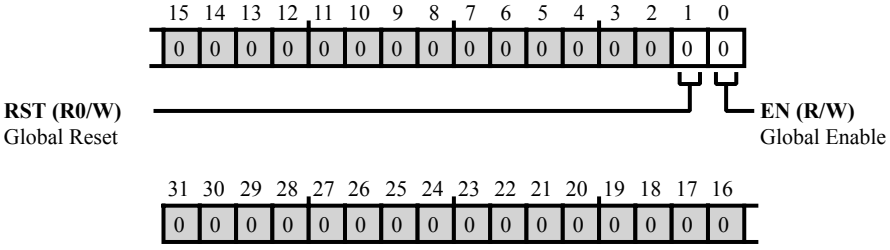

Figure 32-6: SWU_GCTL Register Diagram

Table 32-9: SWU_GCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R0/W)	RST	Global Reset. The SWU_GCTL.RST) is write-1-action/read zero and controls the SWU operational state. Setting SWU_GCTL.RST resets all SWU registers to their default values and halts all SWU operations.	
		0	No Action
		1	Reset
0 (R/W)	EN	Global Enable. The SWU_GCTL.EN controls the SWU operational state. Clearing SWU_GCTL.EN halts the execution of all watchpoint and bandwidth tracking operations without resetting status registers or associated signals. Setting SWU_GCTL.EN enables the SWU to begin/resume operation with the current configuration and status.	
		0	Disable
		1	Enable

Global Status Register

The SWU global status register (SWU_GSTAT) contains status bits for all four watchpoint groups.

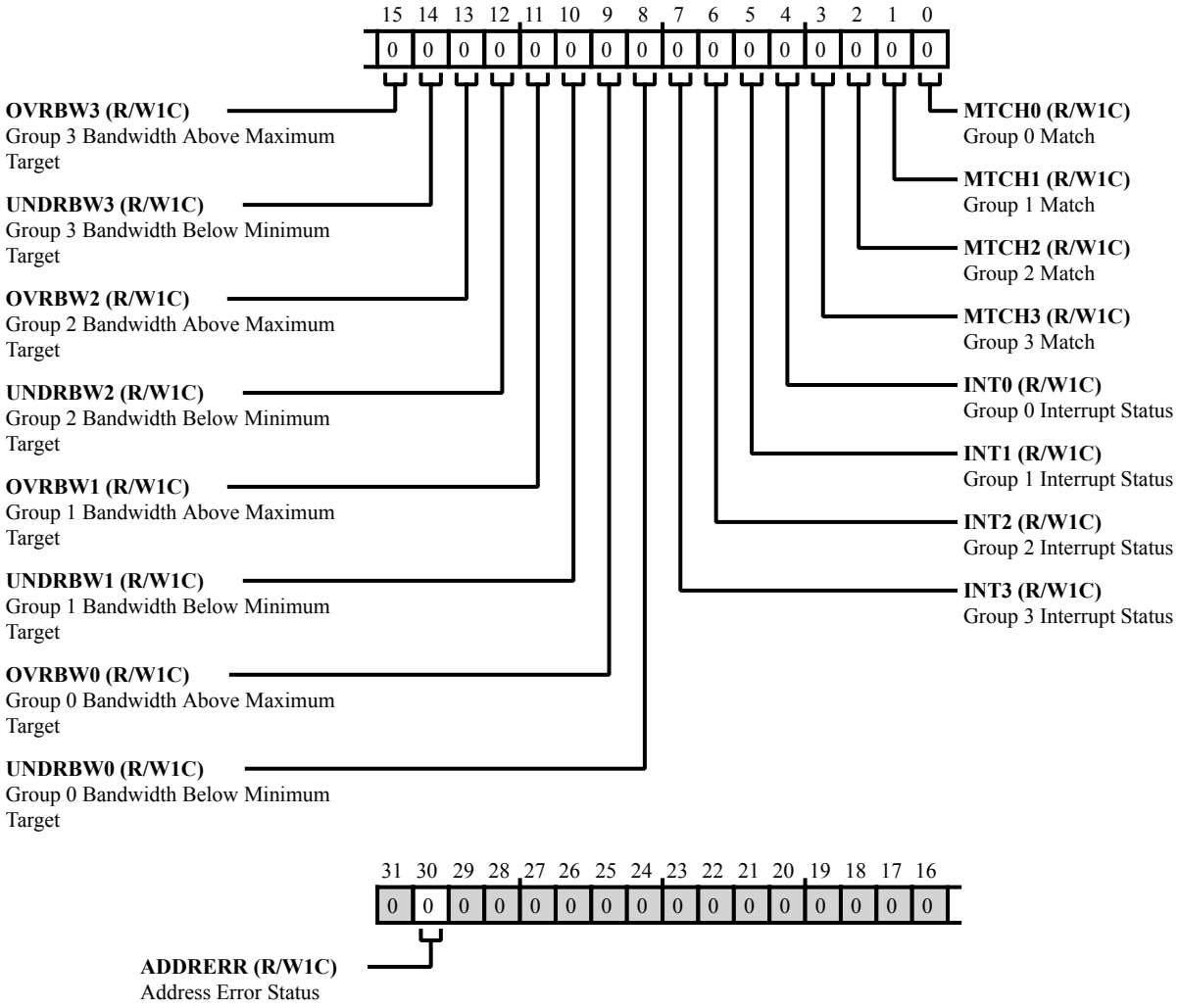

Figure 32-7: SWU_GSTAT Register Diagram

Table 32-10: SWU_GSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W1C)	ADDRERR	Address Error Status. The SWU_GSTAT.ADDRERR indicates that the SWU generated an address error. This status bit is sticky; write-1-to-clear it.	
		0	Inactive
		1	Active

Table 32-10: SWU_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	OVRBW3	Group 3 Bandwidth Above Maximum Target. See SWU_GSTAT.OVRBW0 description.	
		0	Group 3 was not above maximum bandwidth
		1	Group 3 was above maximum bandwidth
14 (R/W1C)	UNDRBW3	Group 3 Bandwidth Below Minimum Target. See SWU_GSTAT.UNDRBW0 description.	
		0	Group 3 was not below minimum bandwitdth
		1	Group 3 was below minimum bandwitdth
13 (R/W1C)	OVRBW2	Group 2 Bandwidth Above Maximum Target. See SWU_GSTAT.OVRBW0 description.	
		0	Group 2 was not above maximum bandwidth
		1	Group 2 was above maximum bandwidth
12 (R/W1C)	UNDRBW2	Group 2 Bandwidth Below Minimum Target. See SWU_GSTAT.UNDRBW0 description.	
		0	Group 2 was not below minimum bandwidth
		1	Group 2 was below minimum bandwidth
11 (R/W1C)	OVRBW1	Group 1 Bandwidth Above Maximum Target. See SWU_GSTAT.OVRBW0 description.	
		0	Group 1 was not above maximum bandwidth
		1	Group 1 was above maximum bandwidth
10 (R/W1C)	UNDRBW1	Group 1 Bandwidth Below Minimum Target. See SWU_GSTAT.UNDRBW0 description.	
		0	Group 1 was not below minimum bandwidth
		1	Group 1 was below minimum bandwidth
9 (R/W1C)	OVRBW0	Group 0 Bandwidth Above Maximum Target. The SWU_GSTAT.OVRBW0 - SWU_GSTAT.OVRBW3 -- Group 0 through 3 watchpoint bandwidth over maximum target bits. Each maximum bandwidth bit indicate (for each group)s that the measured bandwidth over the period defined by the SWU_CNT[n] register was over the maximum target. This status bit is sticky; write-1-to-clear it.	
		0	Group 0 was not above maximum bandwidth
		1	Group 0 was above maximum bandwidth

Table 32-10: SWU_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W1C)	UNDRBW0	Group 0 Bandwidth Below Minimum Target. The SWU_GSTAT.UNDRBW0 - SWU_GSTAT.UNDRBW3 -- Group 0 through 3 watchpoint bandwidth below minimum target bits. Each minimum bandwidth bit indicates (for each group) that the measured bandwidth over the period defined by the SWU_CNT[n] register was below the minimum target. This status bit is sticky; write-1-to-clear it.	
		0	Group 0 was not below minimum bandwidth
		1	Group 0 was below minimum bandwidth
7 (R/W1C)	INT3	Group 3 Interrupt Status. See SWU_GSTAT.INT0 description.	
		0	No Interrupt
		1	Interrupt Occurred
6 (R/W1C)	INT2	Group 2 Interrupt Status. See SWU_GSTAT.INT0 description.	
		0	No Interrupt
		1	Interrupt Occurred
5 (R/W1C)	INT1	Group 1 Interrupt Status. See SWU_GSTAT.INT0 description.	
		0	No Interrupt
		1	Interrupt Occurred
4 (R/W1C)	INT0	Group 0 Interrupt Status. The SWU_GSTAT.INT0 - SWU_GSTAT.INT3 -- Group 0 through 3 interrupt bits. Each interrupt bit indicates (for each group) whether a watchpoint group is contributing to the SWU's interrupt output. This status bit is sticky; write-1-to-clear it.	
		0	No interrupt
		1	Interrupt Occurred
3 (R/W1C)	MTCH3	Group 3 Match. See SWU_GSTAT.MTCH0 description.	
		0	No Match
		1	Group 3 Watchpoint Match

Table 32-10: SWU_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	MTCH2	Group 2 Match. See SWU_GSTAT.MTCH0 description.	
		0	No match
		1	Group 2 Watchpoint Match
1 (R/W1C)	MTCH1	Group 1 Match. See SWU_GSTAT.MTCH0 description.	
		0	No match
		1	Group 1 Watchpoint Match
0 (R/W1C)	MTCH0	Group 0 Match. The SWU_GSTAT.MTCH0 - SWU_GSTAT.MTCH3 -- Group 0 through 3 match bits. Each match bit indicates (for each group) whether a watchpoint match has occured in a SWU watchpoint group, as controlled by the group's related watchpoint control register (SWU_CTL[n]). This status bit is sticky; write-1-to-clear it.	
		0	No match
		1	Group 0 Watchpoint Match

Bandwidth History Register n

The SWU bandwidth history registers (SWU_HIST[n]) contain data copied from a watchpoint group's current bandwidth value (SWU_CUR[n] register, SWU_CUR[n].CURBW bits) at the end of the last two watchpoint periods. At the end of each watchpoint period, the SWU copies the previous bandwidth value from the SWU_HIST[n].BWHIST0 field to the SWU_HIST[n].BWHIST1 field and copies the new bandwidth value from the SWU_CUR[n].CURBW field to the SWU_HIST[n].BWHIST0 field.

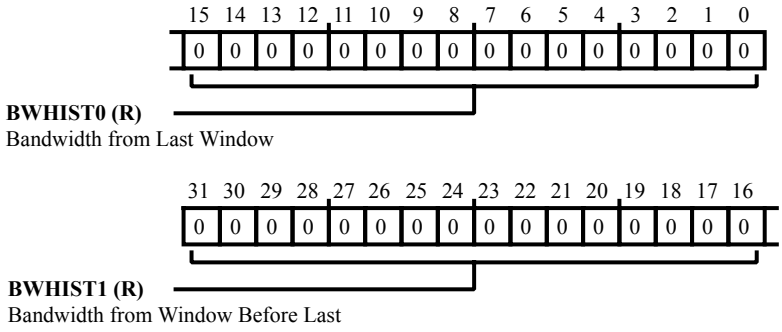

Figure 32-8: SWU_HIST[n] Register Diagram

Table 32-11: SWU_HIST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	BWHIST1	Bandwidth from Window Before Last.
15:0 (R/NW)	BWHIST0	Bandwidth from Last Window.

ID Register n

The SWU ID registers (SWU_ID[n]) contain a 16-bit ID field (SWU_ID[n].ID) and a 16-bit ID mask field (SWU_ID[n].IDMASK) that watchpoint groups use for ID comparison. The ID on the bus is AND'ed with the SWU_ID[n].IDMASK field, then the watchpoint group compares the result against the SWU_ID[n].ID field.

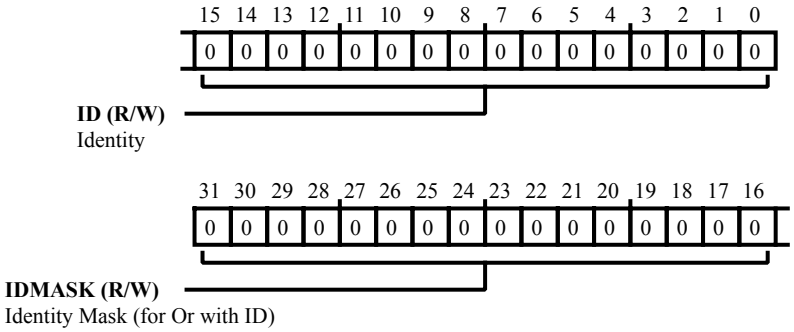

Figure 32-9: SWU_ID[n] Register Diagram

Table 32-12: SWU_ID[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	IDMASK	Identity Mask (for Or with ID).
15:0 (R/W)	ID	Identity.

Lower Address Register n

The SWU lower address registers (SWU_LA[n]) contain each watchpoint group's lower address for address match comparison. In exact match on SWU_LA[n] address mode (SWU_CTL[n].ACMPM bits =01), the watchpoint group uses only this address for match comparison.

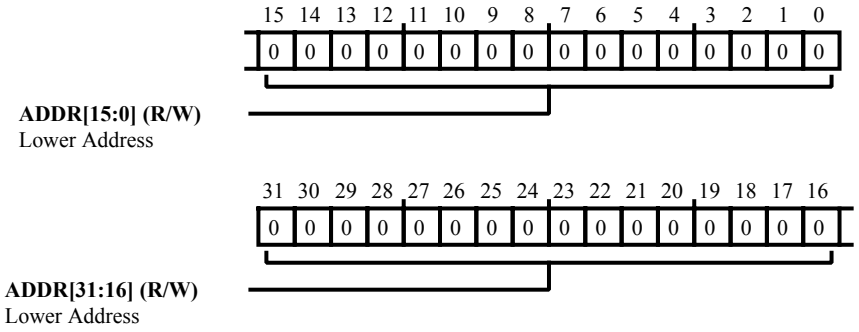

Figure 32-10: SWU_LA[n] Register Diagram

Table 32-13: SWU_LA[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDR	Lower Address.

Target Register n

The SWU target registers (SWU_TARG[n]) contain a minimum value field (SWU_TARG[n].BWMIN) and maximum value field (SWU_TARG[n].BWMAX) of bandwidth targets used by watchpoint groups in bandwidth mode. When the bandwidth period expires, if the current bandwidth value (SWU_CUR[n] register, SWU_CUR[n].CURBW bits) is below the minimum target or above the maximum target, the watchpoint group takes action as enabled by the SWU_CTL[n] register's SWU_CTL[n].MINACT or SWU_CTL[n].MAXACT bits.

In bandwidth mode, note that the watchpoint group increments its count of either data bus transactions or address bus transactions (bursts) as selected by the SWU_CTL[n].BLENINC bit. Keep this mode selection in mind when programming the bandwidth target values.

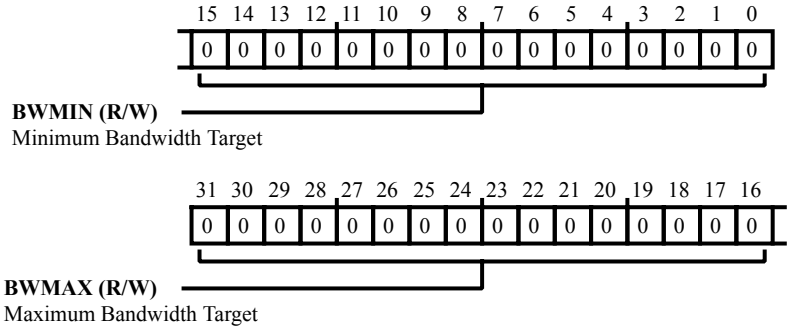

Figure 32-11: SWU_TARG[n] Register Diagram

Table 32-14: SWU_TARG[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	BWMAX	Maximum Bandwidth Target.
15:0 (R/W)	BWMIN	Minimum Bandwidth Target.

Upper Address Register n

The SWU upper address registers (SWU_UA[n]) contain each watchpoint group's upper address for address match comparison. In exact match on SWU_LA[n] address mode (SWU_CTL[n].ACMPM bits =01), the SWU_UA[n] is not used for match comparison.

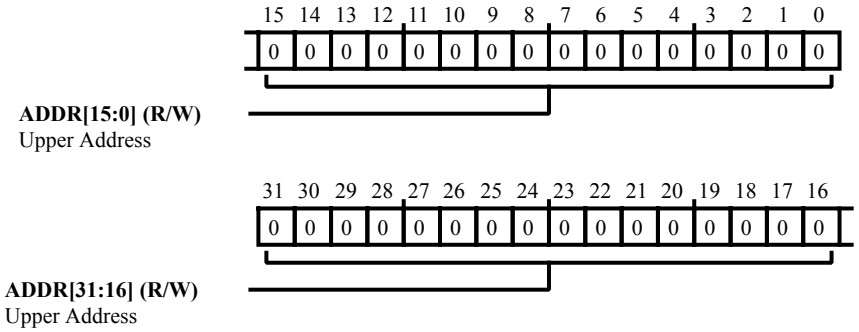

Figure 32-12: SWU_UA[n] Register Diagram

Table 32-15: SWU_UA[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDR	Upper Address.

33 JTAG debug and Serial Wire Debug Port (SWJ-DP)

SWJ-DP is a combined JTAG-DP and SW-DP that enables either a Serial Wire Debug (SWD) or JTAG probe to be connected to a target. SWD signals share pins as JTAG. There is an autodetect mechanism that switches between JTAGDP and SW-DP. Depending on which special data sequence the SWJ-DP uses, the emulator pod transmits to the JTAG pins. The SWJ-DP behaves as a JTAG target if normal JTAG sequences are sent to it and as a single wire target if the SW_DP sequence is transmitted.

Embedded Trace Macrocell (ETM) and Instrumentation Trace Macrocell (ITM)

The processor supports both Embedded Trace Macrocell (ETM) and Instrumentation Trace Macrocell (ITM). These features both offer an optional debug component that enables logging of real-time instruction and data flow within the CPU core. This data is stored and read through special debugger pods that have the trace feature capability. ITM is a single-data pin feature and the ETM is a 4-data pin feature.

ADSP-CM40x TAPC Register Descriptions

TAPC Registers (TAPC) contains the following registers.

Table 33-1: ADSP-CM40x TAPC Register List

Name	Description
TAPC_IDCODE	IDCODE Register
TAPC_RCMSG	Run Control Message Register
TAPC_RCMSG_CLR	Run Control Message Clear Register
TAPC_RCMSG_SET	Run Control Message Set Register
TAPC_RCMSG_TOG	Run Control Message Toggle Register
TAPC_SDBGKEY0	SDBGKEY0 Register
TAPC_SDBGKEY1	SDBGKEY1 Register
TAPC_SDBGKEY2	SDBGKEY2 Register
TAPC_SDBGKEY3	SDBGKEY3 Register

Table 33-1: ADSP-CM40x TAPC Register List (Continued)

Name	Description
TAPC_SDBGKEY_CTL	SDBGKEY Control Register
TAPC_SDBGKEY_STAT	SDBGKEY Status Register
TAPC_USERCODE	USERCODE Register
TAPC_USERKEYCMP0	User Key Compare0 Register
TAPC_USERKEYCMP1	User Key Compare1 Register
TAPC_USERKEYCMP2	User Key Compare2 Register
TAPC_USERKEYCMP3	User Key Compare3 Register

IDCODE Register

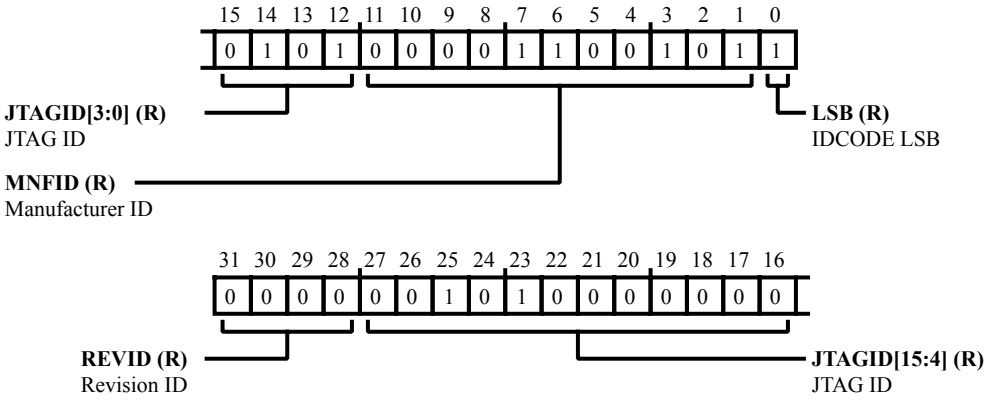

Figure 33-1: TAPC_IDCODE Register Diagram

Table 33-2: TAPC_IDCODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:28 (R/NW)	REVID	Revision ID.
27:12 (R/NW)	JTAGID	JTAG ID.
11:1 (R/NW)	MNFID	Manufacturer ID.
0 (R/NW)	LSB	IDCODE LSB.

Run Control Message Register

The bit fields of this register is defined by Boot Code

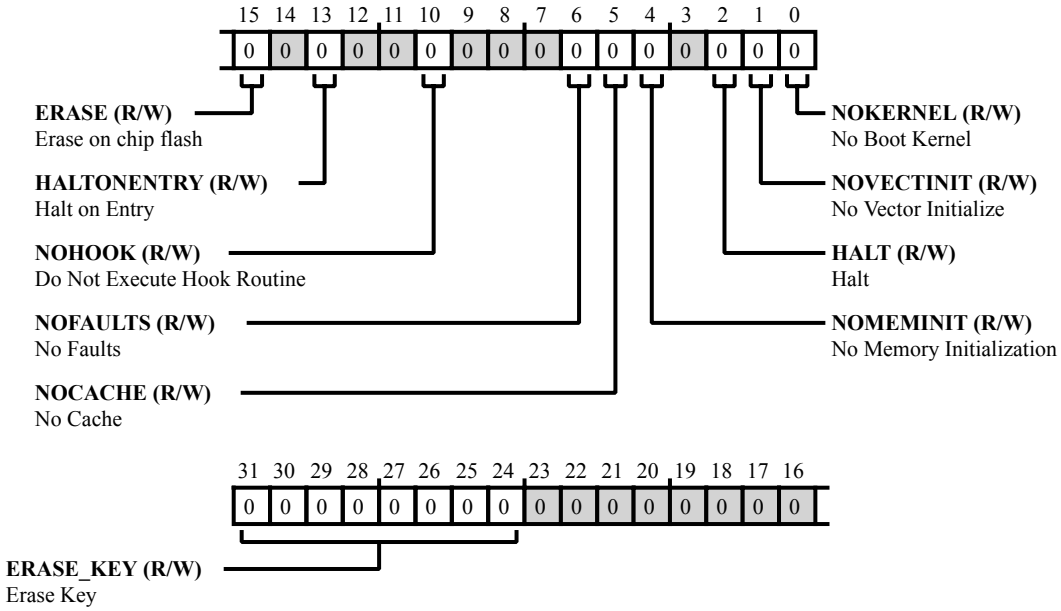

Figure 33-2: TAPC_RCMSG Register Diagram

Table 33-3: TAPC_RCMSG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:24 (R/W)	ERASE_KEY	Erase Key. Set to 0xAD to enable the RCMSG_ERASE functionality	
15 (R/W)	ERASE	Erase on chip flash. The `TAPC_RCMSG.ERASE` bit configures the Boot ROM to Erase on chip flash,when the `TAPC_RCMSG.ERASE_KEY`is set to 0xAD	
13 (R/W)	HALTONENTRY	Halt on Entry. The `TAPC_RCMSG.HALTONENTRY` bit configures the Boot ROM to enter WFI state after preboot secure header processing	
10 (R/W)	NOHOOK	Do Not Execute Hook Routine. The `TAPC_RCMSG.NOHOOK` bit configures the Boot ROM to not perform the hook routine when calling the boot mode	
		0	Perform hook routine
		1	Do not perform hook routine

Table 33-3: TAPC_RCMSG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	NOFAULTS	No Faults. The TAPC_RCMSG.NOFAULTS bit configures the Boot ROM to not perform fault initialization.	
		0	Perform fault initialization
		1	Do not perform fault initialization
5 (R/W)	NOCACHE	No Cache. The TAPC_RCMSG.NOCACHE bit configures the Boot ROM to not perform a cache initialization and to not enable the cache.	
		0	Enable and initialize cache
		1	Do not initialize or enable cache
4 (R/W)	NOMEMINIT	No Memory Initialization. The TAPC_RCMSG.NOMEMINIT bit configures the Boot ROM to not perform a memory initialization.	
		0	Perform memory initialization
		1	Do not perform memory initialization
2 (R/W)	HALT	Halt. The TAPC_RCMSG.HALT bit configures the Boot ROM to execute the no boot routine.	
		0	Do not execute routine
		1	Execute routine
1 (R/W)	NOVECTINIT	No Vector Initialize. The TAPC_RCMSG.NOVECTINIT bit configures the Boot ROM to not vector to the application.	
		0	Vector
		1	Do not vector
0 (R/W)	NOKERNEL	No Boot Kernel. The TAPC_RCMSG.NOKERNEL bit configures the Boot ROM to not execute the boot kernel.	
		0	Execute boot kernel
		1	Do not execute boot kernel

Run Control Message Clear Register

Bits in this register when set during write, clear corresponding bits in RCMSG register.

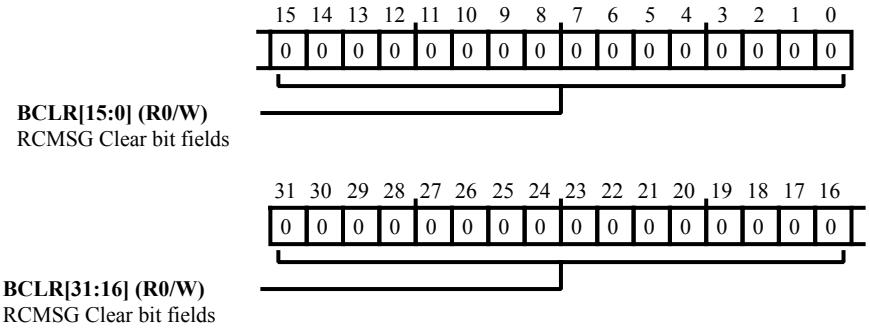

Figure 33-3: TAPC_RCMSG_CLR Register Diagram

Table 33-4: TAPC_RCMSG_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	BCLR	RCMSG Clear bit fields.

Run Control Message Set Register

Bits in this register when set during write, sets corresponding bits in RCMSG register.

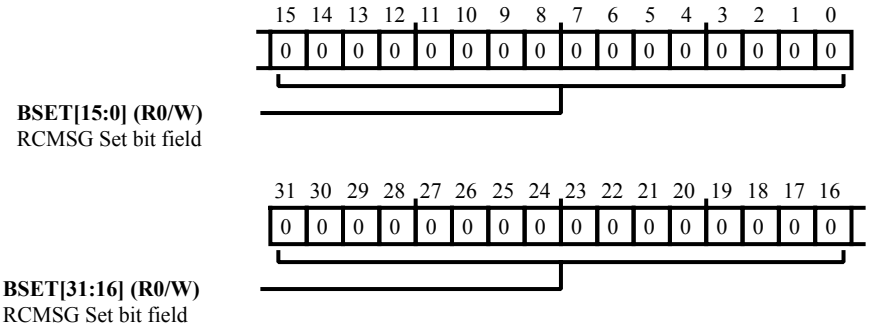

Figure 33-4: TAPC_RCMSG_SET Register Diagram

Table 33-5: TAPC_RCMSG_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	BSET	RCMSG Set bit field.

Run Control Message Toggle Register

Bits in this register when set during write, toggles corresponding bits in RCMSG register.

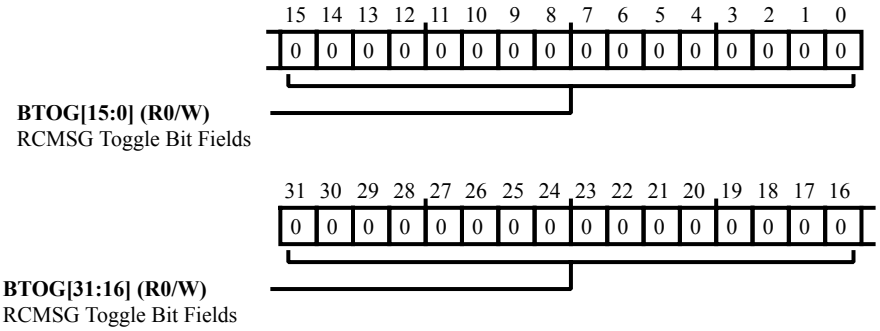

Figure 33-5: TAPC_RCMSG_TOG Register Diagram

Table 33-6: TAPC_RCMSG_TOG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	BTOG	RCMSG Toggle Bit Fields.

SDBGKEY0 Register

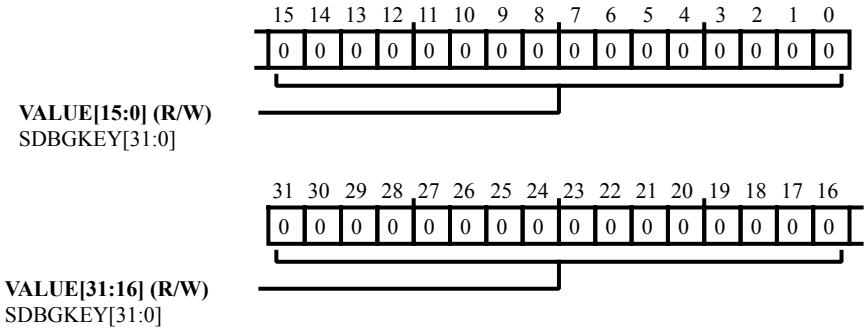

Figure 33-6: TAPC_SDBGKEY0 Register Diagram

Table 33-7: TAPC_SDBGKEY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY[31:0].

SDBGKEY1 Register

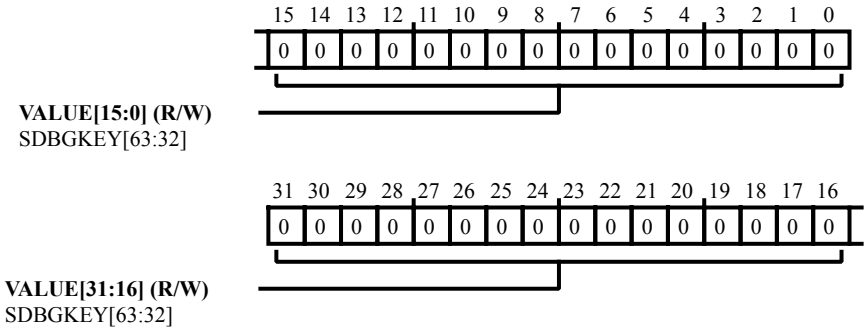

Figure 33-7: TAPC_SDBGKEY1 Register Diagram

Table 33-8: TAPC_SDBGKEY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY[63:32].

SDBGKEY2 Register

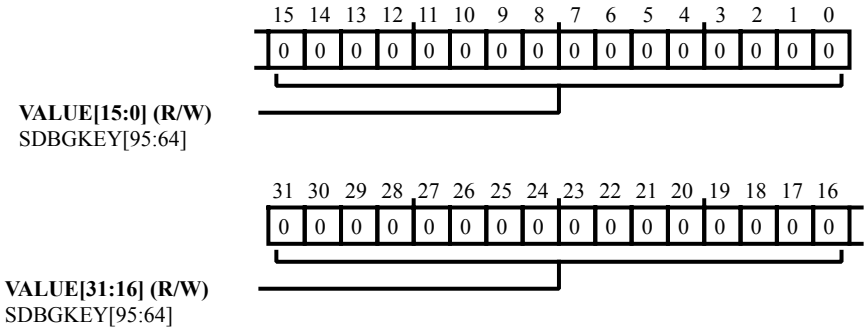

Figure 33-8: TAPC_SDBGKEY2 Register Diagram

Table 33-9: TAPC_SDBGKEY2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY[95:64].

SDBGKEY3 Register

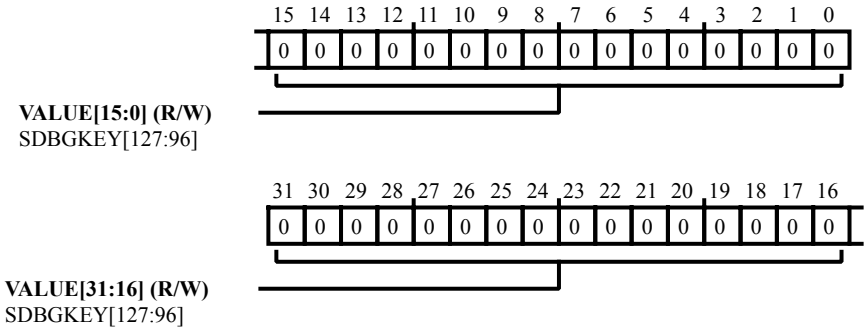

Figure 33-9: TAPC_SDBGKEY3 Register Diagram

Table 33-10: TAPC_SDBGKEY3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY[127:96].

SDBGKEY Control Register

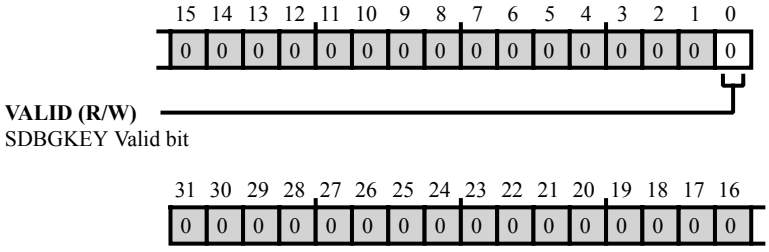

Figure 33-10: TAPC_SDBGKEY_CTL Register Diagram

Table 33-11: TAPC_SDBGKEY_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	VALID	SDBGKEY Valid bit.

SDBGKEY Status Register

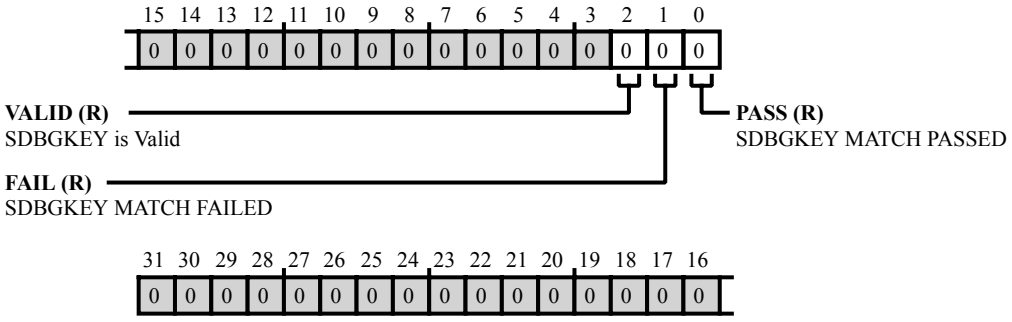

Figure 33-11: TAPC_SDBGKEY_STAT Register Diagram

Table 33-12: TAPC_SDBGKEY_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/NW)	VALID	SDBGKEY is Valid. Indicates if key present in SDBGKEY registers are valid. This is a mirror bit of SDBGKEY_CTL.VALID
1 (R/NW)	FAIL	SDBGKEY MATCH FAILED.
0 (R/NW)	PASS	SDBGKEY MATCH PASSED.

USERCODE Register

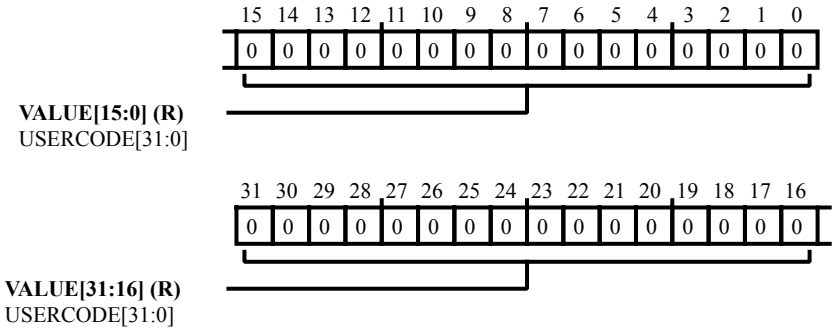

Figure 33-12: TAPC_USERCODE Register Diagram

Table 33-13: TAPC_USERCODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	USERCODE[31:0].

User Key Compare0 Register

Contains bits 31:0 of User compare key

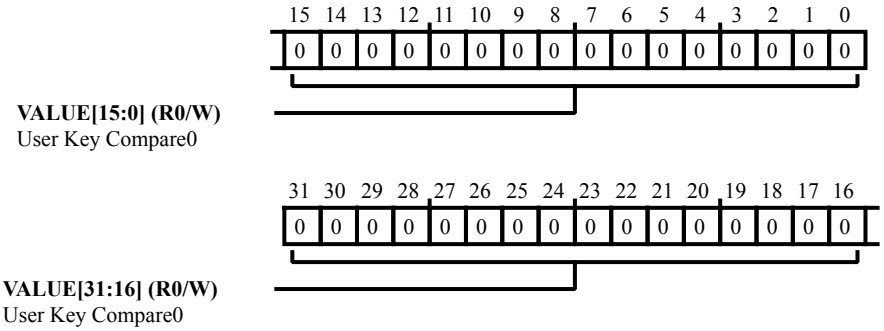

Figure 33-13: TAPC_USERKEYCMP0 Register Diagram

Table 33-14: TAPC_USERKEYCMP0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	VALUE	User Key Compare0.

User Key Compare1 Register

Contains bits 63:32 of User Compare Key

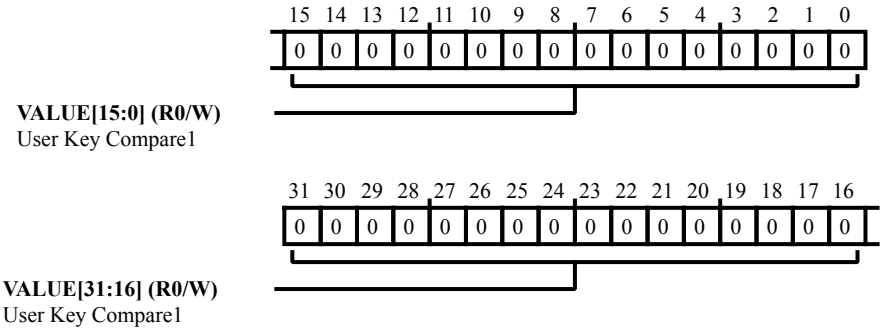

Figure 33-14: TAPC_USERKEYCMP1 Register Diagram

Table 33-15: TAPC_USERKEYCMP1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	VALUE	User Key Compare1.

User Key Compare2 Register

Contains bits 95:64 of User Compare Key

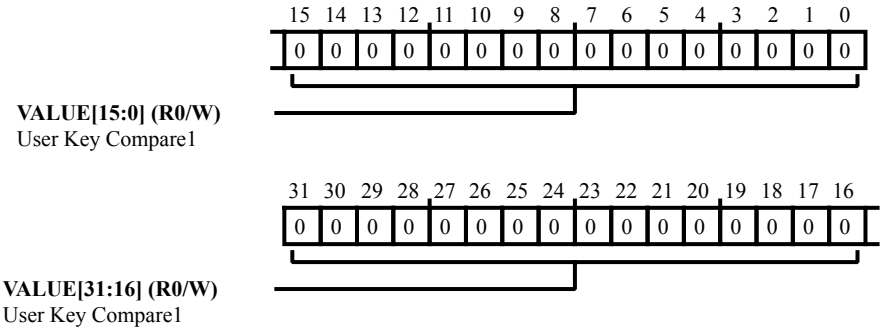

Figure 33-15: TAPC_USERKEYCMP2 Register Diagram

Table 33-16: TAPC_USERKEYCMP2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	VALUE	User Key Compare1.

User Key Compare3 Register

Contains bits 127:96 of User Compare Key

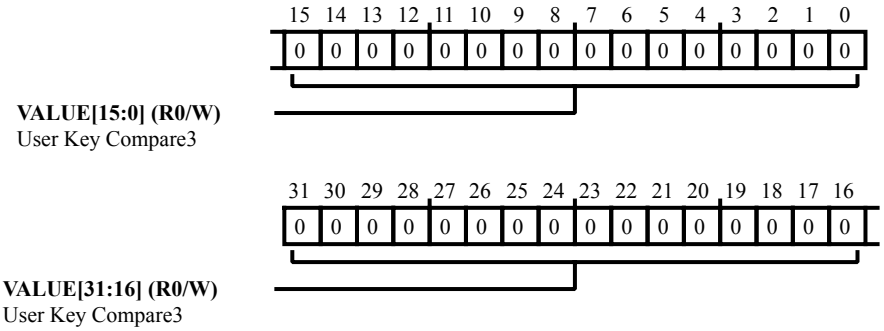

Figure 33-16: TAPC_USERKEYCMP3 Register Diagram

Table 33-17: TAPC_USERKEYCMP3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	VALUE	User Key Compare3.

Appendix A ADSP-CM40x Register List

This appendix lists Memory Mapped Register address and register names. The modules are presented in in alphabetical order.

Table A-1: ADSP-CM40x ADCC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x4000E000	ADCC0_CTL	ADCC0 Control Register	0x00000000
0x4000E004	ADCC0_ERRSTAT	ADCC0 Error Status Register	0x00000000
0x4000E008	ADCC0_ERRMSK	ADCC0 Error Mask Register	0x00000000
0x4000E00C	ADCC0_ERRMSK_SET	ADCC0 Error Mask Set Register	0x00000000
0x4000E010	ADCC0_ERRMSK_CLR	ADCC0 Error Mask Clear Register	0x00000000
0x4000E014	ADCC0_EISTAT	ADCC0 Event Interrupt Status Register	0x00000000
0x4000E018	ADCC0_EIMSK	ADCC0 Event Interrupt Mask Register	0x00000000
0x4000E01C	ADCC0_EIMSK_SET	ADCC0 Event Interrupt Mask Set Register	0x00000000
0x4000E020	ADCC0_EIMSK_CLR	ADCC0 Event Interrupt Mask Clear Register	0x00000000
0x4000E024	ADCC0_FISTAT	ADCC0 Frame Interrupt Status Register	0x00000000
0x4000E028	ADCC0_FIMSK	ADCC0 Frame Interrupt Mask Register	0x00000000
0x4000E02C	ADCC0_FIMSK_SET	ADCC0 Frame Interrupt Mask Set Register	0x00000000
0x4000E030	ADCC0_FIMSK_CLR	ADCC0 Frame Interrupt Mask Clear Register	0x00000000
0x4000E034	ADCC0_EVTEN	ADCC0 Event Enable Register	0x00000000
0x4000E038	ADCC0_EVTEN_SET	ADCC0 Event Enable Set Register	0x00000000
0x4000E03C	ADCC0_EVTEN_CLR	ADCC0 Event Enable Clear Register	0x00000000
0x4000E040	ADCC0_ECOL	ADCC0 Event Collision Status Register	0x00000000
0x4000E044	ADCC0_EMISS	ADCC0 Event Miss Status Register	0x00000000
0x4000E048	ADCC0_BPTR0	ADCC0 Base Pointer 0 Register	0x00000000
0x4000E04C	ADCC0_FRINC0	ADCC0 Frame Increment 0 Register	0x00000000
0x4000E050	ADCC0_CBSIZ0	ADCC0 Circular Buffer Size 0 Register	0x00000000
0x4000E054	ADCC0_TCA0	ADCC0 Timing Control A (ADC0) Register	0x00000000
0x4000E058	ADCC0_TCB0	ADCC0 Timing Control B (ADC0) Register	0x00000000
0x4000E05C	ADCC0_BWMON0	ADCC0 Bandwidth Monitor 0 Register	0x00000000
0x4000E060	ADCC0_CFG	ADCC0 ADC Configuration Register	0x00000000
0x4000E064	ADCC0_BPTR1	ADCC0 DMA Base Pointer 1 Register	0x00000000
0x4000E068	ADCC0_FRINC1	ADCC0 Frame Increment 1 Register	0x00000000

Table A-1: ADSP-CM40x ADCC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4000E06C	ADCC0_CBSIZ1	ADCC0 Circular Buffer Size 1 Register	0x00000000
0x4000E070	ADCC0_TCA1	ADCC0 Timing Control A (ADC1) Register	0x00000000
0x4000E074	ADCC0_TCB1	ADCC0 Timing Control B (ADC1) Register	0x00000000
0x4000E078	ADCC0_BWMON1	ADCC0 Bandwidth Monitor 1 Register	0x00000000
0x4000E07C	ADCC0_EVT00	ADCC0 Event n Time Register	0x00000000
0x4000E080	ADCC0_EVCTL00	ADCC0 Event n Control Register	0x00000000
0x4000E084	ADCC0_EVT01	ADCC0 Event n Time Register	0x00000000
0x4000E088	ADCC0_EVCTL01	ADCC0 Event n Control Register	0x00000000
0x4000E08C	ADCC0_EVT02	ADCC0 Event n Time Register	0x00000000
0x4000E090	ADCC0_EVCTL02	ADCC0 Event n Control Register	0x00000000
0x4000E094	ADCC0_EVT03	ADCC0 Event n Time Register	0x00000000
0x4000E098	ADCC0_EVCTL03	ADCC0 Event n Control Register	0x00000000
0x4000E09C	ADCC0_EVT04	ADCC0 Event n Time Register	0x00000000
0x4000E0A0	ADCC0_EVCTL04	ADCC0 Event n Control Register	0x00000000
0x4000E0A4	ADCC0_EVT05	ADCC0 Event n Time Register	0x00000000
0x4000E0A8	ADCC0_EVCTL05	ADCC0 Event n Control Register	0x00000000
0x4000E0AC	ADCC0_EVT06	ADCC0 Event n Time Register	0x00000000
0x4000E0B0	ADCC0_EVCTL06	ADCC0 Event n Control Register	0x00000000
0x4000E0B4	ADCC0_EVT07	ADCC0 Event n Time Register	0x00000000
0x4000E0B8	ADCC0_EVCTL07	ADCC0 Event n Control Register	0x00000000
0x4000E0BC	ADCC0_EVT08	ADCC0 Event n Time Register	0x00000000
0x4000E0C0	ADCC0_EVCTL08	ADCC0 Event n Control Register	0x00000000
0x4000E0C4	ADCC0_EVT09	ADCC0 Event n Time Register	0x00000000
0x4000E0C8	ADCC0_EVCTL09	ADCC0 Event n Control Register	0x00000000
0x4000E0CC	ADCC0_EVT10	ADCC0 Event n Time Register	0x00000000
0x4000E0D0	ADCC0_EVCTL10	ADCC0 Event n Control Register	0x00000000
0x4000E0D4	ADCC0_EVT11	ADCC0 Event n Time Register	0x00000000
0x4000E0D8	ADCC0_EVCTL11	ADCC0 Event n Control Register	0x00000000
0x4000E0DC	ADCC0_EVT12	ADCC0 Event n Time Register	0x00000000
0x4000E0E0	ADCC0_EVCTL12	ADCC0 Event n Control Register	0x00000000
0x4000E0E4	ADCC0_EVT13	ADCC0 Event n Time Register	0x00000000

Table A-1: ADSP-CM40x ADCC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4000E0E8	ADCC0_EVCTL13	ADCC0 Event n Control Register	0x00000000
0x4000E0EC	ADCC0_EVT14	ADCC0 Event n Time Register	0x00000000
0x4000E0F0	ADCC0_EVCTL14	ADCC0 Event n Control Register	0x00000000
0x4000E0F4	ADCC0_EVT15	ADCC0 Event n Time Register	0x00000000
0x4000E0F8	ADCC0_EVCTL15	ADCC0 Event n Control Register	0x00000000
0x4000E0FC	ADCC0_EVT16	ADCC0 Event n Time Register	0x00000000
0x4000E100	ADCC0_EVCTL16	ADCC0 Event n Control Register	0x00000000
0x4000E104	ADCC0_EVT17	ADCC0 Event n Time Register	0x00000000
0x4000E108	ADCC0_EVCTL17	ADCC0 Event n Control Register	0x00000000
0x4000E10C	ADCC0_EVT18	ADCC0 Event n Time Register	0x00000000
0x4000E110	ADCC0_EVCTL18	ADCC0 Event n Control Register	0x00000000
0x4000E114	ADCC0_EVT19	ADCC0 Event n Time Register	0x00000000
0x4000E118	ADCC0_EVCTL19	ADCC0 Event n Control Register	0x00000000
0x4000E11C	ADCC0_EVT20	ADCC0 Event n Time Register	0x00000000
0x4000E120	ADCC0_EVCTL20	ADCC0 Event n Control Register	0x00000000
0x4000E124	ADCC0_EVT21	ADCC0 Event n Time Register	0x00000000
0x4000E128	ADCC0_EVCTL21	ADCC0 Event n Control Register	0x00000000
0x4000E12C	ADCC0_EVT22	ADCC0 Event n Time Register	0x00000000
0x4000E130	ADCC0_EVCTL22	ADCC0 Event n Control Register	0x00000000
0x4000E134	ADCC0_EVT23	ADCC0 Event n Time Register	0x00000000
0x4000E138	ADCC0_EVCTL23	ADCC0 Event n Control Register	0x00000000
0x4000E200	ADCC0_EPND	ADCC0 Pending Events Status Register	0x00000000
0x4000E204	ADCC0_T0STAT	ADCC0 Timer 0 Status Register	0x00000000
0x4000E208	ADCC0_TMR0	ADCC0 Timer 0 Current Count Register	0x00000000
0x4000E20C	ADCC0_T1STAT	ADCC0 Timer 1 Status Register	0x00000000
0x4000E210	ADCC0_TMR1	ADCC0 Timer 1 Current Count Register	0x00000000
0x4000E214	ADCC0_EVDAT00	ADCC0 Event n Data Register	0x00000000
0x4000E218	ADCC0_EVDAT01	ADCC0 Event n Data Register	0x00000000
0x4000E21C	ADCC0_EVDAT02	ADCC0 Event n Data Register	0x00000000
0x4000E220	ADCC0_EVDAT03	ADCC0 Event n Data Register	0x00000000
0x4000E224	ADCC0_EVDAT04	ADCC0 Event n Data Register	0x00000000

Table A-1: ADSP-CM40x ADCC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4000E228	ADCC0_EVDAT05	ADCC0 Event n Data Register	0x00000000
0x4000E22C	ADCC0_EVDAT06	ADCC0 Event n Data Register	0x00000000
0x4000E230	ADCC0_EVDAT07	ADCC0 Event n Data Register	0x00000000
0x4000E234	ADCC0_EVDAT08	ADCC0 Event n Data Register	0x00000000
0x4000E238	ADCC0_EVDAT09	ADCC0 Event n Data Register	0x00000000
0x4000E23C	ADCC0_EVDAT10	ADCC0 Event n Data Register	0x00000000
0x4000E240	ADCC0_EVDAT11	ADCC0 Event n Data Register	0x00000000
0x4000E244	ADCC0_EVDAT12	ADCC0 Event n Data Register	0x00000000
0x4000E248	ADCC0_EVDAT13	ADCC0 Event n Data Register	0x00000000
0x4000E24C	ADCC0_EVDAT14	ADCC0 Event n Data Register	0x00000000
0x4000E250	ADCC0_EVDAT15	ADCC0 Event n Data Register	0x00000000
0x4000E254	ADCC0_EVDAT16	ADCC0 Event n Data Register	0x00000000
0x4000E258	ADCC0_EVDAT17	ADCC0 Event n Data Register	0x00000000
0x4000E25C	ADCC0_EVDAT18	ADCC0 Event n Data Register	0x00000000
0x4000E260	ADCC0_EVDAT19	ADCC0 Event n Data Register	0x00000000
0x4000E264	ADCC0_EVDAT20	ADCC0 Event n Data Register	0x00000000
0x4000E268	ADCC0_EVDAT21	ADCC0 Event n Data Register	0x00000000
0x4000E26C	ADCC0_EVDAT22	ADCC0 Event n Data Register	0x00000000
0x4000E270	ADCC0_EVDAT23	ADCC0 Event n Data Register	0x00000000
0x4000E274	ADCC0_EVSTAT00	ADCC0 Event n Status Register	0x00000000
0x4000E278	ADCC0_EVSTAT01	ADCC0 Event n Status Register	0x00000000
0x4000E27C	ADCC0_EVSTAT02	ADCC0 Event n Status Register	0x00000000
0x4000E280	ADCC0_EVSTAT03	ADCC0 Event n Status Register	0x00000000
0x4000E284	ADCC0_EVSTAT04	ADCC0 Event n Status Register	0x00000000
0x4000E288	ADCC0_EVSTAT05	ADCC0 Event n Status Register	0x00000000
0x4000E28C	ADCC0_EVSTAT06	ADCC0 Event n Status Register	0x00000000
0x4000E290	ADCC0_EVSTAT07	ADCC0 Event n Status Register	0x00000000
0x4000E294	ADCC0_EVSTAT08	ADCC0 Event n Status Register	0x00000000
0x4000E298	ADCC0_EVSTAT09	ADCC0 Event n Status Register	0x00000000
0x4000E29C	ADCC0_EVSTAT10	ADCC0 Event n Status Register	0x00000000
0x4000E2A0	ADCC0_EVSTAT11	ADCC0 Event n Status Register	0x00000000

Table A-1: ADSP-CM40x ADCC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4000E2A4	ADCC0_EVSTAT12	ADCC0 Event n Status Register	0x00000000
0x4000E2A8	ADCC0_EVSTAT13	ADCC0 Event n Status Register	0x00000000
0x4000E2AC	ADCC0_EVSTAT14	ADCC0 Event n Status Register	0x00000000
0x4000E2B0	ADCC0_EVSTAT15	ADCC0 Event n Status Register	0x00000000
0x4000E2B4	ADCC0_EVSTAT16	ADCC0 Event n Status Register	0x00000000
0x4000E2B8	ADCC0_EVSTAT17	ADCC0 Event n Status Register	0x00000000
0x4000E2BC	ADCC0_EVSTAT18	ADCC0 Event n Status Register	0x00000000
0x4000E2C0	ADCC0_EVSTAT19	ADCC0 Event n Status Register	0x00000000
0x4000E2C4	ADCC0_EVSTAT20	ADCC0 Event n Status Register	0x00000000
0x4000E2C8	ADCC0_EVSTAT21	ADCC0 Event n Status Register	0x00000000
0x4000E2CC	ADCC0_EVSTAT22	ADCC0 Event n Status Register	0x00000000
0x4000E2D0	ADCC0_EVSTAT23	ADCC0 Event n Status Register	0x00000000

Table A-2: ADSP-CM40x CAN0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40002200	CAN0_MC1	CAN0 Mailbox Configuration 1 Register	0x00000000
0x40002204	CAN0_MD1	CAN0 Mailbox Direction 1 Register	0x000000FF
0x40002208	CAN0_TRS1	CAN0 Transmission Request Set 1 Register	0x00000000
0x4000220C	CAN0_TRR1	CAN0 Transmission Request Reset 1 Register	0x00000000
0x40002210	CAN0_TA1	CAN0 Transmission Acknowledge 1 Register	0x00000000
0x40002214	CAN0_AA1	CAN0 Abort Acknowledge 1 Register	0x00000000
0x40002218	CAN0_RMP1	CAN0 Receive Message Pending 1 Register	0x00000000
0x4000221C	CAN0_RML1	CAN0 Receive Message Lost 1 Register	0x00000000
0x40002220	CAN0_MBTIF1	CAN0 Mailbox Transmit Interrupt Flag 1 Register	0x00000000
0x40002224	CAN0_MBRIF1	CAN0 Mailbox Receive Interrupt Flag 1 Register	0x00000000
0x40002228	CAN0_MBIM1	CAN0 Mailbox Interrupt Mask 1 Register	0x00000000
0x4000222C	CAN0_RFH1	CAN0 Remote Frame Handling 1 Register	0x00000000
0x40002230	CAN0_OPSS1	CAN0 Overwrite Protection/Single Shot Transmission 1 Register	0x00000000
0x40002240	CAN0_MC2	CAN0 Mailbox Configuration 2 Register	0x00000000
0x40002244	CAN0_MD2	CAN0 Mailbox Direction 2 Register	0x00000000

Table A-2: ADSP-CM40x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40002248	CAN0_TRS2	CAN0 Transmission Request Set 2 Register	0x00000000
0x4000224C	CAN0_TRR2	CAN0 Transmission Request Reset 2 Register	0x00000000
0x40002250	CAN0_TA2	CAN0 Transmission Acknowledge 2 Register	0x00000000
0x40002254	CAN0_AA2	CAN0 Abort Acknowledge 2 Register	0x00000000
0x40002258	CAN0_RMP2	CAN0 Receive Message Pending 2 Register	0x00000000
0x4000225C	CAN0_RML2	CAN0 Receive Message Lost 2 Register	0x00000000
0x40002260	CAN0_MBTIF2	CAN0 Mailbox Transmit Interrupt Flag 2 Register	0x00000000
0x40002264	CAN0_MBRIF2	CAN0 Mailbox Receive Interrupt Flag 2 Register	0x00000000
0x40002268	CAN0_MBIM2	CAN0 Mailbox Interrupt Mask 2 Register	0x00000000
0x4000226C	CAN0_RFH2	CAN0 Remote Frame Handling 2 Register	0x00000000
0x40002270	CAN0_OPSS2	CAN0 Overwrite Protection/Single Shot Transmission 2 Register	0x00000000
0x40002280	CAN0_CLK	CAN0 Clock Register	0x00000000
0x40002284	CAN0_TIMING	CAN0 Timing Register	0x00000000
0x40002288	CAN0_DBG	CAN0 Debug Register	0x00000008
0x4000228C	CAN0_STAT	CAN0 Status Register	0x00000080
0x40002290	CAN0_CEC	CAN0 Error Counter Register	0x00000000
0x40002294	CAN0_GIS	CAN0 Global CAN Interrupt Status Register	0x00000000
0x40002298	CAN0_GIM	CAN0 Global CAN Interrupt Mask Register	0x00000000
0x4000229C	CAN0_GIF	CAN0 Global CAN Interrupt Flag Register	0x00000000
0x400022A0	CAN0_CTL	CAN0 CAN Master Control Register	0x00000080
0x400022A4	CAN0_INT	CAN0 Interrupt Pending Register	0x00000000
0x400022AC	CAN0_MBTD	CAN0 Temporary Mailbox Disable Register	0x00000000
0x400022B0	CAN0_EWR	CAN0 Error Counter Warning Level Register	0x00006060
0x400022B4	CAN0_ESR	CAN0 Error Status Register	0x00000020
0x400022C4	CAN0_UCCNT	CAN0 Universal Counter Register	0x00000000
0x400022C8	CAN0_UCRC	CAN0 Universal Counter Reload/Capture Register	0x00000000
0x400022CC	CAN0_UCCNF	CAN0 Universal Counter Configuration Mode Register	0x00000000
0x40002300	CAN0_AM00L	CAN0 Acceptance Mask (L) Register	0x00000000
0x40002304	CAN0_AM00H	CAN0 Acceptance Mask (H) Register	0x00000000
0x40002308	CAN0_AM01L	CAN0 Acceptance Mask (L) Register	0x00000000

Table A-2: ADSP-CM40x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4000230C	CAN0_AM01H	CAN0 Acceptance Mask (H) Register	0x00000000
0x40002310	CAN0_AM02L	CAN0 Acceptance Mask (L) Register	0x00000000
0x40002314	CAN0_AM02H	CAN0 Acceptance Mask (H) Register	0x00000000
0x40002318	CAN0_AM03L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4000231C	CAN0_AM03H	CAN0 Acceptance Mask (H) Register	0x00000000
0x40002320	CAN0_AM04L	CAN0 Acceptance Mask (L) Register	0x00000000
0x40002324	CAN0_AM04H	CAN0 Acceptance Mask (H) Register	0x00000000
0x40002328	CAN0_AM05L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4000232C	CAN0_AM05H	CAN0 Acceptance Mask (H) Register	0x00000000
0x40002330	CAN0_AM06L	CAN0 Acceptance Mask (L) Register	0x00000000
0x40002334	CAN0_AM06H	CAN0 Acceptance Mask (H) Register	0x00000000
0x40002338	CAN0_AM07L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4000233C	CAN0_AM07H	CAN0 Acceptance Mask (H) Register	0x00000000
0x40002340	CAN0_AM08L	CAN0 Acceptance Mask (L) Register	0x00000000
0x40002344	CAN0_AM08H	CAN0 Acceptance Mask (H) Register	0x00000000
0x40002348	CAN0_AM09L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4000234C	CAN0_AM09H	CAN0 Acceptance Mask (H) Register	0x00000000
0x40002350	CAN0_AM10L	CAN0 Acceptance Mask (L) Register	0x00000000
0x40002354	CAN0_AM10H	CAN0 Acceptance Mask (H) Register	0x00000000
0x40002358	CAN0_AM11L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4000235C	CAN0_AM11H	CAN0 Acceptance Mask (H) Register	0x00000000
0x40002360	CAN0_AM12L	CAN0 Acceptance Mask (L) Register	0x00000000
0x40002364	CAN0_AM12H	CAN0 Acceptance Mask (H) Register	0x00000000
0x40002368	CAN0_AM13L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4000236C	CAN0_AM13H	CAN0 Acceptance Mask (H) Register	0x00000000
0x40002370	CAN0_AM14L	CAN0 Acceptance Mask (L) Register	0x00000000
0x40002374	CAN0_AM14H	CAN0 Acceptance Mask (H) Register	0x00000000
0x40002378	CAN0_AM15L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4000237C	CAN0_AM15H	CAN0 Acceptance Mask (H) Register	0x00000000
0x40002380	CAN0_AM16L	CAN0 Acceptance Mask (L) Register	0x00000000
0x40002384	CAN0_AM16H	CAN0 Acceptance Mask (H) Register	0x00000000

Table A-2: ADSP-CM40x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40002388	CAN0_AM17L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4000238C	CAN0_AM17H	CAN0 Acceptance Mask (H) Register	0x00000000
0x40002390	CAN0_AM18L	CAN0 Acceptance Mask (L) Register	0x00000000
0x40002394	CAN0_AM18H	CAN0 Acceptance Mask (H) Register	0x00000000
0x40002398	CAN0_AM19L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4000239C	CAN0_AM19H	CAN0 Acceptance Mask (H) Register	0x00000000
0x400023A0	CAN0_AM20L	CAN0 Acceptance Mask (L) Register	0x00000000
0x400023A4	CAN0_AM20H	CAN0 Acceptance Mask (H) Register	0x00000000
0x400023A8	CAN0_AM21L	CAN0 Acceptance Mask (L) Register	0x00000000
0x400023AC	CAN0_AM21H	CAN0 Acceptance Mask (H) Register	0x00000000
0x400023B0	CAN0_AM22L	CAN0 Acceptance Mask (L) Register	0x00000000
0x400023B4	CAN0_AM22H	CAN0 Acceptance Mask (H) Register	0x00000000
0x400023B8	CAN0_AM23L	CAN0 Acceptance Mask (L) Register	0x00000000
0x400023BC	CAN0_AM23H	CAN0 Acceptance Mask (H) Register	0x00000000
0x400023C0	CAN0_AM24L	CAN0 Acceptance Mask (L) Register	0x00000000
0x400023C4	CAN0_AM24H	CAN0 Acceptance Mask (H) Register	0x00000000
0x400023C8	CAN0_AM25L	CAN0 Acceptance Mask (L) Register	0x00000000
0x400023CC	CAN0_AM25H	CAN0 Acceptance Mask (H) Register	0x00000000
0x400023D0	CAN0_AM26L	CAN0 Acceptance Mask (L) Register	0x00000000
0x400023D4	CAN0_AM26H	CAN0 Acceptance Mask (H) Register	0x00000000
0x400023D8	CAN0_AM27L	CAN0 Acceptance Mask (L) Register	0x00000000
0x400023DC	CAN0_AM27H	CAN0 Acceptance Mask (H) Register	0x00000000
0x400023E0	CAN0_AM28L	CAN0 Acceptance Mask (L) Register	0x00000000
0x400023E4	CAN0_AM28H	CAN0 Acceptance Mask (H) Register	0x00000000
0x400023E8	CAN0_AM29L	CAN0 Acceptance Mask (L) Register	0x00000000
0x400023EC	CAN0_AM29H	CAN0 Acceptance Mask (H) Register	0x00000000
0x400023F0	CAN0_AM30L	CAN0 Acceptance Mask (L) Register	0x00000000
0x400023F4	CAN0_AM30H	CAN0 Acceptance Mask (H) Register	0x00000000
0x400023F8	CAN0_AM31L	CAN0 Acceptance Mask (L) Register	0x00000000
0x400023FC	CAN0_AM31H	CAN0 Acceptance Mask (H) Register	0x00000000
0x40002400	CAN0_MB00_DATA0	CAN0 Mailbox Word 0 Register	0x00000000

Table A-2: ADSP-CM40x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40002404	CAN0_MB00_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x40002408	CAN0_MB00_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000240C	CAN0_MB00_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x40002410	CAN0_MB00_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002414	CAN0_MB00_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x40002418	CAN0_MB00_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000241C	CAN0_MB00_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x40002420	CAN0_MB01_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x40002424	CAN0_MB01_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x40002428	CAN0_MB01_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000242C	CAN0_MB01_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x40002430	CAN0_MB01_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002434	CAN0_MB01_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x40002438	CAN0_MB01_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000243C	CAN0_MB01_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x40002440	CAN0_MB02_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x40002444	CAN0_MB02_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x40002448	CAN0_MB02_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000244C	CAN0_MB02_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x40002450	CAN0_MB02_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002454	CAN0_MB02_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x40002458	CAN0_MB02_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000245C	CAN0_MB02_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x40002460	CAN0_MB03_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x40002464	CAN0_MB03_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x40002468	CAN0_MB03_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000246C	CAN0_MB03_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x40002470	CAN0_MB03_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002474	CAN0_MB03_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x40002478	CAN0_MB03_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000247C	CAN0_MB03_ID1	CAN0 Mailbox ID 1 Register	0x00000000

Table A-2: ADSP-CM40x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40002480	CAN0_MB04_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x40002484	CAN0_MB04_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x40002488	CAN0_MB04_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000248C	CAN0_MB04_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x40002490	CAN0_MB04_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002494	CAN0_MB04_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x40002498	CAN0_MB04_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000249C	CAN0_MB04_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x400024A0	CAN0_MB05_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x400024A4	CAN0_MB05_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x400024A8	CAN0_MB05_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x400024AC	CAN0_MB05_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x400024B0	CAN0_MB05_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x400024B4	CAN0_MB05_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x400024B8	CAN0_MB05_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x400024BC	CAN0_MB05_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x400024C0	CAN0_MB06_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x400024C4	CAN0_MB06_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x400024C8	CAN0_MB06_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x400024CC	CAN0_MB06_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x400024D0	CAN0_MB06_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x400024D4	CAN0_MB06_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x400024D8	CAN0_MB06_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x400024DC	CAN0_MB06_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x400024E0	CAN0_MB07_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x400024E4	CAN0_MB07_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x400024E8	CAN0_MB07_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x400024EC	CAN0_MB07_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x400024F0	CAN0_MB07_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x400024F4	CAN0_MB07_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x400024F8	CAN0_MB07_ID0	CAN0 Mailbox ID 0 Register	0x00000000

Table A-2: ADSP-CM40x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x400024FC	CAN0_MB07_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x40002500	CAN0_MB08_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x40002504	CAN0_MB08_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x40002508	CAN0_MB08_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000250C	CAN0_MB08_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x40002510	CAN0_MB08_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002514	CAN0_MB08_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x40002518	CAN0_MB08_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000251C	CAN0_MB08_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x40002520	CAN0_MB09_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x40002524	CAN0_MB09_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x40002528	CAN0_MB09_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000252C	CAN0_MB09_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x40002530	CAN0_MB09_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002534	CAN0_MB09_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x40002538	CAN0_MB09_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000253C	CAN0_MB09_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x40002540	CAN0_MB10_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x40002544	CAN0_MB10_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x40002548	CAN0_MB10_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000254C	CAN0_MB10_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x40002550	CAN0_MB10_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002554	CAN0_MB10_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x40002558	CAN0_MB10_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000255C	CAN0_MB10_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x40002560	CAN0_MB11_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x40002564	CAN0_MB11_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x40002568	CAN0_MB11_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000256C	CAN0_MB11_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x40002570	CAN0_MB11_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002574	CAN0_MB11_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000

Table A-2: ADSP-CM40x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40002578	CAN0_MB11_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000257C	CAN0_MB11_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x40002580	CAN0_MB12_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x40002584	CAN0_MB12_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x40002588	CAN0_MB12_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000258C	CAN0_MB12_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x40002590	CAN0_MB12_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002594	CAN0_MB12_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x40002598	CAN0_MB12_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000259C	CAN0_MB12_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x400025A0	CAN0_MB13_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x400025A4	CAN0_MB13_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x400025A8	CAN0_MB13_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x400025AC	CAN0_MB13_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x400025B0	CAN0_MB13_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x400025B4	CAN0_MB13_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x400025B8	CAN0_MB13_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x400025BC	CAN0_MB13_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x400025C0	CAN0_MB14_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x400025C4	CAN0_MB14_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x400025C8	CAN0_MB14_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x400025CC	CAN0_MB14_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x400025D0	CAN0_MB14_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x400025D4	CAN0_MB14_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x400025D8	CAN0_MB14_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x400025DC	CAN0_MB14_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x400025E0	CAN0_MB15_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x400025E4	CAN0_MB15_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x400025E8	CAN0_MB15_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x400025EC	CAN0_MB15_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x400025F0	CAN0_MB15_LENGTH	CAN0 Mailbox Length Register	0x00000000

Table A-2: ADSP-CM40x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x400025F4	CAN0_MB15_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x400025F8	CAN0_MB15_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x400025FC	CAN0_MB15_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x40002600	CAN0_MB16_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x40002604	CAN0_MB16_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x40002608	CAN0_MB16_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000260C	CAN0_MB16_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x40002610	CAN0_MB16_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002614	CAN0_MB16_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x40002618	CAN0_MB16_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000261C	CAN0_MB16_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x40002620	CAN0_MB17_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x40002624	CAN0_MB17_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x40002628	CAN0_MB17_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000262C	CAN0_MB17_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x40002630	CAN0_MB17_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002634	CAN0_MB17_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x40002638	CAN0_MB17_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000263C	CAN0_MB17_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x40002640	CAN0_MB18_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x40002644	CAN0_MB18_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x40002648	CAN0_MB18_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000264C	CAN0_MB18_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x40002650	CAN0_MB18_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002654	CAN0_MB18_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x40002658	CAN0_MB18_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000265C	CAN0_MB18_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x40002660	CAN0_MB19_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x40002664	CAN0_MB19_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x40002668	CAN0_MB19_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000266C	CAN0_MB19_DATA3	CAN0 Mailbox Word 3 Register	0x00000000

Table A-2: ADSP-CM40x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40002670	CAN0_MB19_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002674	CAN0_MB19_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x40002678	CAN0_MB19_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000267C	CAN0_MB19_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x40002680	CAN0_MB20_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x40002684	CAN0_MB20_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x40002688	CAN0_MB20_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000268C	CAN0_MB20_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x40002690	CAN0_MB20_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002694	CAN0_MB20_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x40002698	CAN0_MB20_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000269C	CAN0_MB20_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x400026A0	CAN0_MB21_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x400026A4	CAN0_MB21_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x400026A8	CAN0_MB21_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x400026AC	CAN0_MB21_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x400026B0	CAN0_MB21_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x400026B4	CAN0_MB21_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x400026B8	CAN0_MB21_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x400026BC	CAN0_MB21_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x400026C0	CAN0_MB22_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x400026C4	CAN0_MB22_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x400026C8	CAN0_MB22_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x400026CC	CAN0_MB22_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x400026D0	CAN0_MB22_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x400026D4	CAN0_MB22_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x400026D8	CAN0_MB22_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x400026DC	CAN0_MB22_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x400026E0	CAN0_MB23_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x400026E4	CAN0_MB23_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x400026E8	CAN0_MB23_DATA2	CAN0 Mailbox Word 2 Register	0x00000000

Table A-2: ADSP-CM40x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x400026EC	CAN0_MB23_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x400026F0	CAN0_MB23_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x400026F4	CAN0_MB23_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x400026F8	CAN0_MB23_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x400026FC	CAN0_MB23_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x40002700	CAN0_MB24_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x40002704	CAN0_MB24_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x40002708	CAN0_MB24_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000270C	CAN0_MB24_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x40002710	CAN0_MB24_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002714	CAN0_MB24_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x40002718	CAN0_MB24_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000271C	CAN0_MB24_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x40002720	CAN0_MB25_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x40002724	CAN0_MB25_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x40002728	CAN0_MB25_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000272C	CAN0_MB25_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x40002730	CAN0_MB25_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002734	CAN0_MB25_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x40002738	CAN0_MB25_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000273C	CAN0_MB25_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x40002740	CAN0_MB26_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x40002744	CAN0_MB26_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x40002748	CAN0_MB26_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000274C	CAN0_MB26_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x40002750	CAN0_MB26_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002754	CAN0_MB26_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x40002758	CAN0_MB26_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000275C	CAN0_MB26_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x40002760	CAN0_MB27_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x40002764	CAN0_MB27_DATA1	CAN0 Mailbox Word 1 Register	0x00000000

Table A-2: ADSP-CM40x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40002768	CAN0_MB27_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000276C	CAN0_MB27_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x40002770	CAN0_MB27_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002774	CAN0_MB27_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x40002778	CAN0_MB27_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000277C	CAN0_MB27_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x40002780	CAN0_MB28_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x40002784	CAN0_MB28_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x40002788	CAN0_MB28_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4000278C	CAN0_MB28_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x40002790	CAN0_MB28_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x40002794	CAN0_MB28_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x40002798	CAN0_MB28_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4000279C	CAN0_MB28_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x400027A0	CAN0_MB29_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x400027A4	CAN0_MB29_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x400027A8	CAN0_MB29_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x400027AC	CAN0_MB29_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x400027B0	CAN0_MB29_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x400027B4	CAN0_MB29_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x400027B8	CAN0_MB29_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x400027BC	CAN0_MB29_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x400027C0	CAN0_MB30_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x400027C4	CAN0_MB30_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x400027C8	CAN0_MB30_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x400027CC	CAN0_MB30_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x400027D0	CAN0_MB30_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x400027D4	CAN0_MB30_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x400027D8	CAN0_MB30_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x400027DC	CAN0_MB30_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x400027E0	CAN0_MB31_DATA0	CAN0 Mailbox Word 0 Register	0x00000000

Table A-2: ADSP-CM40x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x400027E4	CAN0_MB31_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x400027E8	CAN0_MB31_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x400027EC	CAN0_MB31_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x400027F0	CAN0_MB31_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x400027F4	CAN0_MB31_TIMESTAMP	CAN0 Mailbox Timestamp Register	0x00000000
0x400027F8	CAN0_MB31_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x400027FC	CAN0_MB31_ID1	CAN0 Mailbox ID 1 Register	0x00000000

Table A-3: ADSP-CM40x CAN1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40002A00	CAN1_MC1	CAN1 Mailbox Configuration 1 Register	0x00000000
0x40002A04	CAN1_MD1	CAN1 Mailbox Direction 1 Register	0x000000FF
0x40002A08	CAN1_TRS1	CAN1 Transmission Request Set 1 Register	0x00000000
0x40002A0C	CAN1_TRR1	CAN1 Transmission Request Reset 1 Register	0x00000000
0x40002A10	CAN1_TA1	CAN1 Transmission Acknowledge 1 Register	0x00000000
0x40002A14	CAN1_AA1	CAN1 Abort Acknowledge 1 Register	0x00000000
0x40002A18	CAN1_RMP1	CAN1 Receive Message Pending 1 Register	0x00000000
0x40002A1C	CAN1_RML1	CAN1 Receive Message Lost 1 Register	0x00000000
0x40002A20	CAN1_MBTIF1	CAN1 Mailbox Transmit Interrupt Flag 1 Register	0x00000000
0x40002A24	CAN1_MBRIF1	CAN1 Mailbox Receive Interrupt Flag 1 Register	0x00000000
0x40002A28	CAN1_MBIM1	CAN1 Mailbox Interrupt Mask 1 Register	0x00000000
0x40002A2C	CAN1_RFH1	CAN1 Remote Frame Handling 1 Register	0x00000000
0x40002A30	CAN1_OPSS1	CAN1 Overwrite Protection/Single Shot Transmission 1 Register	0x00000000
0x40002A40	CAN1_MC2	CAN1 Mailbox Configuration 2 Register	0x00000000
0x40002A44	CAN1_MD2	CAN1 Mailbox Direction 2 Register	0x00000000
0x40002A48	CAN1_TRS2	CAN1 Transmission Request Set 2 Register	0x00000000
0x40002A4C	CAN1_TRR2	CAN1 Transmission Request Reset 2 Register	0x00000000
0x40002A50	CAN1_TA2	CAN1 Transmission Acknowledge 2 Register	0x00000000
0x40002A54	CAN1_AA2	CAN1 Abort Acknowledge 2 Register	0x00000000
0x40002A58	CAN1_RMP2	CAN1 Receive Message Pending 2 Register	0x00000000

Table A-3: ADSP-CM40x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40002A5C	CAN1_RML2	CAN1 Receive Message Lost 2 Register	0x00000000
0x40002A60	CAN1_MBTIF2	CAN1 Mailbox Transmit Interrupt Flag 2 Register	0x00000000
0x40002A64	CAN1_MBRIF2	CAN1 Mailbox Receive Interrupt Flag 2 Register	0x00000000
0x40002A68	CAN1_MBIM2	CAN1 Mailbox Interrupt Mask 2 Register	0x00000000
0x40002A6C	CAN1_RFH2	CAN1 Remote Frame Handling 2 Register	0x00000000
0x40002A70	CAN1_OPSS2	CAN1 Overwrite Protection/Single Shot Transmission 2 Register	0x00000000
0x40002A80	CAN1_CLK	CAN1 Clock Register	0x00000000
0x40002A84	CAN1_TIMING	CAN1 Timing Register	0x00000000
0x40002A88	CAN1_DBG	CAN1 Debug Register	0x00000008
0x40002A8C	CAN1_STAT	CAN1 Status Register	0x00000080
0x40002A90	CAN1_CEC	CAN1 Error Counter Register	0x00000000
0x40002A94	CAN1_GIS	CAN1 Global CAN Interrupt Status Register	0x00000000
0x40002A98	CAN1_GIM	CAN1 Global CAN Interrupt Mask Register	0x00000000
0x40002A9C	CAN1_GIF	CAN1 Global CAN Interrupt Flag Register	0x00000000
0x40002AA0	CAN1_CTL	CAN1 CAN Master Control Register	0x00000080
0x40002AA4	CAN1_INT	CAN1 Interrupt Pending Register	0x00000000
0x40002AAC	CAN1_MBTD	CAN1 Temporary Mailbox Disable Register	0x00000000
0x40002AB0	CAN1_EWR	CAN1 Error Counter Warning Level Register	0x00006060
0x40002AB4	CAN1_ESR	CAN1 Error Status Register	0x00000020
0x40002AC4	CAN1_UCCNT	CAN1 Universal Counter Register	0x00000000
0x40002AC8	CAN1_UCRC	CAN1 Universal Counter Reload/Capture Register	0x00000000
0x40002ACC	CAN1_UCCNF	CAN1 Universal Counter Configuration Mode Register	0x00000000
0x40002B00	CAN1_AM00L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002B04	CAN1_AM00H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002B08	CAN1_AM01L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002B0C	CAN1_AM01H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002B10	CAN1_AM02L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002B14	CAN1_AM02H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002B18	CAN1_AM03L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002B1C	CAN1_AM03H	CAN1 Acceptance Mask (H) Register	0x00000000

Table A-3: ADSP-CM40x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40002B20	CAN1_AM04L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002B24	CAN1_AM04H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002B28	CAN1_AM05L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002B2C	CAN1_AM05H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002B30	CAN1_AM06L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002B34	CAN1_AM06H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002B38	CAN1_AM07L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002B3C	CAN1_AM07H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002B40	CAN1_AM08L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002B44	CAN1_AM08H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002B48	CAN1_AM09L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002B4C	CAN1_AM09H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002B50	CAN1_AM10L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002B54	CAN1_AM10H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002B58	CAN1_AM11L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002B5C	CAN1_AM11H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002B60	CAN1_AM12L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002B64	CAN1_AM12H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002B68	CAN1_AM13L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002B6C	CAN1_AM13H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002B70	CAN1_AM14L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002B74	CAN1_AM14H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002B78	CAN1_AM15L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002B7C	CAN1_AM15H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002B80	CAN1_AM16L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002B84	CAN1_AM16H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002B88	CAN1_AM17L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002B8C	CAN1_AM17H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002B90	CAN1_AM18L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002B94	CAN1_AM18H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002B98	CAN1_AM19L	CAN1 Acceptance Mask (L) Register	0x00000000

Table A-3: ADSP-CM40x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40002B9C	CAN1_AM19H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002BA0	CAN1_AM20L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002BA4	CAN1_AM20H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002BA8	CAN1_AM21L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002BAC	CAN1_AM21H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002BB0	CAN1_AM22L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002BB4	CAN1_AM22H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002BB8	CAN1_AM23L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002BBC	CAN1_AM23H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002BC0	CAN1_AM24L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002BC4	CAN1_AM24H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002BC8	CAN1_AM25L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002BCC	CAN1_AM25H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002BD0	CAN1_AM26L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002BD4	CAN1_AM26H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002BD8	CAN1_AM27L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002BDC	CAN1_AM27H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002BE0	CAN1_AM28L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002BE4	CAN1_AM28H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002BE8	CAN1_AM29L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002BEC	CAN1_AM29H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002BF0	CAN1_AM30L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002BF4	CAN1_AM30H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002BF8	CAN1_AM31L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40002BFC	CAN1_AM31H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40002C00	CAN1_MB00_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002C04	CAN1_MB00_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002C08	CAN1_MB00_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002C0C	CAN1_MB00_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002C10	CAN1_MB00_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002C14	CAN1_MB00_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000

Table A-3: ADSP-CM40x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40002C18	CAN1_MB00_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002C1C	CAN1_MB00_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002C20	CAN1_MB01_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002C24	CAN1_MB01_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002C28	CAN1_MB01_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002C2C	CAN1_MB01_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002C30	CAN1_MB01_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002C34	CAN1_MB01_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002C38	CAN1_MB01_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002C3C	CAN1_MB01_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002C40	CAN1_MB02_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002C44	CAN1_MB02_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002C48	CAN1_MB02_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002C4C	CAN1_MB02_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002C50	CAN1_MB02_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002C54	CAN1_MB02_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002C58	CAN1_MB02_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002C5C	CAN1_MB02_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002C60	CAN1_MB03_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002C64	CAN1_MB03_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002C68	CAN1_MB03_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002C6C	CAN1_MB03_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002C70	CAN1_MB03_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002C74	CAN1_MB03_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002C78	CAN1_MB03_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002C7C	CAN1_MB03_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002C80	CAN1_MB04_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002C84	CAN1_MB04_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002C88	CAN1_MB04_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002C8C	CAN1_MB04_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002C90	CAN1_MB04_LENGTH	CAN1 Mailbox Length Register	0x00000000

Table A-3: ADSP-CM40x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40002C94	CAN1_MB04_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002C98	CAN1_MB04_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002C9C	CAN1_MB04_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002CA0	CAN1_MB05_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002CA4	CAN1_MB05_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002CA8	CAN1_MB05_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002CAC	CAN1_MB05_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002CB0	CAN1_MB05_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002CB4	CAN1_MB05_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002CB8	CAN1_MB05_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002CBC	CAN1_MB05_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002CC0	CAN1_MB06_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002CC4	CAN1_MB06_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002CC8	CAN1_MB06_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002CCC	CAN1_MB06_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002CD0	CAN1_MB06_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002CD4	CAN1_MB06_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002CD8	CAN1_MB06_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002CDC	CAN1_MB06_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002CE0	CAN1_MB07_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002CE4	CAN1_MB07_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002CE8	CAN1_MB07_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002CEC	CAN1_MB07_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002CF0	CAN1_MB07_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002CF4	CAN1_MB07_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002CF8	CAN1_MB07_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002CFC	CAN1_MB07_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002D00	CAN1_MB08_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002D04	CAN1_MB08_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002D08	CAN1_MB08_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002D0C	CAN1_MB08_DATA3	CAN1 Mailbox Word 3 Register	0x00000000

Table A-3: ADSP-CM40x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40002D10	CAN1_MB08_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002D14	CAN1_MB08_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002D18	CAN1_MB08_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002D1C	CAN1_MB08_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002D20	CAN1_MB09_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002D24	CAN1_MB09_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002D28	CAN1_MB09_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002D2C	CAN1_MB09_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002D30	CAN1_MB09_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002D34	CAN1_MB09_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002D38	CAN1_MB09_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002D3C	CAN1_MB09_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002D40	CAN1_MB10_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002D44	CAN1_MB10_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002D48	CAN1_MB10_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002D4C	CAN1_MB10_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002D50	CAN1_MB10_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002D54	CAN1_MB10_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002D58	CAN1_MB10_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002D5C	CAN1_MB10_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002D60	CAN1_MB11_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002D64	CAN1_MB11_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002D68	CAN1_MB11_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002D6C	CAN1_MB11_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002D70	CAN1_MB11_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002D74	CAN1_MB11_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002D78	CAN1_MB11_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002D7C	CAN1_MB11_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002D80	CAN1_MB12_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002D84	CAN1_MB12_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002D88	CAN1_MB12_DATA2	CAN1 Mailbox Word 2 Register	0x00000000

Table A-3: ADSP-CM40x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40002D8C	CAN1_MB12_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002D90	CAN1_MB12_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002D94	CAN1_MB12_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002D98	CAN1_MB12_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002D9C	CAN1_MB12_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002DA0	CAN1_MB13_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002DA4	CAN1_MB13_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002DA8	CAN1_MB13_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002DAC	CAN1_MB13_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002DB0	CAN1_MB13_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002DB4	CAN1_MB13_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002DB8	CAN1_MB13_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002DBC	CAN1_MB13_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002DC0	CAN1_MB14_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002DC4	CAN1_MB14_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002DC8	CAN1_MB14_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002DCC	CAN1_MB14_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002DD0	CAN1_MB14_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002DD4	CAN1_MB14_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002DD8	CAN1_MB14_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002DDC	CAN1_MB14_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002DE0	CAN1_MB15_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002DE4	CAN1_MB15_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002DE8	CAN1_MB15_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002DEC	CAN1_MB15_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002DF0	CAN1_MB15_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002DF4	CAN1_MB15_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002DF8	CAN1_MB15_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002DFC	CAN1_MB15_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002E00	CAN1_MB16_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002E04	CAN1_MB16_DATA1	CAN1 Mailbox Word 1 Register	0x00000000

Table A-3: ADSP-CM40x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40002E08	CAN1_MB16_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002E0C	CAN1_MB16_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002E10	CAN1_MB16_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002E14	CAN1_MB16_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002E18	CAN1_MB16_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002E1C	CAN1_MB16_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002E20	CAN1_MB17_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002E24	CAN1_MB17_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002E28	CAN1_MB17_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002E2C	CAN1_MB17_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002E30	CAN1_MB17_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002E34	CAN1_MB17_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002E38	CAN1_MB17_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002E3C	CAN1_MB17_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002E40	CAN1_MB18_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002E44	CAN1_MB18_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002E48	CAN1_MB18_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002E4C	CAN1_MB18_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002E50	CAN1_MB18_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002E54	CAN1_MB18_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002E58	CAN1_MB18_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002E5C	CAN1_MB18_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002E60	CAN1_MB19_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002E64	CAN1_MB19_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002E68	CAN1_MB19_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002E6C	CAN1_MB19_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002E70	CAN1_MB19_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002E74	CAN1_MB19_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002E78	CAN1_MB19_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002E7C	CAN1_MB19_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002E80	CAN1_MB20_DATA0	CAN1 Mailbox Word 0 Register	0x00000000

Table A-3: ADSP-CM40x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40002E84	CAN1_MB20_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002E88	CAN1_MB20_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002E8C	CAN1_MB20_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002E90	CAN1_MB20_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002E94	CAN1_MB20_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002E98	CAN1_MB20_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002E9C	CAN1_MB20_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002EA0	CAN1_MB21_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002EA4	CAN1_MB21_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002EA8	CAN1_MB21_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002EAC	CAN1_MB21_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002EB0	CAN1_MB21_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002EB4	CAN1_MB21_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002EB8	CAN1_MB21_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002EBC	CAN1_MB21_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002EC0	CAN1_MB22_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002EC4	CAN1_MB22_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002EC8	CAN1_MB22_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002ECC	CAN1_MB22_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002ED0	CAN1_MB22_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002ED4	CAN1_MB22_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002ED8	CAN1_MB22_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002EDC	CAN1_MB22_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002EE0	CAN1_MB23_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002EE4	CAN1_MB23_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002EE8	CAN1_MB23_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002EEC	CAN1_MB23_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002EF0	CAN1_MB23_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002EF4	CAN1_MB23_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002EF8	CAN1_MB23_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002EFC	CAN1_MB23_ID1	CAN1 Mailbox ID 1 Register	0x00000000

Table A-3: ADSP-CM40x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40002F00	CAN1_MB24_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002F04	CAN1_MB24_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002F08	CAN1_MB24_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002F0C	CAN1_MB24_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002F10	CAN1_MB24_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002F14	CAN1_MB24_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002F18	CAN1_MB24_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002F1C	CAN1_MB24_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002F20	CAN1_MB25_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002F24	CAN1_MB25_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002F28	CAN1_MB25_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002F2C	CAN1_MB25_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002F30	CAN1_MB25_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002F34	CAN1_MB25_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002F38	CAN1_MB25_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002F3C	CAN1_MB25_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002F40	CAN1_MB26_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002F44	CAN1_MB26_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002F48	CAN1_MB26_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002F4C	CAN1_MB26_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002F50	CAN1_MB26_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002F54	CAN1_MB26_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002F58	CAN1_MB26_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002F5C	CAN1_MB26_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002F60	CAN1_MB27_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002F64	CAN1_MB27_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002F68	CAN1_MB27_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002F6C	CAN1_MB27_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002F70	CAN1_MB27_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002F74	CAN1_MB27_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002F78	CAN1_MB27_ID0	CAN1 Mailbox ID 0 Register	0x00000000

Table A-3: ADSP-CM40x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40002F7C	CAN1_MB27_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002F80	CAN1_MB28_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002F84	CAN1_MB28_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002F88	CAN1_MB28_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002F8C	CAN1_MB28_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002F90	CAN1_MB28_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002F94	CAN1_MB28_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002F98	CAN1_MB28_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002F9C	CAN1_MB28_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002FA0	CAN1_MB29_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002FA4	CAN1_MB29_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002FA8	CAN1_MB29_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002FAC	CAN1_MB29_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002FB0	CAN1_MB29_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002FB4	CAN1_MB29_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002FB8	CAN1_MB29_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002FBC	CAN1_MB29_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002FC0	CAN1_MB30_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002FC4	CAN1_MB30_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002FC8	CAN1_MB30_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002FCC	CAN1_MB30_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002FD0	CAN1_MB30_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002FD4	CAN1_MB30_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000
0x40002FD8	CAN1_MB30_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002FDC	CAN1_MB30_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40002FE0	CAN1_MB31_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40002FE4	CAN1_MB31_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40002FE8	CAN1_MB31_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x40002FEC	CAN1_MB31_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40002FF0	CAN1_MB31_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40002FF4	CAN1_MB31_TIMESTAMP	CAN1 Mailbox Timestamp Register	0x00000000

Table A-3: ADSP-CM40x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40002FF8	CAN1_MB31_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x40002FFC	CAN1_MB31_ID1	CAN1 Mailbox ID 1 Register	0x00000000

Table A-4: ADSP-CM40x CGU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40016000	CGU0_CTL	CGU0 Control Register	0x00001000
0x40016008	CGU0_STAT	CGU0 Status Register	0x0000031D
0x4001600C	CGU0_DIV	CGU0 Clocks Divisor Register	0x04082824
0x40016010	CGU0_CLKOUTSEL	CGU0 CLKOUT Select Register	0x00000000
0x40016014	CGU0_OSCWDCTL	CGU0 Oscillator Watchdog Register	0x00007600
0x40016018	CGU0_TSCTL	CGU0 Timestamp Control Register	0x00000000
0x4001601C	CGU0_TSVALUE0	CGU0 Timestamp Counter Initial 32 LSB Value Register	0x00000000
0x40016020	CGU0_TSVALUE1	CGU0 Time Stamp Counter Initial MSB Value Register	0x00000000
0x40016024	CGU0_TSCOUNT0	CGU0 Time Stamp Counter 32 LSB Register	0x00000000
0x40016028	CGU0_TSCOUNT1	CGU0 Time Stamp Counter 32 MSB Register	0x00000000

Table A-5: ADSP-CM40x CNT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40009000	CNT0_CFG	CNT0 Configuration Register	0x00000000
0x40009004	CNT0_IMSK	CNT0 Interrupt Mask Register	0x00000000
0x40009008	CNT0_STAT	CNT0 Status Register	0x00000000
0x4000900C	CNT0_CMD	CNT0 Command Register	0x00000000
0x40009010	CNT0_DEBNCE	CNT0 Debounce Register	0x00000000
0x40009014	CNT0_CNTR	CNT0 Counter Register	0x00000000
0x40009018	CNT0_MAX	CNT0 Maximum Count Register	0x00000000
0x4000901C	CNT0_MIN	CNT0 Minimum Count Register	0x00000000
0x40009020	CNT0_MDIV	CNT0 M Value for Divider	0x00000000
0x40009024	CNT0_NDIV	CNT0 N Value for Divider	0x00000000

Table A-6: ADSP-CM40x CNT1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x4000A000	CNT1_CFG	CNT1 Configuration Register	0x00000000
0x4000A004	CNT1_IMSK	CNT1 Interrupt Mask Register	0x00000000
0x4000A008	CNT1_STAT	CNT1 Status Register	0x00000000
0x4000A00C	CNT1_CMD	CNT1 Command Register	0x00000000
0x4000A010	CNT1_DEBNCE	CNT1 Debounce Register	0x00000000
0x4000A014	CNT1_CNTR	CNT1 Counter Register	0x00000000
0x4000A018	CNT1_MAX	CNT1 Maximum Count Register	0x00000000
0x4000A01C	CNT1_MIN	CNT1 Minimum Count Register	0x00000000
0x4000A020	CNT1_MDIV	CNT1 M Value for Divider	0x00000000
0x4000A024	CNT1_NDIV	CNT1 N Value for Divider	0x00000000

Table A-7: ADSP-CM40x CNT2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x4000B000	CNT2_CFG	CNT2 Configuration Register	0x00000000
0x4000B004	CNT2_IMSK	CNT2 Interrupt Mask Register	0x00000000
0x4000B008	CNT2_STAT	CNT2 Status Register	0x00000000
0x4000B00C	CNT2_CMD	CNT2 Command Register	0x00000000
0x4000B010	CNT2_DEBNCE	CNT2 Debounce Register	0x00000000
0x4000B014	CNT2_CNTR	CNT2 Counter Register	0x00000000
0x4000B018	CNT2_MAX	CNT2 Maximum Count Register	0x00000000
0x4000B01C	CNT2_MIN	CNT2 Minimum Count Register	0x00000000
0x4000B020	CNT2_MDIV	CNT2 M Value for Divider	0x00000000
0x4000B024	CNT2_NDIV	CNT2 N Value for Divider	0x00000000

Table A-8: ADSP-CM40x CNT3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x4000C000	CNT3_CFG	CNT3 Configuration Register	0x00000000
0x4000C004	CNT3_IMSK	CNT3 Interrupt Mask Register	0x00000000
0x4000C008	CNT3_STAT	CNT3 Status Register	0x00000000
0x4000C00C	CNT3_CMD	CNT3 Command Register	0x00000000

Table A-8: ADSP-CM40x CNT3 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4000C010	CNT3_DEBNCE	CNT3 Debounce Register	0x00000000
0x4000C014	CNT3_CNTR	CNT3 Counter Register	0x00000000
0x4000C018	CNT3_MAX	CNT3 Maximum Count Register	0x00000000
0x4000C01C	CNT3_MIN	CNT3 Minimum Count Register	0x00000000
0x4000C020	CNT3_MDIV	CNT3 M Value for Divider	0x00000000
0x4000C024	CNT3_NDIV	CNT3 N Value for Divider	0x00000000

Table A-9: ADSP-CM40x CPTMR0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x4002E004	CPTMR0_RUN	CPTMR0 Run Register	0x00000000
0x4002E008	CPTMR0_RUN_SET	CPTMR0 Run Set	0x00000000
0x4002E00C	CPTMR0_RUN_CLR	CPTMR0 Run Clear	0x00000000
0x4002E010	CPTMR0_DATA_IMSK	CPTMR0 Data Interrupt Mask Register	0x00000007
0x4002E014	CPTMR0_DATA_IMSK_SET	CPTMR0 Data Interrupt Mask Set Register	0x00000007
0x4002E018	CPTMR0_DATA_IMSK_CLR	CPTMR0 Data Interrupt Mask Clear Register	0x00000007
0x4002E01C	CPTMR0_STAT_IMSK	CPTMR0 Status Interrupt Mask Register	0x00000007
0x4002E020	CPTMR0_STAT_IMSK_SET	CPTMR0 Status Interrupt Mask Set Register	0x00000007
0x4002E024	CPTMR0_STAT_IMSK_CLR	CPTMR0 Status Interrupt Mask Clear Register	0x00000007
0x4002E028	CPTMR0_DATA_ILAT	CPTMR0 Data Interrupt Latch Status Register	0x00000000
0x4002E02C	CPTMR0_STAT_ILAT	CPTMR0 Interrupt Latch Status Register	0x00000000
0x4002E800	CPTMR0_CFG0	CPTMR0 Configuration Register	0x00000000
0x4002E804	CPTMR0_CNT0	CPTMR0 Counter Register	0x00000001
0x4002E808	CPTMR0_TON0	CPTMR0 On Time Capture Register	0x00000000
0x4002E880	CPTMR0_CFG1	CPTMR0 Configuration Register	0x00000000
0x4002E884	CPTMR0_CNT1	CPTMR0 Counter Register	0x00000001
0x4002E888	CPTMR0_TON1	CPTMR0 On Time Capture Register	0x00000000
0x4002E900	CPTMR0_CFG2	CPTMR0 Configuration Register	0x00000000
0x4002E904	CPTMR0_CNT2	CPTMR0 Counter Register	0x00000001

Table A-9: ADSP-CM40x CPTMR0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4002E908	CPTMR0_TON2	CPTMR0 On Time Capture Register	0x00000000

Table A-10: ADSP-CM40x CRC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40001C00	CRC0_CTL	CRC0 Control Register	0x00000000
0x40001C04	CRC0_DCNT	CRC0 Data Word Count Register	0x00000000
0x40001C08	CRC0_DCNTRLD	CRC0 Data Word Count Reload Register	0x00000000
0x40001C14	CRC0_COMP	CRC0 Data Compare Register	0x00000000
0x40001C18	CRC0_FILLVAL	CRC0 Fill Value Register	0x00000000
0x40001C1C	CRC0_DFIFO	CRC0 Data FIFO Register	0x00000000
0x40001C20	CRC0_INEN	CRC0 Interrupt Enable Register	0x00000000
0x40001C24	CRC0_INEN_SET	CRC0 Interrupt Enable Set Register	0x00000000
0x40001C28	CRC0_INEN_CLR	CRC0 Interrupt Enable Clear Register	0x00000000
0x40001C2C	CRC0_POLY	CRC0 Polynomial Register	0x00000000
0x40001C40	CRC0_STAT	CRC0 Status Register	0x00000000
0x40001C44	CRC0_DCNTCAP	CRC0 Data Count Capture Register	0x00000000
0x40001C4C	CRC0_RESULT_FIN	CRC0 CRC Final Result Register	0x00000000
0x40001C50	CRC0_RESULT_CUR	CRC0 CRC Current Result Register	0x00000000

Table A-11: ADSP-CM40x DACC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x4000F000	DACC0_CTL0	DACC0 Control 0 Register	0x00000000
0x4000F004	DACC0_CTL1	DACC0 Control 1 Register	0x00000000
0x4000F008	DACC0_ERRSTAT	DACC0 Error Status Register	0x00000000
0x4000F00C	DACC0_ERRMSK	DACC0 Error Mask Register	0x00000000
0x4000F010	DACC0_ERRMSK_SET	DACC0 Error Mask Set Register	0x00000000
0x4000F014	DACC0_ERRMSK_CLR	DACC0 Error Mask Clear Register	0x00000000
0x4000F018	DACC0_ISTAT	DACC0 Interrupt Status Register	0x00000000
0x4000F01C	DACC0_IMSK	DACC0 Interrupt Mask Register	0x00000000
0x4000F020	DACC0_IMSK_SET	DACC0 Interrupt Mask Set Register	0x00000000

Table A-11: ADSP-CM40x DACC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4000F024	DACC0_IMSK_CLR	DACC0 Interrupt Mask Clear Register	0x00000000
0x4000F028	DACC0_TC0	DACC0 Timing Control 0 Register	0x00000000
0x4000F02C	DACC0_BPTR0	DACC0 Base Pointer 0 Register	0x00000000
0x4000F030	DACC0_MOD0	DACC0 Modify 0 Register	0x00000000
0x4000F034	DACC0_CNT0	DACC0 Count 0 Register	0x00000000
0x4000F038	DACC0_DAT0	DACC0 Data FIFO 0 Register	0x00000000
0x4000F03C	DACC0_TC1	DACC0 Timing Control 1 Register	0x00000000
0x4000F040	DACC0_BPTR1	DACC0 Base Pointer 1 Register	0x00000000
0x4000F044	DACC0_MOD1	DACC0 Modify 1 Register	0x00000000
0x4000F048	DACC0_CNT1	DACC0 Count 1 Register	0x00000000
0x4000F04C	DACC0_DAT1	DACC0 Data FIFO 1 Register	0x00000000
0x4000F050	DACC0_BCST_CTL	DACC0 Broadcast (Write) Control Register	0x00000000
0x4000F100	DACC0_CNTCUR0	DACC0 Current Count 0 Register	0x00000000
0x4000F104	DACC0_CNTCUR1	DACC0 Current Count 1 Register	0x00000000
0x4000F108	DACC0_STAT	DACC0 Status Register	0x00000000

Table A-12: ADSP-CM40x DMA0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40011000	DMA6_DSCPTR_NXT	DMA0 Pointer to Next Initial Descriptor Register	0x00000000
0x40011004	DMA6_ADDRSTART	DMA0 Start Address of Current Buffer Register	0x00000000
0x40011008	DMA6_CFG	DMA0 Configuration Register	0x00000000
0x4001100C	DMA6_XCNT	DMA0 Inner Loop Count Start Value Register	0x00000000
0x40011010	DMA6_XMOD	DMA0 Inner Loop Address Increment Register	0x00000000
0x40011014	DMA6_YCNT	DMA0 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40011018	DMA6_YMOD	DMA0 Outer Loop Address Increment (2D only) Register	0x00000000
0x40011024	DMA6_DSCPTR_CUR	DMA0 Current Descriptor Pointer Register	0x00000000
0x40011028	DMA6_DSCPTR_PRV	DMA0 Previous Initial Descriptor Pointer Register	0x00000000
0x4001102C	DMA6_ADDR_CUR	DMA0 Current Address Register	0x00000000
0x40011030	DMA6_STAT	DMA0 Status Register	0x00006000
0x40011034	DMA6_XCNT_CUR	DMA0 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000

Table A-12: ADSP-CM40x DMA0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40011038	DMA6_YCNT_CUR	DMA0 Current Row Count (2D only) Register	0x00000000
0x40011040	DMA6_BWLCNT	DMA0 Bandwidth Limit Count Register	0x00000000
0x40011044	DMA6_BWLCNT_CUR	DMA0 Bandwidth Limit Count Current Register	0x00000000
0x40011048	DMA6_BWMCNT	DMA0 Bandwidth Monitor Count Register	0x00000000
0x4001104C	DMA6_BWMCNT_CUR	DMA0 Bandwidth Monitor Count Current Register	0x00000000

Table A-13: ADSP-CM40x DMA1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40011080	DMA7_DSCPTR_NXT	DMA1 Pointer to Next Initial Descriptor Register	0x00000000
0x40011084	DMA7_ADDRSTART	DMA1 Start Address of Current Buffer Register	0x00000000
0x40011088	DMA7_CFG	DMA1 Configuration Register	0x00000000
0x4001108C	DMA7_XCNT	DMA1 Inner Loop Count Start Value Register	0x00000000
0x40011090	DMA7_XMOD	DMA1 Inner Loop Address Increment Register	0x00000000
0x40011094	DMA7_YCNT	DMA1 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40011098	DMA7_YMOD	DMA1 Outer Loop Address Increment (2D only) Register	0x00000000
0x400110A4	DMA7_DSCPTR_CUR	DMA1 Current Descriptor Pointer Register	0x00000000
0x400110A8	DMA7_DSCPTR_PRV	DMA1 Previous Initial Descriptor Pointer Register	0x00000000
0x400110AC	DMA7_ADDR_CUR	DMA1 Current Address Register	0x00000000
0x400110B0	DMA7_STAT	DMA1 Status Register	0x00006000
0x400110B4	DMA7_XCNT_CUR	DMA1 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000
0x400110B8	DMA7_YCNT_CUR	DMA1 Current Row Count (2D only) Register	0x00000000
0x400110C0	DMA7_BWLCNT	DMA1 Bandwidth Limit Count Register	0x00000000
0x400110C4	DMA7_BWLCNT_CUR	DMA1 Bandwidth Limit Count Current Register	0x00000000
0x400110C8	DMA7_BWMCNT	DMA1 Bandwidth Monitor Count Register	0x00000000
0x400110CC	DMA7_BWMCNT_CUR	DMA1 Bandwidth Monitor Count Current Register	0x00000000

Table A-14: ADSP-CM40x DMA10 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40010000	DMA0_DSCPTR_NXT	DMA10 Pointer to Next Initial Descriptor Register	0x00000000

Table A-14: ADSP-CM40x DMA10 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40010004	DMA0_ADDRSTART	DMA10 Start Address of Current Buffer Register	0x00000000
0x40010008	DMA0_CFG	DMA10 Configuration Register	0x00000000
0x4001000C	DMA0_XCNT	DMA10 Inner Loop Count Start Value Register	0x00000000
0x40010010	DMA0_XMOD	DMA10 Inner Loop Address Increment Register	0x00000000
0x40010014	DMA0_YCNT	DMA10 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40010018	DMA0_YMOD	DMA10 Outer Loop Address Increment (2D only) Register	0x00000000
0x40010024	DMA0_DSCPTR_CUR	DMA10 Current Descriptor Pointer Register	0x00000000
0x40010028	DMA0_DSCPTR_PRV	DMA10 Previous Initial Descriptor Pointer Register	0x00000000
0x4001002C	DMA0_ADDR_CUR	DMA10 Current Address Register	0x00000000
0x40010030	DMA0_STAT	DMA10 Status Register	0x00006000
0x40010034	DMA0_XCNT_CUR	DMA10 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000
0x40010038	DMA0_YCNT_CUR	DMA10 Current Row Count (2D only) Register	0x00000000
0x40010040	DMA0_BWLCNT	DMA10 Bandwidth Limit Count Register	0x00000000
0x40010044	DMA0_BWLCNT_CUR	DMA10 Bandwidth Limit Count Current Register	0x00000000
0x40010048	DMA0_BWMCNT	DMA10 Bandwidth Monitor Count Register	0x00000000
0x4001004C	DMA0_BWMCNT_CUR	DMA10 Bandwidth Monitor Count Current Register	0x00000000

Table A-15: ADSP-CM40x DMA11 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40010080	DMA1_DSCPTR_NXT	DMA11 Pointer to Next Initial Descriptor Register	0x00000000
0x40010084	DMA1_ADDRSTART	DMA11 Start Address of Current Buffer Register	0x00000000
0x40010088	DMA1_CFG	DMA11 Configuration Register	0x00000000
0x4001008C	DMA1_XCNT	DMA11 Inner Loop Count Start Value Register	0x00000000
0x40010090	DMA1_XMOD	DMA11 Inner Loop Address Increment Register	0x00000000
0x40010094	DMA1_YCNT	DMA11 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40010098	DMA1_YMOD	DMA11 Outer Loop Address Increment (2D only) Register	0x00000000
0x400100A4	DMA1_DSCPTR_CUR	DMA11 Current Descriptor Pointer Register	0x00000000

Table A-15: ADSP-CM40x DMA11 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x400100A8	DMA1_DSCPTR_PRV	DMA11 Previous Initial Descriptor Pointer Register	0x00000000
0x400100AC	DMA1_ADDR_CUR	DMA11 Current Address Register	0x00000000
0x400100B0	DMA1_STAT	DMA11 Status Register	0x00006000
0x400100B4	DMA1_XCNT_CUR	DMA11 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000
0x400100B8	DMA1_YCNT_CUR	DMA11 Current Row Count (2D only) Register	0x00000000
0x400100C0	DMA1_BWLCNT	DMA11 Bandwidth Limit Count Register	0x00000000
0x400100C4	DMA1_BWLCNT_CUR	DMA11 Bandwidth Limit Count Current Register	0x00000000
0x400100C8	DMA1_BWMCNT	DMA11 Bandwidth Monitor Count Register	0x00000000
0x400100CC	DMA1_BWMCNT_CUR	DMA11 Bandwidth Monitor Count Current Register	0x00000000

Table A-16: ADSP-CM40x DMA12 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40010100	DMA2_DSCPTR_NXT	DMA12 Pointer to Next Initial Descriptor Register	0x00000000
0x40010104	DMA2_ADDRSTART	DMA12 Start Address of Current Buffer Register	0x00000000
0x40010108	DMA2_CFG	DMA12 Configuration Register	0x00000000
0x4001010C	DMA2_XCNT	DMA12 Inner Loop Count Start Value Register	0x00000000
0x40010110	DMA2_XMOD	DMA12 Inner Loop Address Increment Register	0x00000000
0x40010114	DMA2_YCNT	DMA12 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40010118	DMA2_YMOD	DMA12 Outer Loop Address Increment (2D only) Register	0x00000000
0x40010124	DMA2_DSCPTR_CUR	DMA12 Current Descriptor Pointer Register	0x00000000
0x40010128	DMA2_DSCPTR_PRV	DMA12 Previous Initial Descriptor Pointer Register	0x00000000
0x4001012C	DMA2_ADDR_CUR	DMA12 Current Address Register	0x00000000
0x40010130	DMA2_STAT	DMA12 Status Register	0x00006000
0x40010134	DMA2_XCNT_CUR	DMA12 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000
0x40010138	DMA2_YCNT_CUR	DMA12 Current Row Count (2D only) Register	0x00000000
0x40010140	DMA2_BWLCNT	DMA12 Bandwidth Limit Count Register	0x00000000
0x40010144	DMA2_BWLCNT_CUR	DMA12 Bandwidth Limit Count Current Register	0x00000000
0x40010148	DMA2_BWMCNT	DMA12 Bandwidth Monitor Count Register	0x00000000

Table A-16: ADSP-CM40x DMA12 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4001014C	DMA2_BWMCNT_CUR	DMA12 Bandwidth Monitor Count Current Register	0x00000000

Table A-17: ADSP-CM40x DMA13 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40010180	DMA3_DSCPTR_NXT	DMA13 Pointer to Next Initial Descriptor Register	0x00000000
0x40010184	DMA3_ADDRSTART	DMA13 Start Address of Current Buffer Register	0x00000000
0x40010188	DMA3_CFG	DMA13 Configuration Register	0x00000000
0x4001018C	DMA3_XCNT	DMA13 Inner Loop Count Start Value Register	0x00000000
0x40010190	DMA3_XMOD	DMA13 Inner Loop Address Increment Register	0x00000000
0x40010194	DMA3_YCNT	DMA13 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40010198	DMA3_YMOD	DMA13 Outer Loop Address Increment (2D only) Register	0x00000000
0x400101A4	DMA3_DSCPTR_CUR	DMA13 Current Descriptor Pointer Register	0x00000000
0x400101A8	DMA3_DSCPTR_PRV	DMA13 Previous Initial Descriptor Pointer Register	0x00000000
0x400101AC	DMA3_ADDR_CUR	DMA13 Current Address Register	0x00000000
0x400101B0	DMA3_STAT	DMA13 Status Register	0x00006000
0x400101B4	DMA3_XCNT_CUR	DMA13 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000
0x400101B8	DMA3_YCNT_CUR	DMA13 Current Row Count (2D only) Register	0x00000000
0x400101C0	DMA3_BWLCNT	DMA13 Bandwidth Limit Count Register	0x00000000
0x400101C4	DMA3_BWLCNT_CUR	DMA13 Bandwidth Limit Count Current Register	0x00000000
0x400101C8	DMA3_BWMCNT	DMA13 Bandwidth Monitor Count Register	0x00000000
0x400101CC	DMA3_BWMCNT_CUR	DMA13 Bandwidth Monitor Count Current Register	0x00000000

Table A-18: ADSP-CM40x DMA14 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40010200	DMA4_DSCPTR_NXT	DMA14 Pointer to Next Initial Descriptor Register	0x00000000
0x40010204	DMA4_ADDRSTART	DMA14 Start Address of Current Buffer Register	0x00000000
0x40010208	DMA4_CFG	DMA14 Configuration Register	0x00000000
0x4001020C	DMA4_XCNT	DMA14 Inner Loop Count Start Value Register	0x00000000

Table A-18: ADSP-CM40x DMA14 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40010210	DMA4_XMOD	DMA14 Inner Loop Address Increment Register	0x00000000
0x40010214	DMA4_YCNT	DMA14 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40010218	DMA4_YMOD	DMA14 Outer Loop Address Increment (2D only) Register	0x00000000
0x40010224	DMA4_DSCPTR_CUR	DMA14 Current Descriptor Pointer Register	0x00000000
0x40010228	DMA4_DSCPTR_PRV	DMA14 Previous Initial Descriptor Pointer Register	0x00000000
0x4001022C	DMA4_ADDR_CUR	DMA14 Current Address Register	0x00000000
0x40010230	DMA4_STAT	DMA14 Status Register	0x00006000
0x40010234	DMA4_XCNT_CUR	DMA14 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000
0x40010238	DMA4_YCNT_CUR	DMA14 Current Row Count (2D only) Register	0x00000000
0x40010240	DMA4_BWLCNT	DMA14 Bandwidth Limit Count Register	0x00000000
0x40010244	DMA4_BWLCNT_CUR	DMA14 Bandwidth Limit Count Current Register	0x00000000
0x40010248	DMA4_BWMCNT	DMA14 Bandwidth Monitor Count Register	0x00000000
0x4001024C	DMA4_BWMCNT_CUR	DMA14 Bandwidth Monitor Count Current Register	0x00000000

Table A-19: ADSP-CM40x DMA15 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40010280	DMA5_DSCPTR_NXT	DMA15 Pointer to Next Initial Descriptor Register	0x00000000
0x40010284	DMA5_ADDRSTART	DMA15 Start Address of Current Buffer Register	0x00000000
0x40010288	DMA5_CFG	DMA15 Configuration Register	0x00000000
0x4001028C	DMA5_XCNT	DMA15 Inner Loop Count Start Value Register	0x00000000
0x40010290	DMA5_XMOD	DMA15 Inner Loop Address Increment Register	0x00000000
0x40010294	DMA5_YCNT	DMA15 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40010298	DMA5_YMOD	DMA15 Outer Loop Address Increment (2D only) Register	0x00000000
0x400102A4	DMA5_DSCPTR_CUR	DMA15 Current Descriptor Pointer Register	0x00000000
0x400102A8	DMA5_DSCPTR_PRV	DMA15 Previous Initial Descriptor Pointer Register	0x00000000
0x400102AC	DMA5_ADDR_CUR	DMA15 Current Address Register	0x00000000
0x400102B0	DMA5_STAT	DMA15 Status Register	0x00006000

Table A-19: ADSP-CM40x DMA15 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x400102B4	DMA5_XCNT_CUR	DMA15 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000
0x400102B8	DMA5_YCNT_CUR	DMA15 Current Row Count (2D only) Register	0x00000000
0x400102C0	DMA5_BWLCNT	DMA15 Bandwidth Limit Count Register	0x00000000
0x400102C4	DMA5_BWLCNT_CUR	DMA15 Bandwidth Limit Count Current Register	0x00000000
0x400102C8	DMA5_BWMCNT	DMA15 Bandwidth Monitor Count Register	0x00000000
0x400102CC	DMA5_BWMCNT_CUR	DMA15 Bandwidth Monitor Count Current Register	0x00000000

Table A-20: ADSP-CM40x DMA16 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40012000	DMA12_DSCPTR_NXT	DMA16 Pointer to Next Initial Descriptor Register	0x00000000
0x40012004	DMA12_ADDRSTART	DMA16 Start Address of Current Buffer Register	0x00000000
0x40012008	DMA12_CFG	DMA16 Configuration Register	0x00000000
0x4001200C	DMA12_XCNT	DMA16 Inner Loop Count Start Value Register	0x00000000
0x40012010	DMA12_XMOD	DMA16 Inner Loop Address Increment Register	0x00000000
0x40012014	DMA12_YCNT	DMA16 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40012018	DMA12_YMOD	DMA16 Outer Loop Address Increment (2D only) Register	0x00000000
0x40012024	DMA12_DSCPTR_CUR	DMA16 Current Descriptor Pointer Register	0x00000000
0x40012028	DMA12_DSCPTR_PRV	DMA16 Previous Initial Descriptor Pointer Register	0x00000000
0x4001202C	DMA12_ADDR_CUR	DMA16 Current Address Register	0x00000000
0x40012030	DMA12_STAT	DMA16 Status Register	0x00006000
0x40012034	DMA12_XCNT_CUR	DMA16 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000
0x40012038	DMA12_YCNT_CUR	DMA16 Current Row Count (2D only) Register	0x00000000
0x40012040	DMA12_BWLCNT	DMA16 Bandwidth Limit Count Register	0x00000000
0x40012044	DMA12_BWLCNT_CUR	DMA16 Bandwidth Limit Count Current Register	0x00000000
0x40012048	DMA12_BWMCNT	DMA16 Bandwidth Monitor Count Register	0x00000000
0x4001204C	DMA12_BWMCNT_CUR	DMA16 Bandwidth Monitor Count Current Register	0x00000000

Table A-21: ADSP-CM40x DMA17 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40012080	DMA13_DSCPTR_NXT	DMA17 Pointer to Next Initial Descriptor Register	0x00000000
0x40012084	DMA13_ADDRSTART	DMA17 Start Address of Current Buffer Register	0x00000000
0x40012088	DMA13_CFG	DMA17 Configuration Register	0x00000000
0x4001208C	DMA13_XCNT	DMA17 Inner Loop Count Start Value Register	0x00000000
0x40012090	DMA13_XMOD	DMA17 Inner Loop Address Increment Register	0x00000000
0x40012094	DMA13_YCNT	DMA17 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40012098	DMA13_YMOD	DMA17 Outer Loop Address Increment (2D only) Register	0x00000000
0x400120A4	DMA13_DSCPTR_CUR	DMA17 Current Descriptor Pointer Register	0x00000000
0x400120A8	DMA13_DSCPTR_PRV	DMA17 Previous Initial Descriptor Pointer Register	0x00000000
0x400120AC	DMA13_ADDR_CUR	DMA17 Current Address Register	0x00000000
0x400120B0	DMA13_STAT	DMA17 Status Register	0x00006000
0x400120B4	DMA13_XCNT_CUR	DMA17 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000
0x400120B8	DMA13_YCNT_CUR	DMA17 Current Row Count (2D only) Register	0x00000000
0x400120C0	DMA13_BWLCNT	DMA17 Bandwidth Limit Count Register	0x00000000
0x400120C4	DMA13_BWLCNT_CUR	DMA17 Bandwidth Limit Count Current Register	0x00000000
0x400120C8	DMA13_BWMCNT	DMA17 Bandwidth Monitor Count Register	0x00000000
0x400120CC	DMA13_BWMCNT_CUR	DMA17 Bandwidth Monitor Count Current Register	0x00000000

Table A-22: ADSP-CM40x DMA18 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40012100	DMA14_DSCPTR_NXT	DMA18 Pointer to Next Initial Descriptor Register	0x00000000
0x40012104	DMA14_ADDRSTART	DMA18 Start Address of Current Buffer Register	0x00000000
0x40012108	DMA14_CFG	DMA18 Configuration Register	0x00000000
0x4001210C	DMA14_XCNT	DMA18 Inner Loop Count Start Value Register	0x00000000
0x40012110	DMA14_XMOD	DMA18 Inner Loop Address Increment Register	0x00000000
0x40012114	DMA14_YCNT	DMA18 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40012118	DMA14_YMOD	DMA18 Outer Loop Address Increment (2D only) Register	0x00000000

Table A-22: ADSP-CM40x DMA18 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40012124	DMA14_DSCPTR_CUR	DMA18 Current Descriptor Pointer Register	0x00000000
0x40012128	DMA14_DSCPTR_PRV	DMA18 Previous Initial Descriptor Pointer Register	0x00000000
0x4001212C	DMA14_ADDR_CUR	DMA18 Current Address Register	0x00000000
0x40012130	DMA14_STAT	DMA18 Status Register	0x00006000
0x40012134	DMA14_XCNT_CUR	DMA18 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000
0x40012138	DMA14_YCNT_CUR	DMA18 Current Row Count (2D only) Register	0x00000000
0x40012140	DMA14_BWLCNT	DMA18 Bandwidth Limit Count Register	0x00000000
0x40012144	DMA14_BWLCNT_CUR	DMA18 Bandwidth Limit Count Current Register	0x00000000
0x40012148	DMA14_BWMCNT	DMA18 Bandwidth Monitor Count Register	0x00000000
0x4001214C	DMA14_BWMCNT_CUR	DMA18 Bandwidth Monitor Count Current Register	0x00000000

Table A-23: ADSP-CM40x DMA19 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40012180	DMA15_DSCPTR_NXT	DMA19 Pointer to Next Initial Descriptor Register	0x00000000
0x40012184	DMA15_ADDRSTART	DMA19 Start Address of Current Buffer Register	0x00000000
0x40012188	DMA15_CFG	DMA19 Configuration Register	0x00000000
0x4001218C	DMA15_XCNT	DMA19 Inner Loop Count Start Value Register	0x00000000
0x40012190	DMA15_XMOD	DMA19 Inner Loop Address Increment Register	0x00000000
0x40012194	DMA15_YCNT	DMA19 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40012198	DMA15_YMOD	DMA19 Outer Loop Address Increment (2D only) Register	0x00000000
0x400121A4	DMA15_DSCPTR_CUR	DMA19 Current Descriptor Pointer Register	0x00000000
0x400121A8	DMA15_DSCPTR_PRV	DMA19 Previous Initial Descriptor Pointer Register	0x00000000
0x400121AC	DMA15_ADDR_CUR	DMA19 Current Address Register	0x00000000
0x400121B0	DMA15_STAT	DMA19 Status Register	0x00006000
0x400121B4	DMA15_XCNT_CUR	DMA19 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000
0x400121B8	DMA15_YCNT_CUR	DMA19 Current Row Count (2D only) Register	0x00000000
0x400121C0	DMA15_BWLCNT	DMA19 Bandwidth Limit Count Register	0x00000000
0x400121C4	DMA15_BWLCNT_CUR	DMA19 Bandwidth Limit Count Current Register	0x00000000

Table A-23: ADSP-CM40x DMA19 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x400121C8	DMA15_BWMCNT	DMA19 Bandwidth Monitor Count Register	0x00000000
0x400121CC	DMA15_BWMCNT_CUR	DMA19 Bandwidth Monitor Count Current Register	0x00000000

Table A-24: ADSP-CM40x DMA2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40011100	DMA8_DSCPTR_NXT	DMA2 Pointer to Next Initial Descriptor Register	0x00000000
0x40011104	DMA8_ADDRSTART	DMA2 Start Address of Current Buffer Register	0x00000000
0x40011108	DMA8_CFG	DMA2 Configuration Register	0x00000000
0x4001110C	DMA8_XCNT	DMA2 Inner Loop Count Start Value Register	0x00000000
0x40011110	DMA8_XMOD	DMA2 Inner Loop Address Increment Register	0x00000000
0x40011114	DMA8_YCNT	DMA2 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40011118	DMA8_YMOD	DMA2 Outer Loop Address Increment (2D only) Register	0x00000000
0x40011124	DMA8_DSCPTR_CUR	DMA2 Current Descriptor Pointer Register	0x00000000
0x40011128	DMA8_DSCPTR_PRV	DMA2 Previous Initial Descriptor Pointer Register	0x00000000
0x4001112C	DMA8_ADDR_CUR	DMA2 Current Address Register	0x00000000
0x40011130	DMA8_STAT	DMA2 Status Register	0x00006000
0x40011134	DMA8_XCNT_CUR	DMA2 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000
0x40011138	DMA8_YCNT_CUR	DMA2 Current Row Count (2D only) Register	0x00000000
0x40011140	DMA8_BWLCNT	DMA2 Bandwidth Limit Count Register	0x00000000
0x40011144	DMA8_BWLCNT_CUR	DMA2 Bandwidth Limit Count Current Register	0x00000000
0x40011148	DMA8_BWMCNT	DMA2 Bandwidth Monitor Count Register	0x00000000
0x4001114C	DMA8_BWMCNT_CUR	DMA2 Bandwidth Monitor Count Current Register	0x00000000

Table A-25: ADSP-CM40x DMA20 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40012200	DMA16_DSCPTR_NXT	DMA20 Pointer to Next Initial Descriptor Register	0x00000000
0x40012204	DMA16_ADDRSTART	DMA20 Start Address of Current Buffer Register	0x00000000
0x40012208	DMA16_CFG	DMA20 Configuration Register	0x00000000
0x4001220C	DMA16_XCNT	DMA20 Inner Loop Count Start Value Register	0x00000000

Table A-25: ADSP-CM40x DMA20 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40012210	DMA16_XMOD	DMA20 Inner Loop Address Increment Register	0x00000000
0x40012214	DMA16_YCNT	DMA20 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40012218	DMA16_YMOD	DMA20 Outer Loop Address Increment (2D only) Register	0x00000000
0x40012224	DMA16_DSCPTR_CUR	DMA20 Current Descriptor Pointer Register	0x00000000
0x40012228	DMA16_DSCPTR_PRV	DMA20 Previous Initial Descriptor Pointer Register	0x00000000
0x4001222C	DMA16_ADDR_CUR	DMA20 Current Address Register	0x00000000
0x40012230	DMA16_STAT	DMA20 Status Register	0x00006000
0x40012234	DMA16_XCNT_CUR	DMA20 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000
0x40012238	DMA16_YCNT_CUR	DMA20 Current Row Count (2D only) Register	0x00000000
0x40012240	DMA16_BWLCNT	DMA20 Bandwidth Limit Count Register	0x00000000
0x40012244	DMA16_BWLCNT_CUR	DMA20 Bandwidth Limit Count Current Register	0x00000000
0x40012248	DMA16_BWMCNT	DMA20 Bandwidth Monitor Count Register	0x00000000
0x4001224C	DMA16_BWMCNT_CUR	DMA20 Bandwidth Monitor Count Current Register	0x00000000

Table A-26: ADSP-CM40x DMA3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40011180	DMA9_DSCPTR_NXT	DMA3 Pointer to Next Initial Descriptor Register	0x00000000
0x40011184	DMA9_ADDRSTART	DMA3 Start Address of Current Buffer Register	0x00000000
0x40011188	DMA9_CFG	DMA3 Configuration Register	0x00000000
0x4001118C	DMA9_XCNT	DMA3 Inner Loop Count Start Value Register	0x00000000
0x40011190	DMA9_XMOD	DMA3 Inner Loop Address Increment Register	0x00000000
0x40011194	DMA9_YCNT	DMA3 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40011198	DMA9_YMOD	DMA3 Outer Loop Address Increment (2D only) Register	0x00000000
0x400111A4	DMA9_DSCPTR_CUR	DMA3 Current Descriptor Pointer Register	0x00000000
0x400111A8	DMA9_DSCPTR_PRV	DMA3 Previous Initial Descriptor Pointer Register	0x00000000
0x400111AC	DMA9_ADDR_CUR	DMA3 Current Address Register	0x00000000
0x400111B0	DMA9_STAT	DMA3 Status Register	0x00006000
0x400111B4	DMA9_XCNT_CUR	DMA3 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000

Table A-26: ADSP-CM40x DMA3 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x400111B8	DMA9_YCNT_CUR	DMA3 Current Row Count (2D only) Register	0x00000000
0x400111C0	DMA9_BWLCNT	DMA3 Bandwidth Limit Count Register	0x00000000
0x400111C4	DMA9_BWLCNT_CUR	DMA3 Bandwidth Limit Count Current Register	0x00000000
0x400111C8	DMA9_BWMCNT	DMA3 Bandwidth Monitor Count Register	0x00000000
0x400111CC	DMA9_BWMCNT_CUR	DMA3 Bandwidth Monitor Count Current Register	0x00000000

Table A-27: ADSP-CM40x DMA4 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40011200	DMA10_DSCPTR_NXT	DMA4 Pointer to Next Initial Descriptor Register	0x00000000
0x40011204	DMA10_ADDRSTART	DMA4 Start Address of Current Buffer Register	0x00000000
0x40011208	DMA10_CFG	DMA4 Configuration Register	0x00000000
0x4001120C	DMA10_XCNT	DMA4 Inner Loop Count Start Value Register	0x00000000
0x40011210	DMA10_XMOD	DMA4 Inner Loop Address Increment Register	0x00000000
0x40011214	DMA10_YCNT	DMA4 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40011218	DMA10_YMOD	DMA4 Outer Loop Address Increment (2D only) Register	0x00000000
0x40011224	DMA10_DSCPTR_CUR	DMA4 Current Descriptor Pointer Register	0x00000000
0x40011228	DMA10_DSCPTR_PRV	DMA4 Previous Initial Descriptor Pointer Register	0x00000000
0x4001122C	DMA10_ADDR_CUR	DMA4 Current Address Register	0x00000000
0x40011230	DMA10_STAT	DMA4 Status Register	0x00006000
0x40011234	DMA10_XCNT_CUR	DMA4 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000
0x40011238	DMA10_YCNT_CUR	DMA4 Current Row Count (2D only) Register	0x00000000
0x40011240	DMA10_BWLCNT	DMA4 Bandwidth Limit Count Register	0x00000000
0x40011244	DMA10_BWLCNT_CUR	DMA4 Bandwidth Limit Count Current Register	0x00000000
0x40011248	DMA10_BWMCNT	DMA4 Bandwidth Monitor Count Register	0x00000000
0x4001124C	DMA10_BWMCNT_CUR	DMA4 Bandwidth Monitor Count Current Register	0x00000000

Table A-28: ADSP-CM40x DMA5 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40011280	DMA11_DSCPTR_NXT	DMA5 Pointer to Next Initial Descriptor Register	0x00000000

Table A-28: ADSP-CM40x DMA5 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40011284	DMA11_ADDRSTART	DMA5 Start Address of Current Buffer Register	0x00000000
0x40011288	DMA11_CFG	DMA5 Configuration Register	0x00000000
0x4001128C	DMA11_XCNT	DMA5 Inner Loop Count Start Value Register	0x00000000
0x40011290	DMA11_XMOD	DMA5 Inner Loop Address Increment Register	0x00000000
0x40011294	DMA11_YCNT	DMA5 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40011298	DMA11_YMOD	DMA5 Outer Loop Address Increment (2D only) Register	0x00000000
0x400112A4	DMA11_DSCPTR_CUR	DMA5 Current Descriptor Pointer Register	0x00000000
0x400112A8	DMA11_DSCPTR_PRV	DMA5 Previous Initial Descriptor Pointer Register	0x00000000
0x400112AC	DMA11_ADDR_CUR	DMA5 Current Address Register	0x00000000
0x400112B0	DMA11_STAT	DMA5 Status Register	0x00006000
0x400112B4	DMA11_XCNT_CUR	DMA5 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000
0x400112B8	DMA11_YCNT_CUR	DMA5 Current Row Count (2D only) Register	0x00000000
0x400112C0	DMA11_BWLCNT	DMA5 Bandwidth Limit Count Register	0x00000000
0x400112C4	DMA11_BWLCNT_CUR	DMA5 Bandwidth Limit Count Current Register	0x00000000
0x400112C8	DMA11_BWMCNT	DMA5 Bandwidth Monitor Count Register	0x00000000
0x400112CC	DMA11_BWMCNT_CUR	DMA5 Bandwidth Monitor Count Current Register	0x00000000

Table A-29: ADSP-CM40x DMA6 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40013000	DMA17_DSCPTR_NXT	DMA6 Pointer to Next Initial Descriptor Register	0x00000000
0x40013004	DMA17_ADDRSTART	DMA6 Start Address of Current Buffer Register	0x00000000
0x40013008	DMA17_CFG	DMA6 Configuration Register	0x00000000
0x4001300C	DMA17_XCNT	DMA6 Inner Loop Count Start Value Register	0x00000000
0x40013010	DMA17_XMOD	DMA6 Inner Loop Address Increment Register	0x00000000
0x40013014	DMA17_YCNT	DMA6 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40013018	DMA17_YMOD	DMA6 Outer Loop Address Increment (2D only) Register	0x00000000
0x40013024	DMA17_DSCPTR_CUR	DMA6 Current Descriptor Pointer Register	0x00000000
0x40013028	DMA17_DSCPTR_PRV	DMA6 Previous Initial Descriptor Pointer Register	0x00000000
0x4001302C	DMA17_ADDR_CUR	DMA6 Current Address Register	0x00000000
0x40013030	DMA17_STAT	DMA6 Status Register	0x00006000

Table A-29: ADSP-CM40x DMA6 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40013034	DMA17_XCNT_CUR	DMA6 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000
0x40013038	DMA17_YCNT_CUR	DMA6 Current Row Count (2D only) Register	0x00000000
0x40013040	DMA17_BWLCNT	DMA6 Bandwidth Limit Count Register	0x00000000
0x40013044	DMA17_BWLCNT_CUR	DMA6 Bandwidth Limit Count Current Register	0x00000000
0x40013048	DMA17_BWMCNT	DMA6 Bandwidth Monitor Count Register	0x00000000
0x4001304C	DMA17_BWMCNT_CUR	DMA6 Bandwidth Monitor Count Current Register	0x00000000

Table A-30: ADSP-CM40x DMA7 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40013080	DMA18_DSCPTR_NXT	DMA7 Pointer to Next Initial Descriptor Register	0x00000000
0x40013084	DMA18_ADDRSTART	DMA7 Start Address of Current Buffer Register	0x00000000
0x40013088	DMA18_CFG	DMA7 Configuration Register	0x00000000
0x4001308C	DMA18_XCNT	DMA7 Inner Loop Count Start Value Register	0x00000000
0x40013090	DMA18_XMOD	DMA7 Inner Loop Address Increment Register	0x00000000
0x40013094	DMA18_YCNT	DMA7 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40013098	DMA18_YMOD	DMA7 Outer Loop Address Increment (2D only) Register	0x00000000
0x400130A4	DMA18_DSCPTR_CUR	DMA7 Current Descriptor Pointer Register	0x00000000
0x400130A8	DMA18_DSCPTR_PRV	DMA7 Previous Initial Descriptor Pointer Register	0x00000000
0x400130AC	DMA18_ADDR_CUR	DMA7 Current Address Register	0x00000000
0x400130B0	DMA18_STAT	DMA7 Status Register	0x00006000
0x400130B4	DMA18_XCNT_CUR	DMA7 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000
0x400130B8	DMA18_YCNT_CUR	DMA7 Current Row Count (2D only) Register	0x00000000
0x400130C0	DMA18_BWLCNT	DMA7 Bandwidth Limit Count Register	0x00000000
0x400130C4	DMA18_BWLCNT_CUR	DMA7 Bandwidth Limit Count Current Register	0x00000000
0x400130C8	DMA18_BWMCNT	DMA7 Bandwidth Monitor Count Register	0x00000000
0x400130CC	DMA18_BWMCNT_CUR	DMA7 Bandwidth Monitor Count Current Register	0x00000000

Table A-31: ADSP-CM40x DMA8 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40013100	DMA19_DSCPTR_NXT	DMA8 Pointer to Next Initial Descriptor Register	0x00000000
0x40013104	DMA19_ADDRSTART	DMA8 Start Address of Current Buffer Register	0x00000000
0x40013108	DMA19_CFG	DMA8 Configuration Register	0x00000000
0x4001310C	DMA19_XCNT	DMA8 Inner Loop Count Start Value Register	0x00000000
0x40013110	DMA19_XMOD	DMA8 Inner Loop Address Increment Register	0x00000000
0x40013114	DMA19_YCNT	DMA8 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40013118	DMA19_YMOD	DMA8 Outer Loop Address Increment (2D only) Register	0x00000000
0x40013124	DMA19_DSCPTR_CUR	DMA8 Current Descriptor Pointer Register	0x00000000
0x40013128	DMA19_DSCPTR_PRV	DMA8 Previous Initial Descriptor Pointer Register	0x00000000
0x4001312C	DMA19_ADDR_CUR	DMA8 Current Address Register	0x00000000
0x40013130	DMA19_STAT	DMA8 Status Register	0x00006000
0x40013134	DMA19_XCNT_CUR	DMA8 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000
0x40013138	DMA19_YCNT_CUR	DMA8 Current Row Count (2D only) Register	0x00000000
0x40013140	DMA19_BWLCNT	DMA8 Bandwidth Limit Count Register	0x00000000
0x40013144	DMA19_BWLCNT_CUR	DMA8 Bandwidth Limit Count Current Register	0x00000000
0x40013148	DMA19_BWMCNT	DMA8 Bandwidth Monitor Count Register	0x00000000
0x4001314C	DMA19_BWMCNT_CUR	DMA8 Bandwidth Monitor Count Current Register	0x00000000

Table A-32: ADSP-CM40x DMA9 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40013180	DMA20_DSCPTR_NXT	DMA9 Pointer to Next Initial Descriptor Register	0x00000000
0x40013184	DMA20_ADDRSTART	DMA9 Start Address of Current Buffer Register	0x00000000
0x40013188	DMA20_CFG	DMA9 Configuration Register	0x00000000
0x4001318C	DMA20_XCNT	DMA9 Inner Loop Count Start Value Register	0x00000000
0x40013190	DMA20_XMOD	DMA9 Inner Loop Address Increment Register	0x00000000
0x40013194	DMA20_YCNT	DMA9 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40013198	DMA20_YMOD	DMA9 Outer Loop Address Increment (2D only) Register	0x00000000
0x400131A4	DMA20_DSCPTR_CUR	DMA9 Current Descriptor Pointer Register	0x00000000
0x400131A8	DMA20_DSCPTR_PRV	DMA9 Previous Initial Descriptor Pointer Register	0x00000000
0x400131AC	DMA20_ADDR_CUR	DMA9 Current Address Register	0x00000000

Table A-32: ADSP-CM40x DMA9 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x400131B0	DMA20_STAT	DMA9 Status Register	0x00006000
0x400131B4	DMA20_XCNT_CUR	DMA9 Current Count(1D) or intra-row XCNT (2D) Register	0x00000000
0x400131B8	DMA20_YCNT_CUR	DMA9 Current Row Count (2D only) Register	0x00000000
0x400131C0	DMA20_BWLCNT	DMA9 Bandwidth Limit Count Register	0x00000000
0x400131C4	DMA20_BWLCNT_CUR	DMA9 Bandwidth Limit Count Current Register	0x00000000
0x400131C8	DMA20_BWMCNT	DMA9 Bandwidth Monitor Count Register	0x00000000
0x400131CC	DMA20_BWMCNT_CUR	DMA9 Bandwidth Monitor Count Current Register	0x00000000

Table A-33: ADSP-CM40x DPM0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40017000	DPM0_CTL	DPM0 Control Register	0x00000000
0x40017004	DPM0_STAT	DPM0 Status Register	0x00000002
0x40017008	DPM0_CCBF_DIS	DPM0 Core Clock Buffer Disable Register	0x00000000
0x4001700C	DPM0_CCBF_EN	DPM0 Core Clock Buffer Enable Register	0x00000000
0x40017010	DPM0_CCBF_STAT	DPM0 Core Clock Buffer Status Register	0x00000000
0x40017014	DPM0_CCBF_STAT_STKY	DPM0 Core Clock Buffer Status Sticky Register	0x00000000
0x40017018	DPM0_SCBF_DIS	DPM0 System Clock Buffer Disable Register	0x00000000
0x4001701C	DPM0_WAKE_EN	DPM0 Wakeup Enable Register	0x00000000
0x40017020	DPM0_WAKE_POL	DPM0 Wakeup Polarity Register	0x00000000
0x40017024	DPM0_WAKE_STAT	DPM0 Wakeup Status Register	0x00000000

Table A-34: ADSP-CM40x EMAC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40020000	EMAC0_MACCFG	EMAC0 MAC Configuration Register	0x00008000
0x40020004	EMAC0_MACFRMFILT	EMAC0 MAC Rx Frame Filter Register	0x00000000
0x40020008	EMAC0_HASHTBL_HI	EMAC0 Hash Table High Register	0x00000000
0x4002000C	EMAC0_HASHTBL_LO	EMAC0 Hash Table Low Register	0x00000000
0x40020010	EMAC0_SMI_ADDR	EMAC0 SMI Address Register	0x00000000
0x40020014	EMAC0_SMI_DATA	EMAC0 SMI Data Register	0x00000000

Table A-34: ADSP-CM40x EMAC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40020018	EMAC0_FLOWCTL	EMAC0 FLow Control Register	0x00000000
0x4002001C	EMAC0_VLANTAG	EMAC0 VLAN Tag Register	0x00000000
0x40020024	EMAC0_DBG	EMAC0 Debug Register	0x00000000
0x40020038	EMAC0_ISTAT	EMAC0 Interrupt Status Register	0x00000000
0x4002003C	EMAC0_IMSK	EMAC0 Interrupt Mask Register	0x00000000
0x40020040	EMAC0_ADDR0_HI	EMAC0 MAC Address 0 High Register	0x8000FFFF
0x40020044	EMAC0_ADDR0_LO	EMAC0 MAC Address 0 Low Register	0xFFFFFFFF
0x40020100	EMAC0_MMC_CTL	EMAC0 MMC Control Register	0x00000000
0x40020104	EMAC0_MMC_RXINT	EMAC0 MMC Rx Interrupt Register	0x00000000
0x40020108	EMAC0_MMC_TXINT	EMAC0 MMC Tx Interrupt Register	0x00000000
0x4002010C	EMAC0_MMC_RXIMSK	EMAC0 MMC Rx Interrupt Mask Register	0x00000000
0x40020110	EMAC0_MMC_TXIMSK	EMAC0 MMC TX Interrupt Mask Register	0x00000000
0x40020114	EMAC0_TXOCTCNT_GB	EMAC0 Tx OCT Count (Good/Bad) Register	0x00000000
0x40020118	EMAC0_TXFRMCNT_GB	EMAC0 Tx Frame Count (Good/Bad) Register	0x00000000
0x4002011C	EMAC0_TXBCASTFRM_G	EMAC0 Tx Broadcast Frames (Good) Register	0x00000000
0x40020120	EMAC0_TXMCASTFRM_G	EMAC0 Tx Multicast Frames (Good) Register	0x00000000
0x40020124	EMAC0_TX64_GB	EMAC0 Tx 64-Byte Frames (Good/Bad) Register	0x00000000
0x40020128	EMAC0_TX65TO127_GB	EMAC0 Tx 65- to 127-Byte Frames (Good/Bad) Register	0x00000000
0x4002012C	EMAC0_TX128TO255_GB	EMAC0 Tx 128- to 255-Byte Frames (Good/Bad) Register	0x00000000
0x40020130	EMAC0_TX256TO511_GB	EMAC0 Tx 256- to 511-Byte Frames (Good/Bad) Register	0x00000000
0x40020134	EMAC0_TX512TO1023_GB	EMAC0 Tx 512- to 1023-Byte Frames (Good/Bad) Register	0x00000000
0x40020138	EMAC0_TX1024TOMAX_GB	EMAC0 Tx 1024- to Max-Byte Frames (Good/Bad) Register	0x00000000
0x4002013C	EMAC0_TXUCASTFRM_GB	EMAC0 Tx Unicast Frames (Good/Bad) Register	0x00000000
0x40020140	EMAC0_TXMCASTFRM_GB	EMAC0 Tx Multicast Frames (Good/Bad) Register	0x00000000
0x40020144	EMAC0_TXBCASTFRM_GB	EMAC0 Tx Broadcast Frames (Good/Bad) Register	0x00000000
0x40020148	EMAC0_TXUNDR_ERR	EMAC0 Tx Underflow Error Register	0x00000000

Table A-34: ADSP-CM40x EMAC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4002014C	EMAC0_TXSNGCOL_G	EMAC0 Tx Single Collision (Good) Register	0x00000000
0x40020150	EMAC0_TXMULTCOL_G	EMAC0 Tx Multiple Collision (Good) Register	0x00000000
0x40020154	EMAC0_TXDEFERRED	EMAC0 Tx Deferred Register	0x00000000
0x40020158	EMAC0_TXLATECOL	EMAC0 Tx Late Collision Register	0x00000000
0x4002015C	EMAC0_TXEXCESSCOL	EMAC0 Tx Excess Collision Register	0x00000000
0x40020160	EMAC0_TXCARR_ERR	EMAC0 Tx Carrier Error Register	0x00000000
0x40020164	EMAC0_TXOCTCNT_G	EMAC0 Tx Octet Count (Good) Register	0x00000000
0x40020168	EMAC0_TXFRMCNT_G	EMAC0 Tx Frame Count (Good) Register	0x00000000
0x4002016C	EMAC0_TXEXCESSDEF	EMAC0 Tx Excess Deferral Register	0x00000000
0x40020170	EMAC0_TXPAUSEFRM	EMAC0 Tx Pause Frame Register	0x00000000
0x40020174	EMAC0_TXVLANFRM_G	EMAC0 Tx VLAN Frames (Good) Register	0x00000000
0x40020180	EMAC0_RXFRMCNT_GB	EMAC0 Rx Frame Count (Good/Bad) Register	0x00000000
0x40020184	EMAC0_RXOCTCNT_GB	EMAC0 Rx Octet Count (Good/Bad) Register	0x00000000
0x40020188	EMAC0_RXOCTCNT_G	EMAC0 Rx Octet Count (Good) Register	0x00000000
0x4002018C	EMAC0_RXBCASTFRM_G	EMAC0 Rx Broadcast Frames (Good) Register	0x00000000
0x40020190	EMAC0_RXMCASTFRM_G	EMAC0 Rx Multicast Frames (Good) Register	0x00000000
0x40020194	EMAC0_RXCRC_ERR	EMAC0 Rx CRC Error Register	0x00000000
0x40020198	EMAC0_RXALIGN_ERR	EMAC0 Rx alignment Error Register	0x00000000
0x4002019C	EMAC0_RXRUNT_ERR	EMAC0 Rx Runt Error Register	0x00000000
0x400201A0	EMAC0_RXJAB_ERR	EMAC0 Rx Jab Error Register	0x00000000
0x400201A4	EMAC0_RXUSIZE_G	EMAC0 Rx Undersize (Good) Register	0x00000000
0x400201A8	EMAC0_RXOSIZE_G	EMAC0 Rx Oversize (Good) Register	0x00000000
0x400201AC	EMAC0_RX64_GB	EMAC0 Rx 64-Byte Frames (Good/Bad) Register	0x00000000
0x400201B0	EMAC0_RX65TO127_GB	EMAC0 Rx 65- to 127-Byte Frames (Good/Bad) Register	0x00000000
0x400201B4	EMAC0_RX128TO255_GB	EMAC0 Rx 128- to 255-Byte Frames (Good/Bad) Register	0x00000000
0x400201B8	EMAC0_RX256TO511_GB	EMAC0 Rx 256- to 511-Byte Frames (Good/Bad) Register	0x00000000
0x400201BC	EMAC0_RX512TO1023_GB	EMAC0 Rx 512- to 1023-Byte Frames (Good/Bad) Register	0x00000000
0x400201C0	EMAC0_RX1024TOMAX_GB	EMAC0 Rx 1024- to Max-Byte Frames (Good/Bad) Register	0x00000000

Table A-34: ADSP-CM40x EMAC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x400201C4	EMAC0_RXUCASTFRM_G	EMAC0 Rx Unicast Frames (Good) Register	0x00000000
0x400201C8	EMAC0_RXLEN_ERR	EMAC0 Rx Length Error Register	0x00000000
0x400201CC	EMAC0_RXOORTYPE	EMAC0 Rx Out Of Range Type Register	0x00000000
0x400201D0	EMAC0_RXPAUSEFRM	EMAC0 Rx Pause Frames Register	0x00000000
0x400201D4	EMAC0_RXFIFO_OVF	EMAC0 Rx FIFO Overflow Register	0x00000000
0x400201D8	EMAC0_RXVLANFRM_GB	EMAC0 Rx VLAN Frames (Good/Bad) Register	0x00000000
0x400201DC	EMAC0_RXWDOG_ERR	EMAC0 Rx Watch Dog Error Register	0x00000000
0x40020200	EMAC0_IPC_RXIMSK	EMAC0 MMC IPC Rx Interrupt Mask Register	0x00000000
0x40020208	EMAC0_IPC_RXINT	EMAC0 MMC IPC Rx Interrupt Register	0x00000000
0x40020210	EMAC0_RXIPV4_GD_FRM	EMAC0 Rx IPv4 Datagrams (Good) Register	0x00000000
0x40020214	EMAC0_RXIPV4_HDR_ERR_FRM	EMAC0 Rx IPv4 Datagrams Header Errors Register	0x00000000
0x40020218	EMAC0_RXIPV4_NOPAY_FRM	EMAC0 Rx IPv4 Datagrams No Payload Frame Register	0x00000000
0x4002021C	EMAC0_RXIPV4_FRAG_FRM	EMAC0 Rx IPv4 Datagrams Fragmented Frames Register	0x00000000
0x40020220	EMAC0_RXIPV4_UDSBL_FRM	EMAC0 Rx IPv4 UDP Disabled Frames Register	0x00000000
0x40020224	EMAC0_RXIPV6_GD_FRM	EMAC0 Rx IPv6 Datagrams Good Frames Register	0x00000000
0x40020228	EMAC0_RXIPV6_HDR_ERR_FRM	EMAC0 Rx IPv6 Datagrams Header Error Frames Register	0x00000000
0x4002022C	EMAC0_RXIPV6_NOPAY_FRM	EMAC0 Rx IPv6 Datagrams No Payload Frames Register	0x00000000
0x40020230	EMAC0_RXUDP_GD_FRM	EMAC0 Rx UDP Good Frames Register	0x00000000
0x40020234	EMAC0_RXUDP_ERR_FRM	EMAC0 Rx UDP Error Frames Register	0x00000000
0x40020238	EMAC0_RXTCP_GD_FRM	EMAC0 Rx TCP Good Frames Register	0x00000000
0x4002023C	EMAC0_RXTCP_ERR_FRM	EMAC0 Rx TCP Error Frames Register	0x00000000
0x40020240	EMAC0_RXICMP_GD_FRM	EMAC0 Rx ICMP Good Frames Register	0x00000000
0x40020244	EMAC0_RXICMP_ERR_FRM	EMAC0 Rx ICMP Error Frames Register	0x00000000
0x40020250	EMAC0_RXIPV4_GD_OCT	EMAC0 Rx IPv4 Datagrams Good Octets Register	0x00000000

Table A-34: ADSP-CM40x EMAC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40020254	EMAC0_RXIPV4_HDR_ERR_OCT	EMAC0 Rx IPv4 Datagrams Header Errors Register	0x00000000
0x40020258	EMAC0_RXIPV4_NOPAY_OCT	EMAC0 Rx IPv4 Datagrams No Payload Octets Register	0x00000000
0x4002025C	EMAC0_RXIPV4_FRAG_OCT	EMAC0 Rx IPv4 Datagrams Fragmented Octets Register	0x00000000
0x40020260	EMAC0_RXIPV4_UDSBL_OCT	EMAC0 Rx IPv4 UDP Disabled Octets Register	0x00000000
0x40020264	EMAC0_RXIPV6_GD_OCT	EMAC0 Rx IPv6 Good Octets Register	0x00000000
0x40020268	EMAC0_RXIPV6_HDR_ERR_OCT	EMAC0 Rx IPv6 Header Errors Register	0x00000000
0x4002026C	EMAC0_RXIPV6_NOPAY_OCT	EMAC0 Rx IPv6 No Payload Octets Register	0x00000000
0x40020270	EMAC0_RXUDP_GD_OCT	EMAC0 Rx UDP Good Octets Register	0x00000000
0x40020274	EMAC0_RXUDP_ERR_OCT	EMAC0 Rx UDP Error Octets Register	0x00000000
0x40020278	EMAC0_RXTCP_GD_OCT	EMAC0 Rx TCP Good Octets Register	0x00000000
0x4002027C	EMAC0_RXTCP_ERR_OCT	EMAC0 Rx TCP Error Octets Register	0x00000000
0x40020280	EMAC0_RXICMP_GD_OCT	EMAC0 Rx ICMP Good Octets Register	0x00000000
0x40020284	EMAC0_RXICMP_ERR_OCT	EMAC0 Rx ICMP Error Octets Register	0x00000000
0x40020700	EMAC0_TM_CTL	EMAC0 Time Stamp Control Register	0x00002000
0x40020704	EMAC0_TM_SUBSEC	EMAC0 Time Stamp Sub Second Increment Register	0x00000000
0x40020708	EMAC0_TM_SEC	EMAC0 Time Stamp Low Seconds Register	0x00000000
0x4002070C	EMAC0_TM_NSEC	EMAC0 Time Stamp Nanoseconds Register	0x00000000
0x40020710	EMAC0_TM_SECUPDT	EMAC0 Time Stamp Seconds Update Register	0x00000000
0x40020714	EMAC0_TM_NSECUPDT	EMAC0 Time Stamp Nanoseconds Update Register	0x00000000
0x40020718	EMAC0_TM_ADDEND	EMAC0 Time Stamp Addend Register	0x00000000
0x4002071C	EMAC0_TM_TGTM	EMAC0 Time Stamp Target Time Seconds Register	0x00000000
0x40020720	EMAC0_TM_NTGTM	EMAC0 Time Stamp Target Time Nanoseconds Register	0x00000000
0x40020724	EMAC0_TM_HISEC	EMAC0 Time Stamp High Second Register	0x00000000
0x40020728	EMAC0_TM_STMPSTAT	EMAC0 Time Stamp Status Register	0x00000000

Table A-34: ADSP-CM40x EMAC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4002072C	EMAC0_TM_PPSCTL	EMAC0 PPS Control Register	0x00000000
0x40020730	EMAC0_TM_AUXSTMP_NSEC	EMAC0 Time Stamp Auxiliary TS Nano Seconds Register	0x00000000
0x40020734	EMAC0_TM_AUXSTMP_SEC	EMAC0 Time Stamp Auxiliary TM Seconds Register	0x00000000
0x40020760	EMAC0_TM_PPSINTVL	EMAC0 Time Stamp PPS Interval Register	0x00000000
0x40020764	EMAC0_TM_PPSWIDTH	EMAC0 PPS Width Register	0x00000000
0x40021000	EMAC0_DMA_BUSMODE	EMAC0 DMA Bus Mode Register	0x00020101
0x40021004	EMAC0_DMA_TXPOLL	EMAC0 DMA Tx Poll Demand Register	0x00000000
0x40021008	EMAC0_DMA_RXPOLL	EMAC0 DMA Rx Poll Demand register	0x00000000
0x4002100C	EMAC0_DMA_RXDSC_ADDR	EMAC0 DMA Rx Descriptor List Address Register	0x00000000
0x40021010	EMAC0_DMA_TXDSC_ADDR	EMAC0 DMA Tx Descriptor List Address Register	0x00000000
0x40021014	EMAC0_DMA_STAT	EMAC0 DMA Status Register	0x00000000
0x40021018	EMAC0_DMA_OPMODE	EMAC0 DMA Operation Mode Register	0x00000000
0x4002101C	EMAC0_DMA_IEN	EMAC0 DMA Interrupt Enable Register	0x00000000
0x40021020	EMAC0_DMA_MISS_FRM	EMAC0 DMA Missed Frame Register	0x00000000
0x40021024	EMAC0_DMA_RXIWDOG	EMAC0 DMA Rx Interrupt Watch Dog Register	0x00000000
0x40021028	EMAC0_DMA_BMMODE	EMAC0 DMA SCB Bus Mode Register	0x00110001
0x4002102C	EMAC0_DMA_BMSTAT	EMAC0 DMA SCB Status Register	0x00000000
0x40021048	EMAC0_DMA_TXDSC_CUR	EMAC0 DMA Tx Descriptor Current Register	0x00000000
0x4002104C	EMAC0_DMA_RXDSC_CUR	EMAC0 DMA Rx Descriptor Current Register	0x00000000
0x40021050	EMAC0_DMA_TXBUF_CUR	EMAC0 DMA Tx Buffer Current Register	0x00000000
0x40021054	EMAC0_DMA_RXBUF_CUR	EMAC0 DMA Rx Buffer Current Register	0x00000000

Table A-35: ADSP-CM40x EMUID0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x80004020	EMUID0_ADIID	EMUID0 Analog Devices Identification	0x00004144

Table A-35: ADSP-CM40x EMUID0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x80004024	EMUID0_CHIPID	EMUID0 Chip Identification	0x00000302

Table A-36: ADSP-CM40x HAE0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x400180A0	HAE0_RUN	HAE0 Run Register	0x00000000
0x40018B00	HAE0_CFG0	HAE0 Configuration 0 Register	0x00000000
0x40018B04	HAE0_CFG1	HAE0 Configuration 1 Register	0x00000000
0x40018B08	HAE0_CFG2	HAE0 Configuration 2 Register	0x00000000
0x40018B0C	HAE0_CFG3	HAE0 Configuration 3 Register	0x00000000
0x40018B20	HAE0_STAT	HAE0 Status Register	0x00000000
0x40018B40	HAE0_ISAMPLE	HAE0 I (Current) Sample Register	0x00000000
0x40018B44	HAE0_VSAMPLE	HAE0 V (Voltage) Sample Register	0x00000000
0x40018B80	HAE0_IWAVEFORM	HAE0 I (Current) Waveform Register	0x00000000
0x40018B84	HAE0_VWAVEFORM	HAE0 V (Voltage) Waveform Register	0x00000000
0x40018C00	HAE0_RESULTS_RAM	HAE0 Results RAM Register	0x00000000
0x40019E00	HAE0_CFG4	HAE0 Configuration 4 Register	0x00000000
0x40019E04	HAE0_DIDT_GAIN	HAE0 DIDT Gain Register	0x00000000
0x40019E08	HAE0_DIDT_COEF	HAE0 DIDT Coefficient Register	0x00000000
0x40019E10	HAE0_VLEVEL	HAE0 Voltage Level Register	0x00000000
0x40019E2C	HAE0_H00_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40019E30	HAE0_H01_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40019E34	HAE0_H02_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40019E38	HAE0_H03_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40019E3C	HAE0_H04_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40019E40	HAE0_H05_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40019E44	HAE0_H06_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40019E48	HAE0_H07_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40019E4C	HAE0_H08_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40019E50	HAE0_H09_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40019E54	HAE0_H10_INDX	HAE0 Harmonic n Index Register	0x00000000

Table A-36: ADSP-CM40x HAE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40019E58	HAE0_H11_INDX	HAE0 Harmonic n Index Register	0x00000000

Table A-37: ADSP-CM40x M4_SCS0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x1C001C004	M4_SCS0_ICTR	M4_SCS0 Interrupt Control Type Register	0x00000000
0x1C001C008	M4_SCS0_ACTLR	M4_SCS0 Auxiliary Control Register	0x00000000
0x1C001C010	M4_SCS0_STCSR	M4_SCS0 SysTick Control and Status Register	0x00000000
0x1C001C014	M4_SCS0_STRVR	M4_SCS0 SysTick Reload Value Register	0x00000000
0x1C001C018	M4_SCS0_STCVR	M4_SCS0 SysTick Current Value Register	0x00000000
0x1C001C01C	M4_SCS0_STCR	M4_SCS0 SysTick Calibration Value Register	0x00000000
0x1C001C100	M4_SCS0_NVIC_ISER0	M4_SCS0 Irq Set Enable Register n	0x00000000
0x1C001C104	M4_SCS0_NVIC_ISER1	M4_SCS0 Irq Set Enable Register n	0x00000000
0x1C001C108	M4_SCS0_NVIC_ISER2	M4_SCS0 Irq Set Enable Register n	0x00000000
0x1C001C10C	M4_SCS0_NVIC_ISER3	M4_SCS0 Irq Set Enable Register n	0x00000000
0x1C001C110	M4_SCS0_NVIC_ISER4	M4_SCS0 Irq Set Enable Register n	0x00000000
0x1C001C180	M4_SCS0_NVIC_ICER0	M4_SCS0 Irq Clear Enable Register n	0x00000000
0x1C001C184	M4_SCS0_NVIC_ICER1	M4_SCS0 Irq Clear Enable Register n	0x00000000
0x1C001C188	M4_SCS0_NVIC_ICER2	M4_SCS0 Irq Clear Enable Register n	0x00000000
0x1C001C18C	M4_SCS0_NVIC_ICER3	M4_SCS0 Irq Clear Enable Register n	0x00000000
0x1C001C190	M4_SCS0_NVIC_ICER4	M4_SCS0 Irq Clear Enable Register n	0x00000000
0x1C001C200	M4_SCS0_NVIC_ISPR0	M4_SCS0 Irq Set Pending Register n	0x00000000
0x1C001C204	M4_SCS0_NVIC_ISPR1	M4_SCS0 Irq Set Pending Register n	0x00000000
0x1C001C208	M4_SCS0_NVIC_ISPR2	M4_SCS0 Irq Set Pending Register n	0x00000000
0x1C001C20C	M4_SCS0_NVIC_ISPR3	M4_SCS0 Irq Set Pending Register n	0x00000000
0x1C001C210	M4_SCS0_NVIC_ISPR4	M4_SCS0 Irq Set Pending Register n	0x00000000
0x1C001C280	M4_SCS0_NVIC_ICPR0	M4_SCS0 Irq Clear Pending Register n	0x00000000
0x1C001C284	M4_SCS0_NVIC_ICPR1	M4_SCS0 Irq Clear Pending Register n	0x00000000
0x1C001C288	M4_SCS0_NVIC_ICPR2	M4_SCS0 Irq Clear Pending Register n	0x00000000
0x1C001C28C	M4_SCS0_NVIC_ICPR3	M4_SCS0 Irq Clear Pending Register n	0x00000000
0x1C001C290	M4_SCS0_NVIC_ICPR4	M4_SCS0 Irq Clear Pending Register n	0x00000000

Table A-37: ADSP-CM40x M4_SCS0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x1C001C300	M4_SCS0_NVIC_IABR0	M4_SCS0 Irq Active Bit Register n	0x00000000
0x1C001C304	M4_SCS0_NVIC_IABR1	M4_SCS0 Irq Active Bit Register n	0x00000000
0x1C001C308	M4_SCS0_NVIC_IABR2	M4_SCS0 Irq Active Bit Register n	0x00000000
0x1C001C30C	M4_SCS0_NVIC_IABR3	M4_SCS0 Irq Active Bit Register n	0x00000000
0x1C001C310	M4_SCS0_NVIC_IABR4	M4_SCS0 Irq Active Bit Register n	0x00000000
0x1C001C400	M4_SCS0_NVIC_IPR00	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C404	M4_SCS0_NVIC_IPR01	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C408	M4_SCS0_NVIC_IPR02	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C40C	M4_SCS0_NVIC_IPR03	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C410	M4_SCS0_NVIC_IPR04	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C414	M4_SCS0_NVIC_IPR05	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C418	M4_SCS0_NVIC_IPR06	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C41C	M4_SCS0_NVIC_IPR07	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C420	M4_SCS0_NVIC_IPR08	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C424	M4_SCS0_NVIC_IPR09	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C428	M4_SCS0_NVIC_IPR10	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C42C	M4_SCS0_NVIC_IPR11	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C430	M4_SCS0_NVIC_IPR12	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C434	M4_SCS0_NVIC_IPR13	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C438	M4_SCS0_NVIC_IPR14	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C43C	M4_SCS0_NVIC_IPR15	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C440	M4_SCS0_NVIC_IPR16	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C444	M4_SCS0_NVIC_IPR17	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C448	M4_SCS0_NVIC_IPR18	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C44C	M4_SCS0_NVIC_IPR19	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C450	M4_SCS0_NVIC_IPR20	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C454	M4_SCS0_NVIC_IPR21	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C458	M4_SCS0_NVIC_IPR22	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C45C	M4_SCS0_NVIC_IPR23	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C460	M4_SCS0_NVIC_IPR24	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C464	M4_SCS0_NVIC_IPR25	M4_SCS0 Irq Priority Register nn	0x00000000

Table A-37: ADSP-CM40x M4_SCS0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x1C001C468	M4_SCS0_NVIC_IPR26	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C46C	M4_SCS0_NVIC_IPR27	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C470	M4_SCS0_NVIC_IPR28	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C474	M4_SCS0_NVIC_IPR29	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C478	M4_SCS0_NVIC_IPR30	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C47C	M4_SCS0_NVIC_IPR31	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001C480	M4_SCS0_NVIC_IPR32	M4_SCS0 Irq Priority Register nn	0x00000000
0x1C001CD00	M4_SCS0_CPUID	M4_SCS0 CPUID Base Register	0x410FC241
0x1C001CD04	M4_SCS0_ICSR	M4_SCS0 Interrupt Control State Register	0x00000000
0x1C001CD08	M4_SCS0_VTOR	M4_SCS0 Vector Table Offset Register	0x00000000
0x1C001CD0C	M4_SCS0_AIRCR	M4_SCS0 Application Interrupt/Reset Control Register	0xFA050000
0x1C001CD10	M4_SCS0_SCR	M4_SCS0 System Control Register	0x00000000
0x1C001CD14	M4_SCS0_CCR	M4_SCS0 Configuration Control Register	0x00000200
0x1C001CD18	M4_SCS0_SHPR1	M4_SCS0 System Handlers 4-7 Priority Register	0x00000000
0x1C001CD1C	M4_SCS0_SHPR2	M4_SCS0 System Handlers 8-11 Priority Register	0x00000000
0x1C001CD20	M4_SCS0_SHPR3	M4_SCS0 System Handlers 12-15 Priority Register	0x00000000
0x1C001CD24	M4_SCS0_SHCSR	M4_SCS0 System Handler Control and State Register	0x00000000
0x1C001CD28	M4_SCS0_CFSR	M4_SCS0 Configurable Fault Status Registers	0x00000000
0x1C001CD2C	M4_SCS0_HFSR	M4_SCS0 Hard Fault Status Register	0x00000000
0x1C001CD30	M4_SCS0_DFSR	M4_SCS0 Debug Fault Status Register	0x00000000
0x1C001CD34	M4_SCS0_MMFAR	M4_SCS0 Mem Manage Fault Address Register	0x00000000
0x1C001CD38	M4_SCS0_BFAR	M4_SCS0 Bus Fault Address Register	0x00000000
0x1C001CD3C	M4_SCS0_AFSR	M4_SCS0 Auxiliary Fault Status Register	0x00000000
0x1C001CD40	M4_SCS0_ID_PFR0	M4_SCS0 Processor Feature register0	0x00000030
0x1C001CD44	M4_SCS0_ID_PFR1	M4_SCS0 Processor Feature register1	0x00000200
0x1C001CD48	M4_SCS0_ID_DFR0	M4_SCS0 Debug Feature register0	0x00100000
0x1C001CD4C	M4_SCS0_ID_AFR0	M4_SCS0 Auxiliary Feature register0	0x00000000
0x1C001CD50	M4_SCS0_ID_MMFR0	M4_SCS0 Memory Model Feature register0	0x00100030
0x1C001CD54	M4_SCS0_ID_MMFR1	M4_SCS0 Memory Model Feature register1	0x00000000
0x1C001CD58	M4_SCS0_ID_MMFR2	M4_SCS0 Memory Model Feature register2	0x00000000
0x1C001CD5C	M4_SCS0_ID_MMFR3	M4_SCS0 Memory Model Feature register3	0x00000000

Table A-37: ADSP-CM40x M4_SCS0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x1C001CD60	M4_SCS0_ISAR0	M4_SCS0 ISA Feature register0	0x01141110
0x1C001CD64	M4_SCS0_ID_ISAR1	M4_SCS0 ISA Feature register1	0x02112000
0x1C001CD68	M4_SCS0_ID_ISAR2	M4_SCS0 ISA Feature register2	0x20232231
0x1C001CD6C	M4_SCS0_ID_ISAR3	M4_SCS0 ISA Feature register3	0x01111131
0x1C001CD70	M4_SCS0_ID_ISAR4	M4_SCS0 ISA Feature register4	0x01310102
0x1C001CD88	M4_SCS0_CPACR	M4_SCS0 Coprocessor Access Control Register	0x00000000
0x1C001CD90	M4_SCS0_MPU_TYPE	M4_SCS0 MPU Type Register	0x00000800
0x1C001CD94	M4_SCS0_MPU_CTRL	M4_SCS0 MPU Control Register	0x00000000
0x1C001CD98	M4_SCS0_MPU_RNR	M4_SCS0 MPU Region Number Register	0x00000000
0x1C001CD9C	M4_SCS0_MPU_RBAR	M4_SCS0 MPU Region Base Address Register	0x00000000
0x1C001CDA0	M4_SCS0_MPU_RASR	M4_SCS0 MPU Region Attribute and Size Register	0x00000000
0x1C001CDA4	M4_SCS0_MPU_RBAR_A1	M4_SCS0 MPU Alias 1 Region Base Address register	0x00000000
0x1C001CDA8	M4_SCS0_MPU_RASR_A1	M4_SCS0 MPU Alias 1 Region Attribute and Size register	0x00000000
0x1C001CDAC	M4_SCS0_MPU_RBAR_A2	M4_SCS0 MPU Alias 2 Region Base Address register	0x00000000
0x1C001CDB0	M4_SCS0_MPU_RASR_A2	M4_SCS0 MPU Alias 2 Region Attribute and Size register	0x00000000
0x1C001CDB4	M4_SCS0_MPU_RBAR_A3	M4_SCS0 MPU Alias 3 Region Base Address register	0x00000000
0x1C001CDB8	M4_SCS0_MPU_RASR_A3	M4_SCS0 MPU Alias 3 Region Attribute and Size register	0x00000000
0x1C001CDF0	M4_SCS0_DHCSR	M4_SCS0 Debug Halting Control and Status Register	0x00000000
0x1C001CDF4	M4_SCS0_DCRSR	M4_SCS0 Deubg Core Register Selector Register	0x00000000
0x1C001CDF8	M4_SCS0_DCRDR	M4_SCS0 Debug Core Register Data Register	0x00000000
0x1C001CDFC	M4_SCS0_DEMCR	M4_SCS0 Debug Exception and Monitor Control Register	0x00000000
0x1C001CF00	M4_SCS0_STIR	M4_SCS0 Software Trigger Interrupt Register	0x00000000
0x1C001CF34	M4_SCS0_FPCCR	M4_SCS0 Floating Point Context Control Register	0xC0000000
0x1C001CF38	M4_SCS0_FPCAR	M4_SCS0 Floating-Point Context Address Register	0x00000000
0x1C001CF3C	M4_SCS0_FPDSCR	M4_SCS0 Floating Point Default Status Control Register	0x00000000
0x1C001CF40	M4_SCS0_MVFR0	M4_SCS0 Media and FP Feature Register 0 (MVFR0)	0x10110021
0x1C001CF44	M4_SCS0_MVFR1	M4_SCS0 Media and FP Feature Register 1 (MVFR1)	0x11000011
0x1C001CFD0	M4_SCS0_PID4	M4_SCS0 Peripheral identification register (PID4)	0x00000000
0x1C001CFD4	M4_SCS0_PID5	M4_SCS0 Peripheral identification register (PID5)	0x00000000
0x1C001CFD8	M4_SCS0_PID6	M4_SCS0 Peripheral identification register (PID6)	0x00000000

Table A-37: ADSP-CM40x M4_SCS0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x1C001CFDC	M4_SCS0_PID7	M4_SCS0 Peripheral identification register (PID7)	0x00000000
0x1C001CFE0	M4_SCS0_PID0	M4_SCS0 Peripheral identification register Bits 7:0 (PID0)	0x00000000
0x1C001CFE4	M4_SCS0_PID1	M4_SCS0 Peripheral identification register Bits 15:8 (PID1)	0x00000000
0x1C001CFE8	M4_SCS0_PID2	M4_SCS0 Peripheral identification register Bits 23:16 (PID2)	0x00000000
0x1C001CFEC	M4_SCS0_PID3	M4_SCS0 Peripheral identification register Bits 31:24 (PID3)	0x00000000
0x1C001CFF0	M4_SCS0_CID0	M4_SCS0 Component identification register Bits 7:0 (CID0)	0x00000000
0x1C001CFF4	M4_SCS0_CID1	M4_SCS0 Component identification register Bits 15:8 (CID1)	0x00000000
0x1C001CFF8	M4_SCS0_CID2	M4_SCS0 Component identification register Bits 23:16 (CID2)	0x00000000
0x1C001CFFC	M4_SCS0_CID3	M4_SCS0 Component identification register Bits 31:24 (CID3)	0x00000000

Table A-38: ADSP-CM40x M4P0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0xF0000000	M4P0_CACHE_CFG	M4P0 Code Cache Configuration and Status Register	0x0000FC40
0xF0000004	M4P0_CACHE_PEADDR	M4P0 Code Cache Parity Error Address Register	0x00000000
0xF0000008	M4P0_CACHE_MEMX	M4P0 MEMX Space Configuration Register	0x18000003
0xF000000C	M4P0_CACHE_MEMY	M4P0 MEMY Space Configuration Register	0x60000003
0xF0000020	M4P0_SRAM_CFG	M4P0 SRAM Configuration Register	0x00000002
0xF0000024	M4P0_SRAM_PEADDR_CORE	M4P0 SRAM Parity Error Address (Core) Register	0x00000000
0xF0000028	M4P0_SRAM_PEADDR_DMA	M4P0 SRAM Parity Error Address (DMA) Register	0x00000000
0xF000002C	M4P0_BUSFLT	M4P0 Bus Fault Error Information Register	0x00000000
0xF0000030	M4P0_STCALIB	M4P0 SysTick Calibration Register	0x801E8480
0xF0000040	M4P0_CACHE_CNTCTL	M4P0 Cache Counter Control Register	0x00000000
0xF0000044	M4P0_CACHE_IREF	M4P0 Cache ICODE Reference Counter Register	0x00000000
0xF0000048	M4P0_CACHE_DREF	M4P0 Cache DCODE Reference Counter Register	0x00000000

Table A-38: ADSP-CM40x M4P0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0xF000004C	M4P0_CACHE_IMISS	M4P0 Cache ICODE Miss Counter Register	0x00000000
0xF0000050	M4P0_CACHE_DMISS	M4P0 Cache DCODE Miss Counter Register	0x00000000
0xF0000054	M4P0_CACHE_IFILL	M4P0 Cache ICODE Line Fill Counter Register	0x00000000
0xF0000058	M4P0_CACHE_DFILL	M4P0 Cache DCODE Line Fill Counter Register	0x00000000

Table A-39: ADSP-CM40x PADS0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40005404	PADS0_PCFG0	PADS0 Peripheral Configuration0 Register	0x00000001

Table A-40: ADSP-CM40x PINT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40006000	PINT0_MSK_SET	PINT0 Pint Mask Set Register	0x00000000
0x40006004	PINT0_MSK_CLR	PINT0 Pint Mask Clear Register	0x00000000
0x40006008	PINT0_REQ	PINT0 Pint Request Register	0x00000000
0x4000600C	PINT0_ASSIGN	PINT0 Pint Assign Register	0x00000101
0x40006010	PINT0_EDGE_SET	PINT0 Pint Edge Set Register	0x00000000
0x40006014	PINT0_EDGE_CLR	PINT0 Pint Edge Clear Register	0x00000000
0x40006018	PINT0_INV_SET	PINT0 Pint Invert Set Register	0x00000000
0x4000601C	PINT0_INV_CLR	PINT0 Pint Invert Clear Register	0x00000000
0x40006020	PINT0_PINSTATE	PINT0 Pint Pinstate Register	0x00000000
0x40006024	PINT0_LATCH	PINT0 Pint Latch Register	0x00000000

Table A-41: ADSP-CM40x PINT1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40006100	PINT1_MSK_SET	PINT1 Pint Mask Set Register	0x00000000
0x40006104	PINT1_MSK_CLR	PINT1 Pint Mask Clear Register	0x00000000
0x40006108	PINT1_REQ	PINT1 Pint Request Register	0x00000000
0x4000610C	PINT1_ASSIGN	PINT1 Pint Assign Register	0x00000101
0x40006110	PINT1_EDGE_SET	PINT1 Pint Edge Set Register	0x00000000
0x40006114	PINT1_EDGE_CLR	PINT1 Pint Edge Clear Register	0x00000000

Table A-41: ADSP-CM40x PINT1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40006118	PINT1_INV_SET	PINT1 Pint Invert Set Register	0x00000000
0x4000611C	PINT1_INV_CLR	PINT1 Pint Invert Clear Register	0x00000000
0x40006120	PINT1_PINSTATE	PINT1 Pint Pinstate Register	0x00000000
0x40006124	PINT1_LATCH	PINT1 Pint Latch Register	0x00000000

Table A-42: ADSP-CM40x PINT2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40006200	PINT2_MSK_SET	PINT2 Pint Mask Set Register	0x00000000
0x40006204	PINT2_MSK_CLR	PINT2 Pint Mask Clear Register	0x00000000
0x40006208	PINT2_REQ	PINT2 Pint Request Register	0x00000000
0x4000620C	PINT2_ASSIGN	PINT2 Pint Assign Register	0x00000101
0x40006210	PINT2_EDGE_SET	PINT2 Pint Edge Set Register	0x00000000
0x40006214	PINT2_EDGE_CLR	PINT2 Pint Edge Clear Register	0x00000000
0x40006218	PINT2_INV_SET	PINT2 Pint Invert Set Register	0x00000000
0x4000621C	PINT2_INV_CLR	PINT2 Pint Invert Clear Register	0x00000000
0x40006220	PINT2_PINSTATE	PINT2 Pint Pinstate Register	0x00000000
0x40006224	PINT2_LATCH	PINT2 Pint Latch Register	0x00000000

Table A-43: ADSP-CM40x PINT3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40006300	PINT3_MSK_SET	PINT3 Pint Mask Set Register	0x00000000
0x40006304	PINT3_MSK_CLR	PINT3 Pint Mask Clear Register	0x00000000
0x40006308	PINT3_REQ	PINT3 Pint Request Register	0x00000000
0x4000630C	PINT3_ASSIGN	PINT3 Pint Assign Register	0x00000101
0x40006310	PINT3_EDGE_SET	PINT3 Pint Edge Set Register	0x00000000
0x40006314	PINT3_EDGE_CLR	PINT3 Pint Edge Clear Register	0x00000000
0x40006318	PINT3_INV_SET	PINT3 Pint Invert Set Register	0x00000000
0x4000631C	PINT3_INV_CLR	PINT3 Pint Invert Clear Register	0x00000000
0x40006320	PINT3_PINSTATE	PINT3 Pint Pinstate Register	0x00000000
0x40006324	PINT3_LATCH	PINT3 Pint Latch Register	0x00000000

Table A-44: ADSP-CM40x PINT4 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40006400	PINT4_MSK_SET	PINT4 Pint Mask Set Register	0x00000000
0x40006404	PINT4_MSK_CLR	PINT4 Pint Mask Clear Register	0x00000000
0x40006408	PINT4_REQ	PINT4 Pint Request Register	0x00000000
0x4000640C	PINT4_ASSIGN	PINT4 Pint Assign Register	0x00000101
0x40006410	PINT4_EDGE_SET	PINT4 Pint Edge Set Register	0x00000000
0x40006414	PINT4_EDGE_CLR	PINT4 Pint Edge Clear Register	0x00000000
0x40006418	PINT4_INV_SET	PINT4 Pint Invert Set Register	0x00000000
0x4000641C	PINT4_INV_CLR	PINT4 Pint Invert Clear Register	0x00000000
0x40006420	PINT4_PINSTATE	PINT4 Pint Pinstate Register	0x00000000
0x40006424	PINT4_LATCH	PINT4 Pint Latch Register	0x00000000

Table A-45: ADSP-CM40x PORT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40005000	PORT_FER	PORT0 Port x Function Enable Register	0x00000000
0x40005004	PORT_FER_SET	PORT0 Port x Function Enable Set Register	0x00000000
0x40005008	PORT_FER_CLR	PORT0 Port x Function Enable Clear Register	0x00000000
0x4000500C	PORT_DATA	PORT0 Port x GPIO Data Register	0x00000000
0x40005010	PORT_DATA_SET	PORT0 Port x GPIO Data Set Register	0x00000000
0x40005014	PORT_DATA_CLR	PORT0 Port x GPIO Data Clear Register	0x00000000
0x40005018	PORT_DIR	PORT0 Port x GPIO Direction Register	0x00000000
0x4000501C	PORT_DIR_SET	PORT0 Port x GPIO Direction Set Register	0x00000000
0x40005020	PORT_DIR_CLR	PORT0 Port x GPIO Direction Clear Register	0x00000000
0x40005024	PORT_INEN	PORT0 Port x GPIO Input Enable Register	0x00000000
0x40005028	PORT_INEN_SET	PORT0 Port x GPIO Input Enable Set Register	0x00000000
0x4000502C	PORT_INEN_CLR	PORT0 Port x GPIO Input Enable Clear Register	0x00000000
0x40005030	PORT_MUX	PORT0 Port x Multiplexer Control Register	0x00000000
0x40005034	PORT_DATA_TGL	PORT0 Port x GPIO Output Toggle Register	0x00000000
0x40005038	PORT_POL	PORT0 Port x GPIO Polarity Invert Register	0x00000000
0x4000503C	PORT_POL_SET	PORT0 Port x GPIO Polarity Invert Set Register	0x00000000
0x40005040	PORT_POL_CLR	PORT0 Port x GPIO Polarity Invert Clear Register	0x00000000

Table A-45: ADSP-CM40x PORT0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40005044	PORT_LOCK	PORT0 Port x GPIO Lock Register	0x00000000

Table A-46: ADSP-CM40x PWM0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40008000	PWM0_CTL	PWM0 Control Register	0x00020000
0x40008004	PWM0_CHANCFG	PWM0 Channel Configuration Register	0x00000000
0x40008008	PWM0_TRIPCFG	PWM0 Trip Configuration Register	0x00000000
0x4000800C	PWM0_STAT	PWM0 Status Register	0x00000000
0x40008010	PWM0_IMSK	PWM0 Interrupt Mask Register	0x00000000
0x40008014	PWM0_ILAT	PWM0 Interrupt Latch Register	0x00000000
0x40008018	PWM0_CHOPCFG	PWM0 Chop Configuration Register	0x00000000
0x4000801C	PWM0_DT	PWM0 Dead Time Register	0x00000000
0x40008020	PWM0_SYNC_WID	PWM0 Sync Pulse Width Register	0x000003FF
0x40008024	PWM0_TM0	PWM0 Timer 0 Period Register	0x00000000
0x40008028	PWM0_TM1	PWM0 Timer 1 Period Register	0x00000000
0x4000802C	PWM0_TM2	PWM0 Timer 2 Period Register	0x00000000
0x40008030	PWM0_TM3	PWM0 Timer 3 Period Register	0x00000000
0x40008034	PWM0_TM4	PWM0 Timer 4 Period Register	0x00000000
0x40008038	PWM0_DLYA	PWM0 Channel A Delay Register	0x00000000
0x4000803C	PWM0_DLYB	PWM0 Channel B Delay Register	0x00000000
0x40008040	PWM0_DLYC	PWM0 Channel C Delay Register	0x00000000
0x40008044	PWM0_DLYD	PWM0 Channel D Delay Register	0x00000000
0x40008048	PWM0_ACTL	PWM0 Channel A Control Register	0x00000000
0x4000804C	PWM0_AH0	PWM0 Channel A-High Duty-0 Register	0x00000000
0x40008050	PWM0_AH1	PWM0 Channel A-High Duty-1 Register	0x00000000
0x40008054	PWM0_AH0_HP	PWM0 Channel A-High Heightened-Precision Duty-0 Register	0x00000000
0x40008058	PWM0_AH1_HP	PWM0 Channel A-High Heightened-Precision Duty-1 Register	0x00000000
0x4000805C	PWM0_AL0	PWM0 Channel A-Low Duty-0 Register	0x00000000
0x40008060	PWM0_AL1	PWM0 Channel A-Low Duty-1 Register	0x00000000

Table A-46: ADSP-CM40x PWM0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40008064	PWM0_AL0_HP	PWM0 Channel A-Low Heightened-Precision Duty-0 Register	0x00000000
0x40008068	PWM0_AL1_HP	PWM0 Channel A-Low Heightened-Precision Duty-1 Register	0x00000000
0x4000806C	PWM0_BCTL	PWM0 Channel B Control Register	0x00000000
0x40008070	PWM0_BH0	PWM0 Channel B-High Duty-0 Register	0x00000000
0x40008074	PWM0_BH1	PWM0 Channel B-High Duty-1 Register	0x00000000
0x40008078	PWM0_BH0_HP	PWM0 Channel B-High Heightened-Precision Duty-0 Register	0x00000000
0x4000807C	PWM0_BH1_HP	PWM0 Channel B-High Heightened-Precision Duty-1 Register	0x00000000
0x40008080	PWM0_BL0	PWM0 Channel B-Low Duty-0 Register	0x00000000
0x40008084	PWM0_BL1	PWM0 Channel B-Low Duty-1 Register	0x00000000
0x40008088	PWM0_BL0_HP	PWM0 Channel B-Low Heightened-Precision Duty-0 Register	0x00000000
0x4000808C	PWM0_BL1_HP	PWM0 Channel B-Low Heightened-Precision Duty-1 Register	0x00000000
0x40008090	PWM0_CCTL	PWM0 Channel C Control Register	0x00000000
0x40008094	PWM0_CH0	PWM0 Channel C-High Pulse Duty Register 0	0x00000000
0x40008098	PWM0_CH1	PWM0 Channel C-High Pulse Duty Register 1	0x00000000
0x4000809C	PWM0_CH0_HP	PWM0 Channel C-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x400080A0	PWM0_CH1_HP	PWM0 Channel C-High Pulse Heightened-Precision Duty Register 1	0x00000000
0x400080A4	PWM0_CL0	PWM0 Channel C-Low Pulse Duty Register 0	0x00000000
0x400080A8	PWM0_CL1	PWM0 Channel C-Low Duty-1 Register	0x00000000
0x400080AC	PWM0_CL0_HP	PWM0 Channel C-Low Pulse Duty Register 1	0x00000000
0x400080B0	PWM0_CL1_HP	PWM0 Channel C-Low Heightened-Precision Duty-1 Register	0x00000000
0x400080B4	PWM0_DCTL	PWM0 Channel D Control Register	0x00000000
0x400080B8	PWM0_DH0	PWM0 Channel D-High Duty-0 Register	0x00000000
0x400080BC	PWM0_DH1	PWM0 Channel D-High Pulse Duty Register 1	0x00000000
0x400080C0	PWM0_DH0_HP	PWM0 Channel D-High Pulse Heightened-Precision Duty Register 0	0x00000000

Table A-46: ADSP-CM40x PWM0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x400080C4	PWM0_DH1_HP	PWM0 Channel D High Pulse Heightened-Precision Duty Register 1	0x00000000
0x400080C8	PWM0_DL0	PWM0 Channel D-Low Pulse Duty Register 0	0x00000000
0x400080CC	PWM0_DL1	PWM0 Channel D-Low Pulse Duty Register 1	0x00000000
0x400080D0	PWM0_DL0_HP	PWM0 Channel D-Low Heightened-Precision Duty-0 Register	0x00000000
0x400080D4	PWM0_DL1_HP	PWM0 Channel D-Low Heightened-Precision Duty-1 Register	0x00000000
0x400080D8	PWM0_AH_DUTY0	PWM0 Channel A-High Full Duty0 Register	0x00000000
0x400080DC	PWM0_AH_DUTY1	PWM0 Channel A-High Full Duty1 Register	0x00000000
0x400080E0	PWM0_AL_DUTY0	PWM0 Channel A-Low Full Duty0 Register	0x00000000
0x400080E4	PWM0_AL_DUTY1	PWM0 Channel A-Low Full Duty1 Register	0x00000000
0x400080E8	PWM0_BH_DUTY0	PWM0 Channel B-High Full Duty0 Register	0x00000000
0x400080EC	PWM0_BH_DUTY1	PWM0 Channel B-High Full Duty1 Register	0x00000000
0x400080F0	PWM0_BL_DUTY0	PWM0 Channel B-Low Full Duty0 Register	0x00000000
0x400080F4	PWM0_BL_DUTY1	PWM0 Channel B-Low Full Duty1 Register	0x00000000
0x400080F8	PWM0_CH_DUTY0	PWM0 Channel C-High Full Duty0 Register	0x00000000
0x400080FC	PWM0_CH_DUTY1	PWM0 Channel C-High Full Duty1 Register	0x00000000
0x40008100	PWM0_CL_DUTY0	PWM0 Channel C-Low Full Duty0 Register	0x00000000
0x40008104	PWM0_CL_DUTY1	PWM0 Channel C-Low Full Duty1 Register	0x00000000
0x40008108	PWM0_DH_DUTY0	PWM0 Channel D-High Full Duty0 Register	0x00000000
0x4000810C	PWM0_DH_DUTY1	PWM0 Channel D-High Full Duty1 Register	0x00000000
0x40008110	PWM0_DL_DUTY0	PWM0 Channel D-Low Full Duty0 Register	0x00000000
0x40008114	PWM0_DL_DUTY1	PWM0 Channel D-Low Full Duty1 Register	0x00000000

Table A-47: ADSP-CM40x PWM1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40008400	PWM1_CTL	PWM1 Control Register	0x00020000
0x40008404	PWM1_CHANCFG	PWM1 Channel Configuration Register	0x00000000
0x40008408	PWM1_TRIPCFG	PWM1 Trip Configuration Register	0x00000000
0x4000840C	PWM1_STAT	PWM1 Status Register	0x00000000

Table A-47: ADSP-CM40x PWM1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40008410	PWM1_IMSK	PWM1 Interrupt Mask Register	0x00000000
0x40008414	PWM1_ILAT	PWM1 Interrupt Latch Register	0x00000000
0x40008418	PWM1_CHOPCFG	PWM1 Chop Configuration Register	0x00000000
0x4000841C	PWM1_DT	PWM1 Dead Time Register	0x00000000
0x40008420	PWM1_SYNC_WID	PWM1 Sync Pulse Width Register	0x000003FF
0x40008424	PWM1_TM0	PWM1 Timer 0 Period Register	0x00000000
0x40008428	PWM1_TM1	PWM1 Timer 1 Period Register	0x00000000
0x4000842C	PWM1_TM2	PWM1 Timer 2 Period Register	0x00000000
0x40008430	PWM1_TM3	PWM1 Timer 3 Period Register	0x00000000
0x40008434	PWM1_TM4	PWM1 Timer 4 Period Register	0x00000000
0x40008438	PWM1_DLYA	PWM1 Channel A Delay Register	0x00000000
0x4000843C	PWM1_DLYB	PWM1 Channel B Delay Register	0x00000000
0x40008440	PWM1_DLYC	PWM1 Channel C Delay Register	0x00000000
0x40008444	PWM1_DLYD	PWM1 Channel D Delay Register	0x00000000
0x40008448	PWM1_ACTL	PWM1 Channel A Control Register	0x00000000
0x4000844C	PWM1_AH0	PWM1 Channel A-High Duty-0 Register	0x00000000
0x40008450	PWM1_AH1	PWM1 Channel A-High Duty-1 Register	0x00000000
0x40008454	PWM1_AH0_HP	PWM1 Channel A-High Heightened-Precision Duty-0 Register	0x00000000
0x40008458	PWM1_AH1_HP	PWM1 Channel A-High Heightened-Precision Duty-1 Register	0x00000000
0x4000845C	PWM1_AL0	PWM1 Channel A-Low Duty-0 Register	0x00000000
0x40008460	PWM1_AL1	PWM1 Channel A-Low Duty-1 Register	0x00000000
0x40008464	PWM1_AL0_HP	PWM1 Channel A-Low Heightened-Precision Duty-0 Register	0x00000000
0x40008468	PWM1_AL1_HP	PWM1 Channel A-Low Heightened-Precision Duty-1 Register	0x00000000
0x4000846C	PWM1_BCTL	PWM1 Channel B Control Register	0x00000000
0x40008470	PWM1_BH0	PWM1 Channel B-High Duty-0 Register	0x00000000
0x40008474	PWM1_BH1	PWM1 Channel B-High Duty-1 Register	0x00000000
0x40008478	PWM1_BH0_HP	PWM1 Channel B-High Heightened-Precision Duty-0 Register	0x00000000

Table A-47: ADSP-CM40x PWM1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4000847C	PWM1_BH1_HP	PWM1 Channel B-High Heightened-Precision Duty-1 Register	0x00000000
0x40008480	PWM1_BL0	PWM1 Channel B-Low Duty-0 Register	0x00000000
0x40008484	PWM1_BL1	PWM1 Channel B-Low Duty-1 Register	0x00000000
0x40008488	PWM1_BL0_HP	PWM1 Channel B-Low Heightened-Precision Duty-0 Register	0x00000000
0x4000848C	PWM1_BL1_HP	PWM1 Channel B-Low Heightened-Precision Duty-1 Register	0x00000000
0x40008490	PWM1_CCTL	PWM1 Channel C Control Register	0x00000000
0x40008494	PWM1_CH0	PWM1 Channel C-High Pulse Duty Register 0	0x00000000
0x40008498	PWM1_CH1	PWM1 Channel C-High Pulse Duty Register 1	0x00000000
0x4000849C	PWM1_CH0_HP	PWM1 Channel C-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x400084A0	PWM1_CH1_HP	PWM1 Channel C-High Pulse Heightened-Precision Duty Register 1	0x00000000
0x400084A4	PWM1_CL0	PWM1 Channel C-Low Pulse Duty Register 0	0x00000000
0x400084A8	PWM1_CL1	PWM1 Channel C-Low Duty-1 Register	0x00000000
0x400084AC	PWM1_CL0_HP	PWM1 Channel C-Low Pulse Duty Register 1	0x00000000
0x400084B0	PWM1_CL1_HP	PWM1 Channel C-Low Heightened-Precision Duty-1 Register	0x00000000
0x400084B4	PWM1_DCTL	PWM1 Channel D Control Register	0x00000000
0x400084B8	PWM1_DH0	PWM1 Channel D-High Duty-0 Register	0x00000000
0x400084BC	PWM1_DH1	PWM1 Channel D-High Pulse Duty Register 1	0x00000000
0x400084C0	PWM1_DH0_HP	PWM1 Channel D-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x400084C4	PWM1_DH1_HP	PWM1 Channel D High Pulse Heightened-Precision Duty Register 1	0x00000000
0x400084C8	PWM1_DL0	PWM1 Channel D-Low Pulse Duty Register 0	0x00000000
0x400084CC	PWM1_DL1	PWM1 Channel D-Low Pulse Duty Register 1	0x00000000
0x400084D0	PWM1_DL0_HP	PWM1 Channel D-Low Heightened-Precision Duty-0 Register	0x00000000
0x400084D4	PWM1_DL1_HP	PWM1 Channel D-Low Heightened-Precision Duty-1 Register	0x00000000
0x400084D8	PWM1_AH_DUTY0	PWM1 Channel A-High Full Duty0 Register	0x00000000

Table A-47: ADSP-CM40x PWM1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x400084DC	PWM1_AH_DUTY1	PWM1 Channel A-High Full Duty1 Register	0x00000000
0x400084E0	PWM1_AL_DUTY0	PWM1 Channel A-Low Full Duty0 Register	0x00000000
0x400084E4	PWM1_AL_DUTY1	PWM1 Channel A-Low Full Duty1 Register	0x00000000
0x400084E8	PWM1_BH_DUTY0	PWM1 Channel B-High Full Duty0 Register	0x00000000
0x400084EC	PWM1_BH_DUTY1	PWM1 Channel B-High Full Duty1 Register	0x00000000
0x400084F0	PWM1_BL_DUTY0	PWM1 Channel B-Low Full Duty0 Register	0x00000000
0x400084F4	PWM1_BL_DUTY1	PWM1 Channel B-Low Full Duty1 Register	0x00000000
0x400084F8	PWM1_CH_DUTY0	PWM1 Channel C-High Full Duty0 Register	0x00000000
0x400084FC	PWM1_CH_DUTY1	PWM1 Channel C-High Full Duty1 Register	0x00000000
0x40008500	PWM1_CL_DUTY0	PWM1 Channel C-Low Full Duty0 Register	0x00000000
0x40008504	PWM1_CL_DUTY1	PWM1 Channel C-Low Full Duty1 Register	0x00000000
0x40008508	PWM1_DH_DUTY0	PWM1 Channel D-High Full Duty0 Register	0x00000000
0x4000850C	PWM1_DH_DUTY1	PWM1 Channel D-High Full Duty1 Register	0x00000000
0x40008510	PWM1_DL_DUTY0	PWM1 Channel D-Low Full Duty0 Register	0x00000000
0x40008514	PWM1_DL_DUTY1	PWM1 Channel D-Low Full Duty1 Register	0x00000000

Table A-48: ADSP-CM40x PWM2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40008800	PWM2_CTL	PWM2 Control Register	0x00020000
0x40008804	PWM2_CHANCFG	PWM2 Channel Configuration Register	0x00000000
0x40008808	PWM2_TRIPCFG	PWM2 Trip Configuration Register	0x00000000
0x4000880C	PWM2_STAT	PWM2 Status Register	0x00000000
0x40008810	PWM2_IMSK	PWM2 Interrupt Mask Register	0x00000000
0x40008814	PWM2_ILAT	PWM2 Interrupt Latch Register	0x00000000
0x40008818	PWM2_CHOPCFG	PWM2 Chop Configuration Register	0x00000000
0x4000881C	PWM2_DT	PWM2 Dead Time Register	0x00000000
0x40008820	PWM2_SYNC_WID	PWM2 Sync Pulse Width Register	0x000003FF
0x40008824	PWM2_TM0	PWM2 Timer 0 Period Register	0x00000000
0x40008828	PWM2_TM1	PWM2 Timer 1 Period Register	0x00000000
0x4000882C	PWM2_TM2	PWM2 Timer 2 Period Register	0x00000000

Table A-48: ADSP-CM40x PWM2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40008830	PWM2_TM3	PWM2 Timer 3 Period Register	0x00000000
0x40008834	PWM2_TM4	PWM2 Timer 4 Period Register	0x00000000
0x40008838	PWM2_DLYA	PWM2 Channel A Delay Register	0x00000000
0x4000883C	PWM2_DLYB	PWM2 Channel B Delay Register	0x00000000
0x40008840	PWM2_DLYC	PWM2 Channel C Delay Register	0x00000000
0x40008844	PWM2_DLYD	PWM2 Channel D Delay Register	0x00000000
0x40008848	PWM2_ACTL	PWM2 Channel A Control Register	0x00000000
0x4000884C	PWM2_AH0	PWM2 Channel A-High Duty-0 Register	0x00000000
0x40008850	PWM2_AH1	PWM2 Channel A-High Duty-1 Register	0x00000000
0x40008854	PWM2_AH0_HP	PWM2 Channel A-High Heightened-Precision Duty-0 Register	0x00000000
0x40008858	PWM2_AH1_HP	PWM2 Channel A-High Heightened-Precision Duty-1 Register	0x00000000
0x4000885C	PWM2_AL0	PWM2 Channel A-Low Duty-0 Register	0x00000000
0x40008860	PWM2_AL1	PWM2 Channel A-Low Duty-1 Register	0x00000000
0x40008864	PWM2_AL0_HP	PWM2 Channel A-Low Heightened-Precision Duty-0 Register	0x00000000
0x40008868	PWM2_AL1_HP	PWM2 Channel A-Low Heightened-Precision Duty-1 Register	0x00000000
0x4000886C	PWM2_BCTL	PWM2 Channel B Control Register	0x00000000
0x40008870	PWM2_BH0	PWM2 Channel B-High Duty-0 Register	0x00000000
0x40008874	PWM2_BH1	PWM2 Channel B-High Duty-1 Register	0x00000000
0x40008878	PWM2_BH0_HP	PWM2 Channel B-High Heightened-Precision Duty-0 Register	0x00000000
0x4000887C	PWM2_BH1_HP	PWM2 Channel B-High Heightened-Precision Duty-1 Register	0x00000000
0x40008880	PWM2_BL0	PWM2 Channel B-Low Duty-0 Register	0x00000000
0x40008884	PWM2_BL1	PWM2 Channel B-Low Duty-1 Register	0x00000000
0x40008888	PWM2_BL0_HP	PWM2 Channel B-Low Heightened-Precision Duty-0 Register	0x00000000
0x4000888C	PWM2_BL1_HP	PWM2 Channel B-Low Heightened-Precision Duty-1 Register	0x00000000
0x40008890	PWM2_CCTL	PWM2 Channel C Control Register	0x00000000

Table A-48: ADSP-CM40x PWM2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40008894	PWM2_CH0	PWM2 Channel C-High Pulse Duty Register 0	0x00000000
0x40008898	PWM2_CH1	PWM2 Channel C-High Pulse Duty Register 1	0x00000000
0x4000889C	PWM2_CH0_HP	PWM2 Channel C-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x400088A0	PWM2_CH1_HP	PWM2 Channel C-High Pulse Heightened-Precision Duty Register 1	0x00000000
0x400088A4	PWM2_CL0	PWM2 Channel C-Low Pulse Duty Register 0	0x00000000
0x400088A8	PWM2_CL1	PWM2 Channel C-Low Duty-1 Register	0x00000000
0x400088AC	PWM2_CL0_HP	PWM2 Channel C-Low Pulse Duty Register 1	0x00000000
0x400088B0	PWM2_CL1_HP	PWM2 Channel C-Low Heightened-Precision Duty-1 Register	0x00000000
0x400088B4	PWM2_DCTL	PWM2 Channel D Control Register	0x00000000
0x400088B8	PWM2_DH0	PWM2 Channel D-High Duty-0 Register	0x00000000
0x400088BC	PWM2_DH1	PWM2 Channel D-High Pulse Duty Register 1	0x00000000
0x400088C0	PWM2_DH0_HP	PWM2 Channel D-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x400088C4	PWM2_DH1_HP	PWM2 Channel D High Pulse Heightened-Precision Duty Register 1	0x00000000
0x400088C8	PWM2_DL0	PWM2 Channel D-Low Pulse Duty Register 0	0x00000000
0x400088CC	PWM2_DL1	PWM2 Channel D-Low Pulse Duty Register 1	0x00000000
0x400088D0	PWM2_DL0_HP	PWM2 Channel D-Low Heightened-Precision Duty-0 Register	0x00000000
0x400088D4	PWM2_DL1_HP	PWM2 Channel D-Low Heightened-Precision Duty-1 Register	0x00000000
0x400088D8	PWM2_AH_DUTY0	PWM2 Channel A-High Full Duty0 Register	0x00000000
0x400088DC	PWM2_AH_DUTY1	PWM2 Channel A-High Full Duty1 Register	0x00000000
0x400088E0	PWM2_AL_DUTY0	PWM2 Channel A-Low Full Duty0 Register	0x00000000
0x400088E4	PWM2_AL_DUTY1	PWM2 Channel A-Low Full Duty1 Register	0x00000000
0x400088E8	PWM2_BH_DUTY0	PWM2 Channel B-High Full Duty0 Register	0x00000000
0x400088EC	PWM2_BH_DUTY1	PWM2 Channel B-High Full Duty1 Register	0x00000000
0x400088F0	PWM2_BL_DUTY0	PWM2 Channel B-Low Full Duty0 Register	0x00000000
0x400088F4	PWM2_BL_DUTY1	PWM2 Channel B-Low Full Duty1 Register	0x00000000
0x400088F8	PWM2_CH_DUTY0	PWM2 Channel C-High Full Duty0 Register	0x00000000

Table A-48: ADSP-CM40x PWM2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x400088FC	PWM2_CH_DUTY1	PWM2 Channel C-High Full Duty1 Register	0x00000000
0x40008900	PWM2_CL_DUTY0	PWM2 Channel C-Low Full Duty0 Register	0x00000000
0x40008904	PWM2_CL_DUTY1	PWM2 Channel C-Low Full Duty1 Register	0x00000000
0x40008908	PWM2_DH_DUTY0	PWM2 Channel D-High Full Duty0 Register	0x00000000
0x4000890C	PWM2_DH_DUTY1	PWM2 Channel D-High Full Duty1 Register	0x00000000
0x40008910	PWM2_DL_DUTY0	PWM2 Channel D-Low Full Duty0 Register	0x00000000
0x40008914	PWM2_DL_DUTY1	PWM2 Channel D-Low Full Duty1 Register	0x00000000

Table A-49: ADSP-CM40x RCU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40014000	RCU0_CTL	RCU0 Control Register	0x00000700
0x40014004	RCU0_STAT	RCU0 Status Register	0x00000021
0x40014008	RCU0_CRCTL	RCU0 Core Reset outputs Control Register	0x00000000
0x4001400C	RCU0_CRSTAT	RCU0 Core Reset outputs Status Register	0x00000000
0x40014018	RCU0_SRRQSTAT	RCU0 System Reset Request Status	0x00000000
0x40014024	RCU0_SVECT_LCK	RCU0 SVECT Lock Register	0x00000000
0x40014028	RCU0_BCODE	RCU0 Boot Code Register	0x00000000
0x4001402C	RCU0_SVECT0	RCU0 Software Vector Register 0	0x00000001
0x4001406C	RCU0_MSG	RCU0 Message Register	0x00000000
0x40014070	RCU0_MSG_SET	RCU0 Message Set Bits Register	0x00000000
0x40014074	RCU0_MSG_CLR	RCU0 Message Clear Bits Register	0x00000000
0x4001407C	RCU0_REVID	RCU0 Revision ID Register	0x00000030

Table A-50: ADSP-CM40x SCB0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x80242020	SCB0_MST00_IB_SYNC	SCB0 Master 0 IB Sync Mode	0x00000004
0x80242100	SCB0_MST00_IB_RQOS	SCB0 Master 0 Read Quality of Service	0x00000001
0x80242104	SCB0_MST00_IB_WQOS	SCB0 Master 0 Write Quality of Service	0x00000001
0x80243020	SCB0_MST01_IB_SYNC	SCB0 Master 1 IB Sync Mode	0x00000004
0x80243100	SCB0_MST01_IB_RQOS	SCB0 Master 1 Read Quality of Service	0x00000001

Table A-50: ADSP-CM40x SCB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x80243104	SCB0_MST01_IB_WQOS	SCB0 Master 1 Write Quality of Service	0x00000001
0x80244100	SCB0_MST02_RQOS	SCB0 Master 2 Read Quality of Service	0x00000001
0x80244104	SCB0_MST02_WQOS	SCB0 Master 2 Write Quality of Service	0x00000001
0x80245100	SCB0_MST03_RQOS	SCB0 Master 3 Read Quality of Service	0x00000001
0x80245104	SCB0_MST03_WQOS	SCB0 Master 3 Write Quality of Service	0x00000001
0x80246100	SCB0_MST04_RQOS	SCB0 Master 4 Read Quality of Service	0x00000001
0x80246104	SCB0_MST04_WQOS	SCB0 Master 4 Write Quality of Service	0x00000001
0x80247100	SCB0_MST05_RQOS	SCB0 Master 5 Read Quality of Service	0x00000001
0x80247104	SCB0_MST05_WQOS	SCB0 Master 5 Write Quality of Service	0x00000001
0x80248100	SCB0_MST06_RQOS	SCB0 Master 6 Read Quality of Service	0x00000001
0x80248104	SCB0_MST06_WQOS	SCB0 Master 6 Write Quality of Service	0x00000001
0x80249100	SCB0_MST07_RQOS	SCB0 Master 7 Read Quality of Service	0x00000001
0x80249104	SCB0_MST07_WQOS	SCB0 Master 7 Write Quality of Service	0x00000001
0x8024A100	SCB0_MST08_RQOS	SCB0 Master 8 Read Quality of Service	0x00000001
0x8024A104	SCB0_MST08_WQOS	SCB0 Master 8 Write Quality of Service	0x00000001
0x8024B100	SCB0_MST09_RQOS	SCB0 Master 9 Read Quality of Service	0x00000001
0x8024B104	SCB0_MST09_WQOS	SCB0 Master 9 Write Quality of Service	0x00000001
0x8024C100	SCB0_MST10_RQOS	SCB0 Master 10 Read Quality of Service	0x00000001
0x8024C104	SCB0_MST10_WQOS	SCB0 Master 10 Write Quality of Service	0x00000001
0x8024D100	SCB0_MST11_RQOS	SCB0 Master 11 Read Quality of Service	0x00000001
0x8024D104	SCB0_MST11_WQOS	SCB0 Master 11 Write Quality of Service	0x00000001
0x8024E100	SCB0_MST12_RQOS	SCB0 Master 12 Read Quality of Service	0x00000001
0x8024E104	SCB0_MST12_WQOS	SCB0 Master 12 Write Quality of Service	0x00000001
0x8024F100	SCB0_MST13_RQOS	SCB0 Master 13 Read Quality of Service	0x00000001
0x8024F104	SCB0_MST13_WQOS	SCB0 Master 13 Write Quality of Service	0x00000001
0x80250100	SCB0_MST14_RQOS	SCB0 Master 14 Read Quality of Service	0x00000001
0x80250104	SCB0_MST14_WQOS	SCB0 Master 14 Write Quality of Service	0x00000001
0x80251100	SCB0_MST15_RQOS	SCB0 Master 15 Read Quality of Service	0x00000001
0x80251104	SCB0_MST15_WQOS	SCB0 Master 15 Write Quality of Service	0x00000001
0x80252100	SCB0_MST16_RQOS	SCB0 Master 16 Read Quality of Service	0x00000001
0x80252104	SCB0_MST16_WQOS	SCB0 Master 16 Write Quality of Service	0x00000001

Table A-50: ADSP-CM40x SCB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x80253100	SCB0_MST17_RQOS	SCB0 Master 17 Read Quality of Service	0x00000000
0x80253104	SCB0_MST17_WQOS	SCB0 Master 17 Write Quality of Service	0x00000000
0x80254100	SCB0_MST18_RQOS	SCB0 Master 18 Read Quality of Service	0x00000000
0x80254104	SCB0_MST18_WQOS	SCB0 Master 18 Write Quality of Service	0x00000000
0x80255100	SCB0_MST19_RQOS	SCB0 Master 19 Read Quality of Service	0x00000001
0x80255104	SCB0_MST19_WQOS	SCB0 Master 19 Write Quality of Service	0x00000001
0x80256100	SCB0_MST20_RQOS	SCB0 Master20 Read Quality of Service	0x00000001
0x80256104	SCB0_MST20_WQOS	SCB0 Master 20 Write Quality of Service	0x00000001
0x80257100	SCB0_MST21_RQOS	SCB0 Master 21 Read Quality of Service	0x00000001
0x80257104	SCB0_MST21_WQOS	SCB0 Master 21 Write Quality of Service	0x00000001
0x80258100	SCB0_MST22_RQOS	SCB0 Master 22 Read Quality of Service	0x00000001
0x80258104	SCB0_MST22_WQOS	SCB0 Master 22 Write Quality of Service	0x00000001
0x80259100	SCB0_MST23_RQOS	SCB0 Master 23 Read Quality of Service	0x00000001
0x80259104	SCB0_MST23_WQOS	SCB0 Master 23 Write Quality of Service	0x00000001
0x8025A100	SCB0_MST24_RQOS	SCB0 Master 24 Read Quality of Service	0x00000001
0x8025A104	SCB0_MST24_WQOS	SCB0 Master 24 Write Quality of Service	0x00000001
0x8025B100	SCB0_MST25_RQOS	SCB0 Master 25 Read Quality of Service	0x00000000
0x8025B104	SCB0_MST25_WQOS	SCB0 Master 25 Write Quality of Service	0x00000000
0x8025C100	SCB0_MST26_RQOS	SCB0 Master 26 Read Quality of Service	0x00000000
0x8025C104	SCB0_MST26_WQOS	SCB0 Master 26 Write Quality of Service	0x00000000

Table A-51: ADSP-CM40x SEC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40028000	SEC0_GCTL	SEC0 Global Control Register	0x00000000
0x40028004	SEC0_GSTAT	SEC0 Global Status Register	0x00000000
0x40028008	SEC0_RAISE	SEC0 Global Raise Register	0x00000000
0x40028010	SEC0_FCTL	SEC0 Fault Control Register	0x00000000
0x40028014	SEC0_FSTAT	SEC0 Fault Status Register	0x00000000
0x40028018	SEC0_FSID	SEC0 Fault Source ID Register	0x00000000
0x4002801C	SEC0_FEND	SEC0 Fault End Register	0x00000000

Table A-51: ADSP-CM40x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40028020	SEC0_FDLY	SEC0 Fault Delay Register	0x00000000
0x40028024	SEC0_FDLY_CUR	SEC0 Fault Delay Current Register	0x00000000
0x40028028	SEC0_FSRDLY	SEC0 Fault System Reset Delay Register	0x00000000
0x4002802C	SEC0_FSRDLY_CUR	SEC0 Fault System Reset Delay Current Register	0x00000000
0x40028030	SEC0_FCOPP	SEC0 Fault COP Period Register	0x00000000
0x40028034	SEC0_FCOPP_CUR	SEC0 Fault COP Period Current Register	0x00000000
0x40028800	SEC0_SCTL0	SEC0 Source Control Register n	0x00000000
0x40028804	SEC0_SSTAT0	SEC0 Source Status Register n	0x00000000
0x40028808	SEC0_SCTL1	SEC0 Source Control Register n	0x00000000
0x4002880C	SEC0_SSTAT1	SEC0 Source Status Register n	0x00000000
0x40028810	SEC0_SCTL2	SEC0 Source Control Register n	0x00000000
0x40028814	SEC0_SSTAT2	SEC0 Source Status Register n	0x00000000
0x40028818	SEC0_SCTL3	SEC0 Source Control Register n	0x00000000
0x4002881C	SEC0_SSTAT3	SEC0 Source Status Register n	0x00000000
0x40028820	SEC0_SCTL4	SEC0 Source Control Register n	0x00000000
0x40028824	SEC0_SSTAT4	SEC0 Source Status Register n	0x00000000
0x40028828	SEC0_SCTL5	SEC0 Source Control Register n	0x00000000
0x4002882C	SEC0_SSTAT5	SEC0 Source Status Register n	0x00000000
0x40028830	SEC0_SCTL6	SEC0 Source Control Register n	0x00000000
0x40028834	SEC0_SSTAT6	SEC0 Source Status Register n	0x00000000
0x40028838	SEC0_SCTL7	SEC0 Source Control Register n	0x00000000
0x4002883C	SEC0_SSTAT7	SEC0 Source Status Register n	0x00000000
0x40028840	SEC0_SCTL8	SEC0 Source Control Register n	0x00000000
0x40028844	SEC0_SSTAT8	SEC0 Source Status Register n	0x00000000
0x40028848	SEC0_SCTL9	SEC0 Source Control Register n	0x00000000
0x4002884C	SEC0_SSTAT9	SEC0 Source Status Register n	0x00000000
0x40028850	SEC0_SCTL10	SEC0 Source Control Register n	0x00000000
0x40028854	SEC0_SSTAT10	SEC0 Source Status Register n	0x00000000
0x40028858	SEC0_SCTL11	SEC0 Source Control Register n	0x00000000
0x4002885C	SEC0_SSTAT11	SEC0 Source Status Register n	0x00000000
0x40028860	SEC0_SCTL12	SEC0 Source Control Register n	0x00000000

Table A-51: ADSP-CM40x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40028864	SEC0_SSTAT12	SEC0 Source Status Register n	0x00000000
0x40028868	SEC0_SCTL13	SEC0 Source Control Register n	0x00000000
0x4002886C	SEC0_SSTAT13	SEC0 Source Status Register n	0x00000000
0x40028870	SEC0_SCTL14	SEC0 Source Control Register n	0x00000000
0x40028874	SEC0_SSTAT14	SEC0 Source Status Register n	0x00000000
0x40028878	SEC0_SCTL15	SEC0 Source Control Register n	0x00000000
0x4002887C	SEC0_SSTAT15	SEC0 Source Status Register n	0x00000000
0x40028880	SEC0_SCTL16	SEC0 Source Control Register n	0x00000000
0x40028884	SEC0_SSTAT16	SEC0 Source Status Register n	0x00000000
0x40028888	SEC0_SCTL17	SEC0 Source Control Register n	0x00000000
0x4002888C	SEC0_SSTAT17	SEC0 Source Status Register n	0x00000000
0x40028890	SEC0_SCTL18	SEC0 Source Control Register n	0x00000000
0x40028894	SEC0_SSTAT18	SEC0 Source Status Register n	0x00000000
0x40028898	SEC0_SCTL19	SEC0 Source Control Register n	0x00000000
0x4002889C	SEC0_SSTAT19	SEC0 Source Status Register n	0x00000000
0x400288A0	SEC0_SCTL20	SEC0 Source Control Register n	0x00000000
0x400288A4	SEC0_SSTAT20	SEC0 Source Status Register n	0x00000000
0x400288A8	SEC0_SCTL21	SEC0 Source Control Register n	0x00000000
0x400288AC	SEC0_SSTAT21	SEC0 Source Status Register n	0x00000000
0x400288B0	SEC0_SCTL22	SEC0 Source Control Register n	0x00000000
0x400288B4	SEC0_SSTAT22	SEC0 Source Status Register n	0x00000000
0x400288B8	SEC0_SCTL23	SEC0 Source Control Register n	0x00000000
0x400288BC	SEC0_SSTAT23	SEC0 Source Status Register n	0x00000000
0x400288C0	SEC0_SCTL24	SEC0 Source Control Register n	0x00000000
0x400288C4	SEC0_SSTAT24	SEC0 Source Status Register n	0x00000000
0x400288C8	SEC0_SCTL25	SEC0 Source Control Register n	0x00000000
0x400288CC	SEC0_SSTAT25	SEC0 Source Status Register n	0x00000000
0x400288D0	SEC0_SCTL26	SEC0 Source Control Register n	0x00000000
0x400288D4	SEC0_SSTAT26	SEC0 Source Status Register n	0x00000000
0x400288D8	SEC0_SCTL27	SEC0 Source Control Register n	0x00000000
0x400288DC	SEC0_SSTAT27	SEC0 Source Status Register n	0x00000000

Table A-51: ADSP-CM40x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x400288E0	SEC0_SCTL28	SEC0 Source Control Register n	0x00000000
0x400288E4	SEC0_SSTAT28	SEC0 Source Status Register n	0x00000000
0x400288E8	SEC0_SCTL29	SEC0 Source Control Register n	0x00000000
0x400288EC	SEC0_SSTAT29	SEC0 Source Status Register n	0x00000000
0x400288F0	SEC0_SCTL30	SEC0 Source Control Register n	0x00000000
0x400288F4	SEC0_SSTAT30	SEC0 Source Status Register n	0x00000000
0x400288F8	SEC0_SCTL31	SEC0 Source Control Register n	0x00000000
0x400288FC	SEC0_SSTAT31	SEC0 Source Status Register n	0x00000000
0x40028900	SEC0_SCTL32	SEC0 Source Control Register n	0x00000000
0x40028904	SEC0_SSTAT32	SEC0 Source Status Register n	0x00000000
0x40028908	SEC0_SCTL33	SEC0 Source Control Register n	0x00000000
0x4002890C	SEC0_SSTAT33	SEC0 Source Status Register n	0x00000000
0x40028910	SEC0_SCTL34	SEC0 Source Control Register n	0x00000000
0x40028914	SEC0_SSTAT34	SEC0 Source Status Register n	0x00000000
0x40028918	SEC0_SCTL35	SEC0 Source Control Register n	0x00000000
0x4002891C	SEC0_SSTAT35	SEC0 Source Status Register n	0x00000000
0x40028920	SEC0_SCTL36	SEC0 Source Control Register n	0x00000000
0x40028924	SEC0_SSTAT36	SEC0 Source Status Register n	0x00000000
0x40028928	SEC0_SCTL37	SEC0 Source Control Register n	0x00000000
0x4002892C	SEC0_SSTAT37	SEC0 Source Status Register n	0x00000000
0x40028930	SEC0_SCTL38	SEC0 Source Control Register n	0x00000000
0x40028934	SEC0_SSTAT38	SEC0 Source Status Register n	0x00000000
0x40028938	SEC0_SCTL39	SEC0 Source Control Register n	0x00000000
0x4002893C	SEC0_SSTAT39	SEC0 Source Status Register n	0x00000000
0x40028940	SEC0_SCTL40	SEC0 Source Control Register n	0x00000000
0x40028944	SEC0_SSTAT40	SEC0 Source Status Register n	0x00000000
0x40028948	SEC0_SCTL41	SEC0 Source Control Register n	0x00000000
0x4002894C	SEC0_SSTAT41	SEC0 Source Status Register n	0x00000000
0x40028950	SEC0_SCTL42	SEC0 Source Control Register n	0x00000000
0x40028954	SEC0_SSTAT42	SEC0 Source Status Register n	0x00000000
0x40028958	SEC0_SCTL43	SEC0 Source Control Register n	0x00000000

Table A-51: ADSP-CM40x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4002895C	SEC0_SSTAT43	SEC0 Source Status Register n	0x00000000
0x40028960	SEC0_SCTL44	SEC0 Source Control Register n	0x00000000
0x40028964	SEC0_SSTAT44	SEC0 Source Status Register n	0x00000000
0x40028968	SEC0_SCTL45	SEC0 Source Control Register n	0x00000000
0x4002896C	SEC0_SSTAT45	SEC0 Source Status Register n	0x00000000
0x40028970	SEC0_SCTL46	SEC0 Source Control Register n	0x00000000
0x40028974	SEC0_SSTAT46	SEC0 Source Status Register n	0x00000000
0x40028978	SEC0_SCTL47	SEC0 Source Control Register n	0x00000000
0x4002897C	SEC0_SSTAT47	SEC0 Source Status Register n	0x00000000
0x40028980	SEC0_SCTL48	SEC0 Source Control Register n	0x00000000
0x40028984	SEC0_SSTAT48	SEC0 Source Status Register n	0x00000000
0x40028988	SEC0_SCTL49	SEC0 Source Control Register n	0x00000000
0x4002898C	SEC0_SSTAT49	SEC0 Source Status Register n	0x00000000
0x40028990	SEC0_SCTL50	SEC0 Source Control Register n	0x00000000
0x40028994	SEC0_SSTAT50	SEC0 Source Status Register n	0x00000000
0x40028998	SEC0_SCTL51	SEC0 Source Control Register n	0x00000000
0x4002899C	SEC0_SSTAT51	SEC0 Source Status Register n	0x00000000
0x400289A0	SEC0_SCTL52	SEC0 Source Control Register n	0x00000000
0x400289A4	SEC0_SSTAT52	SEC0 Source Status Register n	0x00000000
0x400289A8	SEC0_SCTL53	SEC0 Source Control Register n	0x00000000
0x400289AC	SEC0_SSTAT53	SEC0 Source Status Register n	0x00000000
0x400289B0	SEC0_SCTL54	SEC0 Source Control Register n	0x00000000
0x400289B4	SEC0_SSTAT54	SEC0 Source Status Register n	0x00000000
0x400289B8	SEC0_SCTL55	SEC0 Source Control Register n	0x00000000
0x400289BC	SEC0_SSTAT55	SEC0 Source Status Register n	0x00000000
0x400289C0	SEC0_SCTL56	SEC0 Source Control Register n	0x00000000
0x400289C4	SEC0_SSTAT56	SEC0 Source Status Register n	0x00000000
0x400289C8	SEC0_SCTL57	SEC0 Source Control Register n	0x00000000
0x400289CC	SEC0_SSTAT57	SEC0 Source Status Register n	0x00000000
0x400289D0	SEC0_SCTL58	SEC0 Source Control Register n	0x00000000
0x400289D4	SEC0_SSTAT58	SEC0 Source Status Register n	0x00000000

Table A-51: ADSP-CM40x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x400289D8	SEC0_SCTL59	SEC0 Source Control Register n	0x00000000
0x400289DC	SEC0_SSTAT59	SEC0 Source Status Register n	0x00000000
0x400289E0	SEC0_SCTL60	SEC0 Source Control Register n	0x00000000
0x400289E4	SEC0_SSTAT60	SEC0 Source Status Register n	0x00000000
0x400289E8	SEC0_SCTL61	SEC0 Source Control Register n	0x00000000
0x400289EC	SEC0_SSTAT61	SEC0 Source Status Register n	0x00000000
0x400289F0	SEC0_SCTL62	SEC0 Source Control Register n	0x00000000
0x400289F4	SEC0_SSTAT62	SEC0 Source Status Register n	0x00000000
0x400289F8	SEC0_SCTL63	SEC0 Source Control Register n	0x00000000
0x400289FC	SEC0_SSTAT63	SEC0 Source Status Register n	0x00000000
0x40028A00	SEC0_SCTL64	SEC0 Source Control Register n	0x00000000
0x40028A04	SEC0_SSTAT64	SEC0 Source Status Register n	0x00000000
0x40028A08	SEC0_SCTL65	SEC0 Source Control Register n	0x00000000
0x40028A0C	SEC0_SSTAT65	SEC0 Source Status Register n	0x00000000
0x40028A10	SEC0_SCTL66	SEC0 Source Control Register n	0x00000000
0x40028A14	SEC0_SSTAT66	SEC0 Source Status Register n	0x00000000
0x40028A18	SEC0_SCTL67	SEC0 Source Control Register n	0x00000000
0x40028A1C	SEC0_SSTAT67	SEC0 Source Status Register n	0x00000000
0x40028A20	SEC0_SCTL68	SEC0 Source Control Register n	0x00000000
0x40028A24	SEC0_SSTAT68	SEC0 Source Status Register n	0x00000000
0x40028A28	SEC0_SCTL69	SEC0 Source Control Register n	0x00000000
0x40028A2C	SEC0_SSTAT69	SEC0 Source Status Register n	0x00000000
0x40028A30	SEC0_SCTL70	SEC0 Source Control Register n	0x00000000
0x40028A34	SEC0_SSTAT70	SEC0 Source Status Register n	0x00000000
0x40028A38	SEC0_SCTL71	SEC0 Source Control Register n	0x00000000
0x40028A3C	SEC0_SSTAT71	SEC0 Source Status Register n	0x00000000
0x40028A40	SEC0_SCTL72	SEC0 Source Control Register n	0x00000000
0x40028A44	SEC0_SSTAT72	SEC0 Source Status Register n	0x00000000
0x40028A48	SEC0_SCTL73	SEC0 Source Control Register n	0x00000000
0x40028A4C	SEC0_SSTAT73	SEC0 Source Status Register n	0x00000000
0x40028A50	SEC0_SCTL74	SEC0 Source Control Register n	0x00000000

Table A-51: ADSP-CM40x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40028A54	SEC0_SSTAT74	SEC0 Source Status Register n	0x00000000
0x40028A58	SEC0_SCTL75	SEC0 Source Control Register n	0x00000000
0x40028A5C	SEC0_SSTAT75	SEC0 Source Status Register n	0x00000000
0x40028A60	SEC0_SCTL76	SEC0 Source Control Register n	0x00000000
0x40028A64	SEC0_SSTAT76	SEC0 Source Status Register n	0x00000000
0x40028A68	SEC0_SCTL77	SEC0 Source Control Register n	0x00000000
0x40028A6C	SEC0_SSTAT77	SEC0 Source Status Register n	0x00000000
0x40028A70	SEC0_SCTL78	SEC0 Source Control Register n	0x00000000
0x40028A74	SEC0_SSTAT78	SEC0 Source Status Register n	0x00000000
0x40028A78	SEC0_SCTL79	SEC0 Source Control Register n	0x00000000
0x40028A7C	SEC0_SSTAT79	SEC0 Source Status Register n	0x00000000
0x40028A80	SEC0_SCTL80	SEC0 Source Control Register n	0x00000000
0x40028A84	SEC0_SSTAT80	SEC0 Source Status Register n	0x00000000
0x40028A88	SEC0_SCTL81	SEC0 Source Control Register n	0x00000000
0x40028A8C	SEC0_SSTAT81	SEC0 Source Status Register n	0x00000000
0x40028A90	SEC0_SCTL82	SEC0 Source Control Register n	0x00000000
0x40028A94	SEC0_SSTAT82	SEC0 Source Status Register n	0x00000000
0x40028A98	SEC0_SCTL83	SEC0 Source Control Register n	0x00000000
0x40028A9C	SEC0_SSTAT83	SEC0 Source Status Register n	0x00000000
0x40028AA0	SEC0_SCTL84	SEC0 Source Control Register n	0x00000000
0x40028AA4	SEC0_SSTAT84	SEC0 Source Status Register n	0x00000000
0x40028AA8	SEC0_SCTL85	SEC0 Source Control Register n	0x00000000
0x40028AAC	SEC0_SSTAT85	SEC0 Source Status Register n	0x00000000
0x40028AB0	SEC0_SCTL86	SEC0 Source Control Register n	0x00000000
0x40028AB4	SEC0_SSTAT86	SEC0 Source Status Register n	0x00000000
0x40028AB8	SEC0_SCTL87	SEC0 Source Control Register n	0x00000000
0x40028ABC	SEC0_SSTAT87	SEC0 Source Status Register n	0x00000000
0x40028AC0	SEC0_SCTL88	SEC0 Source Control Register n	0x00000000
0x40028AC4	SEC0_SSTAT88	SEC0 Source Status Register n	0x00000000
0x40028AC8	SEC0_SCTL89	SEC0 Source Control Register n	0x00000000
0x40028ACC	SEC0_SSTAT89	SEC0 Source Status Register n	0x00000000

Table A-51: ADSP-CM40x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40028AD0	SEC0_SCTL90	SEC0 Source Control Register n	0x00000000
0x40028AD4	SEC0_SSTAT90	SEC0 Source Status Register n	0x00000000
0x40028AD8	SEC0_SCTL91	SEC0 Source Control Register n	0x00000000
0x40028ADC	SEC0_SSTAT91	SEC0 Source Status Register n	0x00000000
0x40028AE0	SEC0_SCTL92	SEC0 Source Control Register n	0x00000000
0x40028AE4	SEC0_SSTAT92	SEC0 Source Status Register n	0x00000000
0x40028AE8	SEC0_SCTL93	SEC0 Source Control Register n	0x00000000
0x40028AEC	SEC0_SSTAT93	SEC0 Source Status Register n	0x00000000
0x40028AF0	SEC0_SCTL94	SEC0 Source Control Register n	0x00000000
0x40028AF4	SEC0_SSTAT94	SEC0 Source Status Register n	0x00000000
0x40028AF8	SEC0_SCTL95	SEC0 Source Control Register n	0x00000000
0x40028AFC	SEC0_SSTAT95	SEC0 Source Status Register n	0x00000000
0x40028B00	SEC0_SCTL96	SEC0 Source Control Register n	0x00000000
0x40028B04	SEC0_SSTAT96	SEC0 Source Status Register n	0x00000000
0x40028B08	SEC0_SCTL97	SEC0 Source Control Register n	0x00000000
0x40028B0C	SEC0_SSTAT97	SEC0 Source Status Register n	0x00000000
0x40028B10	SEC0_SCTL98	SEC0 Source Control Register n	0x00000000
0x40028B14	SEC0_SSTAT98	SEC0 Source Status Register n	0x00000000
0x40028B18	SEC0_SCTL99	SEC0 Source Control Register n	0x00000000
0x40028B1C	SEC0_SSTAT99	SEC0 Source Status Register n	0x00000000
0x40028B20	SEC0_SCTL100	SEC0 Source Control Register n	0x00000000
0x40028B24	SEC0_SSTAT100	SEC0 Source Status Register n	0x00000000
0x40028B28	SEC0_SCTL101	SEC0 Source Control Register n	0x00000000
0x40028B2C	SEC0_SSTAT101	SEC0 Source Status Register n	0x00000000
0x40028B30	SEC0_SCTL102	SEC0 Source Control Register n	0x00000000
0x40028B34	SEC0_SSTAT102	SEC0 Source Status Register n	0x00000000
0x40028B38	SEC0_SCTL103	SEC0 Source Control Register n	0x00000000
0x40028B3C	SEC0_SSTAT103	SEC0 Source Status Register n	0x00000000
0x40028B40	SEC0_SCTL104	SEC0 Source Control Register n	0x00000000
0x40028B44	SEC0_SSTAT104	SEC0 Source Status Register n	0x00000000
0x40028B48	SEC0_SCTL105	SEC0 Source Control Register n	0x00000000

Table A-51: ADSP-CM40x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40028B4C	SEC0_SSTAT105	SEC0 Source Status Register n	0x00000000
0x40028B50	SEC0_SCTL106	SEC0 Source Control Register n	0x00000000
0x40028B54	SEC0_SSTAT106	SEC0 Source Status Register n	0x00000000
0x40028B58	SEC0_SCTL107	SEC0 Source Control Register n	0x00000000
0x40028B5C	SEC0_SSTAT107	SEC0 Source Status Register n	0x00000000
0x40028B60	SEC0_SCTL108	SEC0 Source Control Register n	0x00000000
0x40028B64	SEC0_SSTAT108	SEC0 Source Status Register n	0x00000000
0x40028B68	SEC0_SCTL109	SEC0 Source Control Register n	0x00000000
0x40028B6C	SEC0_SSTAT109	SEC0 Source Status Register n	0x00000000
0x40028B70	SEC0_SCTL110	SEC0 Source Control Register n	0x00000000
0x40028B74	SEC0_SSTAT110	SEC0 Source Status Register n	0x00000000
0x40028B78	SEC0_SCTL111	SEC0 Source Control Register n	0x00000000
0x40028B7C	SEC0_SSTAT111	SEC0 Source Status Register n	0x00000000
0x40028B80	SEC0_SCTL112	SEC0 Source Control Register n	0x00000000
0x40028B84	SEC0_SSTAT112	SEC0 Source Status Register n	0x00000000
0x40028B88	SEC0_SCTL113	SEC0 Source Control Register n	0x00000000
0x40028B8C	SEC0_SSTAT113	SEC0 Source Status Register n	0x00000000
0x40028B90	SEC0_SCTL114	SEC0 Source Control Register n	0x00000000
0x40028B94	SEC0_SSTAT114	SEC0 Source Status Register n	0x00000000
0x40028B98	SEC0_SCTL115	SEC0 Source Control Register n	0x00000000
0x40028B9C	SEC0_SSTAT115	SEC0 Source Status Register n	0x00000000
0x40028BA0	SEC0_SCTL116	SEC0 Source Control Register n	0x00000000
0x40028BA4	SEC0_SSTAT116	SEC0 Source Status Register n	0x00000000
0x40028BA8	SEC0_SCTL117	SEC0 Source Control Register n	0x00000000
0x40028BAC	SEC0_SSTAT117	SEC0 Source Status Register n	0x00000000
0x40028BB0	SEC0_SCTL118	SEC0 Source Control Register n	0x00000000
0x40028BB4	SEC0_SSTAT118	SEC0 Source Status Register n	0x00000000
0x40028BB8	SEC0_SCTL119	SEC0 Source Control Register n	0x00000000
0x40028BBC	SEC0_SSTAT119	SEC0 Source Status Register n	0x00000000
0x40028BC0	SEC0_SCTL120	SEC0 Source Control Register n	0x00000000
0x40028BC4	SEC0_SSTAT120	SEC0 Source Status Register n	0x00000000

Table A-51: ADSP-CM40x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40028BC8	SEC0_SCTL121	SEC0 Source Control Register n	0x00000000
0x40028BCC	SEC0_SSTAT121	SEC0 Source Status Register n	0x00000000
0x40028BD0	SEC0_SCTL122	SEC0 Source Control Register n	0x00000000
0x40028BD4	SEC0_SSTAT122	SEC0 Source Status Register n	0x00000000
0x40028BD8	SEC0_SCTL123	SEC0 Source Control Register n	0x00000000
0x40028BDC	SEC0_SSTAT123	SEC0 Source Status Register n	0x00000000
0x40028BE0	SEC0_SCTL124	SEC0 Source Control Register n	0x00000000
0x40028BE4	SEC0_SSTAT124	SEC0 Source Status Register n	0x00000000
0x40028BE8	SEC0_SCTL125	SEC0 Source Control Register n	0x00000000
0x40028BEC	SEC0_SSTAT125	SEC0 Source Status Register n	0x00000000
0x40028BF0	SEC0_SCTL126	SEC0 Source Control Register n	0x00000000
0x40028BF4	SEC0_SSTAT126	SEC0 Source Status Register n	0x00000000
0x40028BF8	SEC0_SCTL127	SEC0 Source Control Register n	0x00000000
0x40028BFC	SEC0_SSTAT127	SEC0 Source Status Register n	0x00000000
0x40028C00	SEC0_SCTL128	SEC0 Source Control Register n	0x00000000
0x40028C04	SEC0_SSTAT128	SEC0 Source Status Register n	0x00000000

Table A-52: ADSP-CM40x SINC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x4000D000	SINC0_CTL	SINC0 Control Register	0x00000000
0x4000D004	SINC0_STAT	SINC0 Status Register	0x00000000
0x4000D008	SINC0_CLK	SINC0 Clock Control Register	0x10001000
0x4000D010	SINC0_RATE0	SINC0 Rate Control for Group 0 Register	0x00000804
0x4000D014	SINC0_RATE1	SINC0 Rate Control for Group 1 Register	0x00000804
0x4000D018	SINC0_LEVEL0	SINC0 Level Control for Group 0 Register	0x04000000
0x4000D01C	SINC0_LEVEL1	SINC0 Level Control for Group 1 Register	0x04000000
0x4000D020	SINC0_LIMIT0	SINC0 (Amplitude) Limits for Secondary Filter 0 Register	0x00000000
0x4000D024	SINC0_LIMIT1	SINC0 (Amplitude) Limits for Secondary Filter 1 Register	0x00000000
0x4000D028	SINC0_LIMIT2	SINC0 (Amplitude) Limits for Secondary Filter 2 Register	0x00000000
0x4000D02C	SINC0_LIMIT3	SINC0 (Amplitude) Limits for Secondary Filter 3 Register	0x00000000

Table A-52: ADSP-CM40x SINC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4000D030	SINC0_BIAS0	SINC0 Bias for Group 0 Register	0x00000000
0x4000D034	SINC0_BIAS1	SINC0 Bias for Group 1 Register	0x00000000
0x4000D038	SINC0_PPTR0	SINC0 Primary (Filters) Pointer for Group 0 Register	0x00000000
0x4000D03C	SINC0_PPTR1	SINC0 Primary (Filters) Pointer for Group 1 Register	0x00000000
0x4000D040	SINC0_PHEAD0	SINC0 Primary (Filters) Head for Group 0 Register	0x00000000
0x4000D044	SINC0_PHEAD1	SINC0 Primary (Filters) Head for Group 1 Register	0x00000000
0x4000D048	SINC0_PTAIL0	SINC0 Primary (Filters) Tail for Group 0 Register	0x00000000
0x4000D04C	SINC0_PTAIL1	SINC0 Primary (Filters) Tail for Group 1 Register	0x00000000
0x4000D050	SINC0_HIS_STAT	SINC0 History Status Register	0x00000000
0x4000D080	SINC0_P0SEC_HIST0	SINC0 Pair 0 Secondary (Filter) History n Register	0x00000000
0x4000D084	SINC0_P0SEC_HIST1	SINC0 Pair 0 Secondary (Filter) History n Register	0x00000000
0x4000D088	SINC0_P0SEC_HIST2	SINC0 Pair 0 Secondary (Filter) History n Register	0x00000000
0x4000D08C	SINC0_P0SEC_HIST3	SINC0 Pair 0 Secondary (Filter) History n Register	0x00000000
0x4000D090	SINC0_P1SEC_HIST0	SINC0 Pair 1 Secondary (Filter) History n Register	0x00000000
0x4000D094	SINC0_P1SEC_HIST1	SINC0 Pair 1 Secondary (Filter) History n Register	0x00000000
0x4000D098	SINC0_P1SEC_HIST2	SINC0 Pair 1 Secondary (Filter) History n Register	0x00000000
0x4000D09C	SINC0_P1SEC_HIST3	SINC0 Pair 1 Secondary (Filter) History n Register	0x00000000
0x4000D0A0	SINC0_P2SEC_HIST0	SINC0 Pair 2 Secondary (Filter) History n Register	0x00000000
0x4000D0A4	SINC0_P2SEC_HIST1	SINC0 Pair 2 Secondary (Filter) History n Register	0x00000000
0x4000D0A8	SINC0_P2SEC_HIST2	SINC0 Pair 2 Secondary (Filter) History n Register	0x00000000
0x4000D0AC	SINC0_P2SEC_HIST3	SINC0 Pair 2 Secondary (Filter) History n Register	0x00000000
0x4000D0B0	SINC0_P3SEC_HIST0	SINC0 Pair 3 Secondary (Filter) History n Register	0x00000000
0x4000D0B4	SINC0_P3SEC_HIST1	SINC0 Pair 3 Secondary (Filter) History n Register	0x00000000
0x4000D0B8	SINC0_P3SEC_HIST2	SINC0 Pair 3 Secondary (Filter) History n Register	0x00000000
0x4000D0BC	SINC0_P3SEC_HIST3	SINC0 Pair 3 Secondary (Filter) History n Register	0x00000000

Table A-53: ADSP-CM40x SMC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x4000700C	SMC0_B0CTL	SMC0 Bank 0 Control Register	0x01000000
0x40007010	SMC0_B0TIM	SMC0 Bank 0 Timing Register	0x01010101

Table A-53: ADSP-CM40x SMC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40007014	SMC0_B0ETIM	SMC0 Bank 0 Extended Timing Register	0x00020200
0x4000701C	SMC0_B1CTL	SMC0 Bank 1 Control Register	0x00000000
0x40007020	SMC0_B1TIM	SMC0 Bank 1 Timing Register	0x01010101
0x40007024	SMC0_B1ETIM	SMC0 Bank 1 Extended Timing Register	0x00020200
0x4000702C	SMC0_B2CTL	SMC0 Bank 2 Control Register	0x00000000
0x40007030	SMC0_B2TIM	SMC0 Bank 2 Timing Register	0x01010101
0x40007034	SMC0_B2ETIM	SMC0 Bank 2 Extended Timing Register	0x00020200
0x4000703C	SMC0_B3CTL	SMC0 Bank 3 Control Register	0x00000000
0x40007040	SMC0_B3TIM	SMC0 Bank 3 Timing Register	0x01010101
0x40007044	SMC0_B3ETIM	SMC0 Bank 3 Extended Timing Register	0x00020200

Table A-54: ADSP-CM40x SPI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x4002A004	SPI0_CTL	SPI0 Control Register	0x00000050
0x4002A008	SPI0_RXCTL	SPI0 Receive Control Register	0x00000000
0x4002A00C	SPI0_TXCTL	SPI0 Transmit Control Register	0x00000000
0x4002A010	SPI0_CLK	SPI0 Clock Rate Register	0x00000000
0x4002A014	SPI0_DLY	SPI0 Delay Register	0x00000301
0x4002A018	SPI0_SLVSEL	SPI0 Slave Select Register	0x0000FE00
0x4002A01C	SPI0_RWC	SPI0 Received Word Count Register	0x00000000
0x4002A020	SPI0_RWCR	SPI0 Received Word Count Reload Register	0x00000000
0x4002A024	SPI0_TWC	SPI0 Transmitted Word Count Register	0x00000000
0x4002A028	SPI0_TWCR	SPI0 Transmitted Word Count Reload Register	0x00000000
0x4002A030	SPI0_IMSK	SPI0 Interrupt Mask Register	0x00000000
0x4002A034	SPI0_IMSK_CLR	SPI0 Interrupt Mask Clear Register	0x00000000
0x4002A038	SPI0_IMSK_SET	SPI0 Interrupt Mask Set Register	0x00000000
0x4002A040	SPI0_STAT	SPI0 Status Register	0x00440001
0x4002A044	SPI0_ILAT	SPI0 Masked Interrupt Condition Register	0x00000000
0x4002A048	SPI0_ILAT_CLR	SPI0 Masked Interrupt Clear Register	0x00000000
0x4002A050	SPI0_RFIFO	SPI0 Receive FIFO Data Register	0x00000000

Table A-54: ADSP-CM40x SPI0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4002A058	SPI0_TFIFO	SPI0 Transmit FIFO Data Register	0x00000000
0x4002A060	SPI0_MMRDH	SPI0 Memory Mapped Read Header	0x00000000
0x4002A064	SPI0_MMTOP	SPI0 SPI Memory Top Address	0x00000000

Table A-55: ADSP-CM40x SPI1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x4002B004	SPI1_CTL	SPI1 Control Register	0x00000050
0x4002B008	SPI1_RXCTL	SPI1 Receive Control Register	0x00000000
0x4002B00C	SPI1_TXCTL	SPI1 Transmit Control Register	0x00000000
0x4002B010	SPI1_CLK	SPI1 Clock Rate Register	0x00000000
0x4002B014	SPI1_DLY	SPI1 Delay Register	0x00000301
0x4002B018	SPI1_SLVSEL	SPI1 Slave Select Register	0x0000FE00
0x4002B01C	SPI1_RWC	SPI1 Received Word Count Register	0x00000000
0x4002B020	SPI1_RWCR	SPI1 Received Word Count Reload Register	0x00000000
0x4002B024	SPI1_TWC	SPI1 Transmitted Word Count Register	0x00000000
0x4002B028	SPI1_TWCR	SPI1 Transmitted Word Count Reload Register	0x00000000
0x4002B030	SPI1_IMSK	SPI1 Interrupt Mask Register	0x00000000
0x4002B034	SPI1_IMSK_CLR	SPI1 Interrupt Mask Clear Register	0x00000000
0x4002B038	SPI1_IMSK_SET	SPI1 Interrupt Mask Set Register	0x00000000
0x4002B040	SPI1_STAT	SPI1 Status Register	0x00440001
0x4002B044	SPI1_ILAT	SPI1 Masked Interrupt Condition Register	0x00000000
0x4002B048	SPI1_ILAT_CLR	SPI1 Masked Interrupt Clear Register	0x00000000
0x4002B050	SPI1_RFIFO	SPI1 Receive FIFO Data Register	0x00000000
0x4002B058	SPI1_TFIFO	SPI1 Transmit FIFO Data Register	0x00000000
0x4002B060	SPI1_MMRDH	SPI1 Memory Mapped Read Header	0x00000000
0x4002B064	SPI1_MMTOP	SPI1 SPI Memory Top Address	0x00000000

Table A-56: ADSP-CM40x SPI2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x4002C004	SPI2_CTL	SPI2 Control Register	0x00000050

Table A-56: ADSP-CM40x SPI2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4002C008	SPI2_RXCTL	SPI2 Receive Control Register	0x00000000
0x4002C00C	SPI2_TXCTL	SPI2 Transmit Control Register	0x00000000
0x4002C010	SPI2_CLK	SPI2 Clock Rate Register	0x00000000
0x4002C014	SPI2_DLY	SPI2 Delay Register	0x00000301
0x4002C018	SPI2_SLVSEL	SPI2 Slave Select Register	0x0000FE00
0x4002C01C	SPI2_RWC	SPI2 Received Word Count Register	0x00000000
0x4002C020	SPI2_RWCR	SPI2 Received Word Count Reload Register	0x00000000
0x4002C024	SPI2_TWC	SPI2 Transmitted Word Count Register	0x00000000
0x4002C028	SPI2_TWCR	SPI2 Transmitted Word Count Reload Register	0x00000000
0x4002C030	SPI2_IMSK	SPI2 Interrupt Mask Register	0x00000000
0x4002C034	SPI2_IMSK_CLR	SPI2 Interrupt Mask Clear Register	0x00000000
0x4002C038	SPI2_IMSK_SET	SPI2 Interrupt Mask Set Register	0x00000000
0x4002C040	SPI2_STAT	SPI2 Status Register	0x00440001
0x4002C044	SPI2_ILAT	SPI2 Masked Interrupt Condition Register	0x00000000
0x4002C048	SPI2_ILAT_CLR	SPI2 Masked Interrupt Clear Register	0x00000000
0x4002C050	SPI2_RFIFO	SPI2 Receive FIFO Data Register	0x00000000
0x4002C058	SPI2_TFIFO	SPI2 Transmit FIFO Data Register	0x00000000
0x4002C060	SPI2_MMRDH	SPI2 Memory Mapped Read Header	0x00000000
0x4002C064	SPI2_MMTOP	SPI2 SPI Memory Top Address	0x00000000

Table A-57: ADSP-CM40x SPORT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40001000	SPORT0_CTL_A	SPORT0 Half SPORT 'A' Control Register	0x00000000
0x40001004	SPORT0_DIV_A	SPORT0 Half SPORT 'A' Divisor Register	0x00000000
0x40001008	SPORT0_MCTL_A	SPORT0 Half SPORT 'A' Multi-channel Control Register	0x00000000
0x4000100C	SPORT0_CS0_A	SPORT0 Half SPORT 'A' Multi-channel 0-31 Select Register	0x00000000
0x40001010	SPORT0_CS1_A	SPORT0 Half SPORT 'A' Multi-channel 32-63 Select Register	0x00000000
0x40001014	SPORT0_CS2_A	SPORT0 Half SPORT 'A' Multi-channel 64-95 Select Register	0x00000000

Table A-57: ADSP-CM40x SPORT0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40001018	SPORT0_CS3_A	SPORT0 Half SPORT 'A' Multi-channel 96-127 Select Register	0x00000000
0x40001020	SPORT0_ERR_A	SPORT0 Half SPORT 'A' Error Register	0x00000000
0x40001024	SPORT0_MSTAT_A	SPORT0 Half SPORT 'A' Multi-channel Status Register	0x00000000
0x40001028	SPORT0_CTL2_A	SPORT0 Half SPORT 'A' Control 2 Register	0x00000000
0x40001040	SPORT0_TXPRI_A	SPORT0 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x40001044	SPORT0_RXPRI_A	SPORT0 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x40001048	SPORT0_TXSEC_A	SPORT0 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x4000104C	SPORT0_RXSEC_A	SPORT0 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x40001080	SPORT0_CTL_B	SPORT0 Half SPORT 'B' Control Register	0x00000000
0x40001084	SPORT0_DIV_B	SPORT0 Half SPORT 'B' Divisor Register	0x00000000
0x40001088	SPORT0_MCTL_B	SPORT0 Half SPORT 'B' Multi-channel Control Register	0x00000000
0x4000108C	SPORT0_CS0_B	SPORT0 Half SPORT 'B' Multi-channel 0-31 Select Register	0x00000000
0x40001090	SPORT0_CS1_B	SPORT0 Half SPORT 'B' Multi-channel 32-63 Select Register	0x00000000
0x40001094	SPORT0_CS2_B	SPORT0 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000
0x40001098	SPORT0_CS3_B	SPORT0 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x400010A0	SPORT0_ERR_B	SPORT0 Half SPORT 'B' Error Register	0x00000000
0x400010A4	SPORT0_MSTAT_B	SPORT0 Half SPORT 'B' Multi-channel Status Register	0x00000000
0x400010A8	SPORT0_CTL2_B	SPORT0 Half SPORT 'B' Control 2 Register	0x00000000
0x400010C0	SPORT0_TXPRI_B	SPORT0 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000
0x400010C4	SPORT0_RXPRI_B	SPORT0 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x400010C8	SPORT0_TXSEC_B	SPORT0 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x400010CC	SPORT0_RXSEC_B	SPORT0 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-58: ADSP-CM40x SPORT1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40001100	SPORT1_CTL_A	SPORT1 Half SPORT 'A' Control Register	0x00000000
0x40001104	SPORT1_DIV_A	SPORT1 Half SPORT 'A' Divisor Register	0x00000000

Table A-58: ADSP-CM40x SPORT1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40001108	SPORT1_MCTL_A	SPORT1 Half SPORT 'A' Multi-channel Control Register	0x00000000
0x4000110C	SPORT1_CS0_A	SPORT1 Half SPORT 'A' Multi-channel 0-31 Select Register	0x00000000
0x40001110	SPORT1_CS1_A	SPORT1 Half SPORT 'A' Multi-channel 32-63 Select Register	0x00000000
0x40001114	SPORT1_CS2_A	SPORT1 Half SPORT 'A' Multi-channel 64-95 Select Register	0x00000000
0x40001118	SPORT1_CS3_A	SPORT1 Half SPORT 'A' Multi-channel 96-127 Select Register	0x00000000
0x40001120	SPORT1_ERR_A	SPORT1 Half SPORT 'A' Error Register	0x00000000
0x40001124	SPORT1_MSTAT_A	SPORT1 Half SPORT 'A' Multi-channel Status Register	0x00000000
0x40001128	SPORT1_CTL2_A	SPORT1 Half SPORT 'A' Control 2 Register	0x00000000
0x40001140	SPORT1_TXPRI_A	SPORT1 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x40001144	SPORT1_RXPRI_A	SPORT1 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x40001148	SPORT1_TXSEC_A	SPORT1 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x4000114C	SPORT1_RXSEC_A	SPORT1 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x40001180	SPORT1_CTL_B	SPORT1 Half SPORT 'B' Control Register	0x00000000
0x40001184	SPORT1_DIV_B	SPORT1 Half SPORT 'B' Divisor Register	0x00000000
0x40001188	SPORT1_MCTL_B	SPORT1 Half SPORT 'B' Multi-channel Control Register	0x00000000
0x4000118C	SPORT1_CS0_B	SPORT1 Half SPORT 'B' Multi-channel 0-31 Select Register	0x00000000
0x40001190	SPORT1_CS1_B	SPORT1 Half SPORT 'B' Multi-channel 32-63 Select Register	0x00000000
0x40001194	SPORT1_CS2_B	SPORT1 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000
0x40001198	SPORT1_CS3_B	SPORT1 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x400011A0	SPORT1_ERR_B	SPORT1 Half SPORT 'B' Error Register	0x00000000
0x400011A4	SPORT1_MSTAT_B	SPORT1 Half SPORT 'B' Multi-channel Status Register	0x00000000
0x400011A8	SPORT1_CTL2_B	SPORT1 Half SPORT 'B' Control 2 Register	0x00000000
0x400011C0	SPORT1_TXPRI_B	SPORT1 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000
0x400011C4	SPORT1_RXPRI_B	SPORT1 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x400011C8	SPORT1_TXSEC_B	SPORT1 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000

Table A-58: ADSP-CM40x SPORT1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x400011CC	SPORT1_RXSEC_B	SPORT1 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-59: ADSP-CM40x SPU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40027000	SPU0_CTL	SPU0 Control Register	0x000000AD
0x40027004	SPU0_STAT	SPU0 Status Register	0x00000000
0x40027400	SPU0_WP0	SPU0 Write Protect Register n	0x00000000
0x40027404	SPU0_WP1	SPU0 Write Protect Register n	0x00000000
0x40027408	SPU0_WP2	SPU0 Write Protect Register n	0x00000000
0x4002740C	SPU0_WP3	SPU0 Write Protect Register n	0x00000000
0x40027410	SPU0_WP4	SPU0 Write Protect Register n	0x00000000
0x40027414	SPU0_WP5	SPU0 Write Protect Register n	0x00000000
0x40027418	SPU0_WP6	SPU0 Write Protect Register n	0x00000000
0x4002741C	SPU0_WP7	SPU0 Write Protect Register n	0x00000000
0x40027420	SPU0_WP8	SPU0 Write Protect Register n	0x00000000
0x40027424	SPU0_WP9	SPU0 Write Protect Register n	0x00000000
0x40027428	SPU0_WP10	SPU0 Write Protect Register n	0x00000000
0x4002742C	SPU0_WP11	SPU0 Write Protect Register n	0x00000000
0x40027430	SPU0_WP12	SPU0 Write Protect Register n	0x00000000
0x40027434	SPU0_WP13	SPU0 Write Protect Register n	0x00000000
0x40027438	SPU0_WP14	SPU0 Write Protect Register n	0x00000000
0x4002743C	SPU0_WP15	SPU0 Write Protect Register n	0x00000000
0x40027440	SPU0_WP16	SPU0 Write Protect Register n	0x00000000
0x40027444	SPU0_WP17	SPU0 Write Protect Register n	0x00000000
0x40027448	SPU0_WP18	SPU0 Write Protect Register n	0x00000000
0x4002744C	SPU0_WP19	SPU0 Write Protect Register n	0x00000000
0x40027450	SPU0_WP20	SPU0 Write Protect Register n	0x00000000
0x40027454	SPU0_WP21	SPU0 Write Protect Register n	0x00000000
0x40027458	SPU0_WP22	SPU0 Write Protect Register n	0x00000000
0x4002745C	SPU0_WP23	SPU0 Write Protect Register n	0x00000000

Table A-59: ADSP-CM40x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40027460	SPU0_WP24	SPU0 Write Protect Register n	0x00000000
0x40027464	SPU0_WP25	SPU0 Write Protect Register n	0x00000000
0x40027468	SPU0_WP26	SPU0 Write Protect Register n	0x00000000
0x4002746C	SPU0_WP27	SPU0 Write Protect Register n	0x00000000
0x40027470	SPU0_WP28	SPU0 Write Protect Register n	0x00000000
0x40027474	SPU0_WP29	SPU0 Write Protect Register n	0x00000000
0x40027478	SPU0_WP30	SPU0 Write Protect Register n	0x00000000
0x4002747C	SPU0_WP31	SPU0 Write Protect Register n	0x00000000
0x40027480	SPU0_WP32	SPU0 Write Protect Register n	0x00000000
0x40027484	SPU0_WP33	SPU0 Write Protect Register n	0x00000000
0x40027488	SPU0_WP34	SPU0 Write Protect Register n	0x00000000
0x4002748C	SPU0_WP35	SPU0 Write Protect Register n	0x00000000
0x40027490	SPU0_WP36	SPU0 Write Protect Register n	0x00000000
0x40027494	SPU0_WP37	SPU0 Write Protect Register n	0x00000000
0x40027498	SPU0_WP38	SPU0 Write Protect Register n	0x00000000
0x4002749C	SPU0_WP39	SPU0 Write Protect Register n	0x00000000
0x400274A0	SPU0_WP40	SPU0 Write Protect Register n	0x00000000
0x400274A4	SPU0_WP41	SPU0 Write Protect Register n	0x00000000
0x400274A8	SPU0_WP42	SPU0 Write Protect Register n	0x00000000
0x400274AC	SPU0_WP43	SPU0 Write Protect Register n	0x00000000
0x400274B0	SPU0_WP44	SPU0 Write Protect Register n	0x00000000
0x400274B4	SPU0_WP45	SPU0 Write Protect Register n	0x00000000
0x400274B8	SPU0_WP46	SPU0 Write Protect Register n	0x00000000
0x400274BC	SPU0_WP47	SPU0 Write Protect Register n	0x00000000
0x400274C0	SPU0_WP48	SPU0 Write Protect Register n	0x00000000
0x400274C4	SPU0_WP49	SPU0 Write Protect Register n	0x00000000
0x400274C8	SPU0_WP50	SPU0 Write Protect Register n	0x00000000
0x400274CC	SPU0_WP51	SPU0 Write Protect Register n	0x00000000
0x400274D0	SPU0_WP52	SPU0 Write Protect Register n	0x00000000
0x400274D4	SPU0_WP53	SPU0 Write Protect Register n	0x00000000
0x400274D8	SPU0_WP54	SPU0 Write Protect Register n	0x00000000

Table A-59: ADSP-CM40x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x400274DC	SPU0_WP55	SPU0 Write Protect Register n	0x00000000
0x400274E0	SPU0_WP56	SPU0 Write Protect Register n	0x00000000
0x400274E4	SPU0_WP57	SPU0 Write Protect Register n	0x00000000
0x400274E8	SPU0_WP58	SPU0 Write Protect Register n	0x00000000
0x400274EC	SPU0_WP59	SPU0 Write Protect Register n	0x00000000

Table A-60: ADSP-CM40x SWU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40022000	SWU0_GCTL	SWU0 Global Control Register	0x00000000
0x40022004	SWU0_GSTAT	SWU0 Global Status Register	0x00000000
0x40022010	SWU0_CTL0	SWU0 Control Register n	0x00000000
0x40022014	SWU0_LA0	SWU0 Lower Address Register n	0x00000000
0x40022018	SWU0_UA0	SWU0 Upper Address Register n	0x00000000
0x4002201C	SWU0_ID0	SWU0 ID Register n	0x00000000
0x40022020	SWU0_CNT0	SWU0 Count Register n	0x00000000
0x40022024	SWU0_TARG0	SWU0 Target Register n	0x00000000
0x40022028	SWU0_HIST0	SWU0 Bandwidth History Register n	0x00000000
0x4002202C	SWU0_CUR0	SWU0 Current Register n	0x00000000
0x40022030	SWU0_CTL1	SWU0 Control Register n	0x00000000
0x40022034	SWU0_LA1	SWU0 Lower Address Register n	0x00000000
0x40022038	SWU0_UA1	SWU0 Upper Address Register n	0x00000000
0x4002203C	SWU0_ID1	SWU0 ID Register n	0x00000000
0x40022040	SWU0_CNT1	SWU0 Count Register n	0x00000000
0x40022044	SWU0_TARG1	SWU0 Target Register n	0x00000000
0x40022048	SWU0_HIST1	SWU0 Bandwidth History Register n	0x00000000
0x4002204C	SWU0_CUR1	SWU0 Current Register n	0x00000000
0x40022050	SWU0_CTL2	SWU0 Control Register n	0x00000000
0x40022054	SWU0_LA2	SWU0 Lower Address Register n	0x00000000
0x40022058	SWU0_UA2	SWU0 Upper Address Register n	0x00000000
0x4002205C	SWU0_ID2	SWU0 ID Register n	0x00000000

Table A-60: ADSP-CM40x SWU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40022060	SWU0_CNT2	SWU0 Count Register n	0x00000000
0x40022064	SWU0_TARG2	SWU0 Target Register n	0x00000000
0x40022068	SWU0_HIST2	SWU0 Bandwidth History Register n	0x00000000
0x4002206C	SWU0_CUR2	SWU0 Current Register n	0x00000000
0x40022070	SWU0_CTL3	SWU0 Control Register n	0x00000000
0x40022074	SWU0_LA3	SWU0 Lower Address Register n	0x00000000
0x40022078	SWU0_UA3	SWU0 Upper Address Register n	0x00000000
0x4002207C	SWU0_ID3	SWU0 ID Register n	0x00000000
0x40022080	SWU0_CNT3	SWU0 Count Register n	0x00000000
0x40022084	SWU0_TARG3	SWU0 Target Register n	0x00000000
0x40022088	SWU0_HIST3	SWU0 Bandwidth History Register n	0x00000000
0x4002208C	SWU0_CUR3	SWU0 Current Register n	0x00000000

Table A-61: ADSP-CM40x SWU1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40023000	SWU1_GCTL	SWU1 Global Control Register	0x00000000
0x40023004	SWU1_GSTAT	SWU1 Global Status Register	0x00000000
0x40023010	SWU1_CTL0	SWU1 Control Register n	0x00000000
0x40023014	SWU1_LA0	SWU1 Lower Address Register n	0x00000000
0x40023018	SWU1_UA0	SWU1 Upper Address Register n	0x00000000
0x4002301C	SWU1_ID0	SWU1 ID Register n	0x00000000
0x40023020	SWU1_CNT0	SWU1 Count Register n	0x00000000
0x40023024	SWU1_TARG0	SWU1 Target Register n	0x00000000
0x40023028	SWU1_HIST0	SWU1 Bandwidth History Register n	0x00000000
0x4002302C	SWU1_CUR0	SWU1 Current Register n	0x00000000
0x40023030	SWU1_CTL1	SWU1 Control Register n	0x00000000
0x40023034	SWU1_LA1	SWU1 Lower Address Register n	0x00000000
0x40023038	SWU1_UA1	SWU1 Upper Address Register n	0x00000000
0x4002303C	SWU1_ID1	SWU1 ID Register n	0x00000000
0x40023040	SWU1_CNT1	SWU1 Count Register n	0x00000000

Table A-61: ADSP-CM40x SWU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40023044	SWU1_TARG1	SWU1 Target Register n	0x00000000
0x40023048	SWU1_HIST1	SWU1 Bandwidth History Register n	0x00000000
0x4002304C	SWU1_CUR1	SWU1 Current Register n	0x00000000
0x40023050	SWU1_CTL2	SWU1 Control Register n	0x00000000
0x40023054	SWU1_LA2	SWU1 Lower Address Register n	0x00000000
0x40023058	SWU1_UA2	SWU1 Upper Address Register n	0x00000000
0x4002305C	SWU1_ID2	SWU1 ID Register n	0x00000000
0x40023060	SWU1_CNT2	SWU1 Count Register n	0x00000000
0x40023064	SWU1_TARG2	SWU1 Target Register n	0x00000000
0x40023068	SWU1_HIST2	SWU1 Bandwidth History Register n	0x00000000
0x4002306C	SWU1_CUR2	SWU1 Current Register n	0x00000000
0x40023070	SWU1_CTL3	SWU1 Control Register n	0x00000000
0x40023074	SWU1_LA3	SWU1 Lower Address Register n	0x00000000
0x40023078	SWU1_UA3	SWU1 Upper Address Register n	0x00000000
0x4002307C	SWU1_ID3	SWU1 ID Register n	0x00000000
0x40023080	SWU1_CNT3	SWU1 Count Register n	0x00000000
0x40023084	SWU1_TARG3	SWU1 Target Register n	0x00000000
0x40023088	SWU1_HIST3	SWU1 Bandwidth History Register n	0x00000000
0x4002308C	SWU1_CUR3	SWU1 Current Register n	0x00000000

Table A-62: ADSP-CM40x SWU2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40024000	SWU2_GCTL	SWU2 Global Control Register	0x00000000
0x40024004	SWU2_GSTAT	SWU2 Global Status Register	0x00000000
0x40024010	SWU2_CTL0	SWU2 Control Register n	0x00000000
0x40024014	SWU2_LA0	SWU2 Lower Address Register n	0x00000000
0x40024018	SWU2_UA0	SWU2 Upper Address Register n	0x00000000
0x4002401C	SWU2_ID0	SWU2 ID Register n	0x00000000
0x40024020	SWU2_CNT0	SWU2 Count Register n	0x00000000
0x40024024	SWU2_TARG0	SWU2 Target Register n	0x00000000

Table A-62: ADSP-CM40x SWU2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40024028	SWU2_HIST0	SWU2 Bandwidth History Register n	0x00000000
0x4002402C	SWU2_CUR0	SWU2 Current Register n	0x00000000
0x40024030	SWU2_CTL1	SWU2 Control Register n	0x00000000
0x40024034	SWU2_LA1	SWU2 Lower Address Register n	0x00000000
0x40024038	SWU2_UA1	SWU2 Upper Address Register n	0x00000000
0x4002403C	SWU2_ID1	SWU2 ID Register n	0x00000000
0x40024040	SWU2_CNT1	SWU2 Count Register n	0x00000000
0x40024044	SWU2_TARG1	SWU2 Target Register n	0x00000000
0x40024048	SWU2_HIST1	SWU2 Bandwidth History Register n	0x00000000
0x4002404C	SWU2_CUR1	SWU2 Current Register n	0x00000000
0x40024050	SWU2_CTL2	SWU2 Control Register n	0x00000000
0x40024054	SWU2_LA2	SWU2 Lower Address Register n	0x00000000
0x40024058	SWU2_UA2	SWU2 Upper Address Register n	0x00000000
0x4002405C	SWU2_ID2	SWU2 ID Register n	0x00000000
0x40024060	SWU2_CNT2	SWU2 Count Register n	0x00000000
0x40024064	SWU2_TARG2	SWU2 Target Register n	0x00000000
0x40024068	SWU2_HIST2	SWU2 Bandwidth History Register n	0x00000000
0x4002406C	SWU2_CUR2	SWU2 Current Register n	0x00000000
0x40024070	SWU2_CTL3	SWU2 Control Register n	0x00000000
0x40024074	SWU2_LA3	SWU2 Lower Address Register n	0x00000000
0x40024078	SWU2_UA3	SWU2 Upper Address Register n	0x00000000
0x4002407C	SWU2_ID3	SWU2 ID Register n	0x00000000
0x40024080	SWU2_CNT3	SWU2 Count Register n	0x00000000
0x40024084	SWU2_TARG3	SWU2 Target Register n	0x00000000
0x40024088	SWU2_HIST3	SWU2 Bandwidth History Register n	0x00000000
0x4002408C	SWU2_CUR3	SWU2 Current Register n	0x00000000

Table A-63: ADSP-CM40x SWU3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40025000	SWU3_GCTL	SWU3 Global Control Register	0x00000000

Table A-63: ADSP-CM40x SWU3 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40025004	SWU3_GSTAT	SWU3 Global Status Register	0x00000000
0x40025010	SWU3_CTL0	SWU3 Control Register n	0x00000000
0x40025014	SWU3_LA0	SWU3 Lower Address Register n	0x00000000
0x40025018	SWU3_UA0	SWU3 Upper Address Register n	0x00000000
0x4002501C	SWU3_ID0	SWU3 ID Register n	0x00000000
0x40025020	SWU3_CNT0	SWU3 Count Register n	0x00000000
0x40025024	SWU3_TARG0	SWU3 Target Register n	0x00000000
0x40025028	SWU3_HIST0	SWU3 Bandwidth History Register n	0x00000000
0x4002502C	SWU3_CUR0	SWU3 Current Register n	0x00000000
0x40025030	SWU3_CTL1	SWU3 Control Register n	0x00000000
0x40025034	SWU3_LA1	SWU3 Lower Address Register n	0x00000000
0x40025038	SWU3_UA1	SWU3 Upper Address Register n	0x00000000
0x4002503C	SWU3_ID1	SWU3 ID Register n	0x00000000
0x40025040	SWU3_CNT1	SWU3 Count Register n	0x00000000
0x40025044	SWU3_TARG1	SWU3 Target Register n	0x00000000
0x40025048	SWU3_HIST1	SWU3 Bandwidth History Register n	0x00000000
0x4002504C	SWU3_CUR1	SWU3 Current Register n	0x00000000
0x40025050	SWU3_CTL2	SWU3 Control Register n	0x00000000
0x40025054	SWU3_LA2	SWU3 Lower Address Register n	0x00000000
0x40025058	SWU3_UA2	SWU3 Upper Address Register n	0x00000000
0x4002505C	SWU3_ID2	SWU3 ID Register n	0x00000000
0x40025060	SWU3_CNT2	SWU3 Count Register n	0x00000000
0x40025064	SWU3_TARG2	SWU3 Target Register n	0x00000000
0x40025068	SWU3_HIST2	SWU3 Bandwidth History Register n	0x00000000
0x4002506C	SWU3_CUR2	SWU3 Current Register n	0x00000000
0x40025070	SWU3_CTL3	SWU3 Control Register n	0x00000000
0x40025074	SWU3_LA3	SWU3 Lower Address Register n	0x00000000
0x40025078	SWU3_UA3	SWU3 Upper Address Register n	0x00000000
0x4002507C	SWU3_ID3	SWU3 ID Register n	0x00000000
0x40025080	SWU3_CNT3	SWU3 Count Register n	0x00000000
0x40025084	SWU3_TARG3	SWU3 Target Register n	0x00000000

Table A-63: ADSP-CM40x SWU3 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40025088	SWU3_HIST3	SWU3 Bandwidth History Register n	0x00000000
0x4002508C	SWU3_CUR3	SWU3 Current Register n	0x00000000

Table A-64: ADSP-CM40x SWU4 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40026000	SWU4_GCTL	SWU4 Global Control Register	0x00000000
0x40026004	SWU4_GSTAT	SWU4 Global Status Register	0x00000000
0x40026010	SWU4_CTL0	SWU4 Control Register n	0x00000000
0x40026014	SWU4_LA0	SWU4 Lower Address Register n	0x00000000
0x40026018	SWU4_UA0	SWU4 Upper Address Register n	0x00000000
0x4002601C	SWU4_ID0	SWU4 ID Register n	0x00000000
0x40026020	SWU4_CNT0	SWU4 Count Register n	0x00000000
0x40026024	SWU4_TARG0	SWU4 Target Register n	0x00000000
0x40026028	SWU4_HIST0	SWU4 Bandwidth History Register n	0x00000000
0x4002602C	SWU4_CUR0	SWU4 Current Register n	0x00000000
0x40026030	SWU4_CTL1	SWU4 Control Register n	0x00000000
0x40026034	SWU4_LA1	SWU4 Lower Address Register n	0x00000000
0x40026038	SWU4_UA1	SWU4 Upper Address Register n	0x00000000
0x4002603C	SWU4_ID1	SWU4 ID Register n	0x00000000
0x40026040	SWU4_CNT1	SWU4 Count Register n	0x00000000
0x40026044	SWU4_TARG1	SWU4 Target Register n	0x00000000
0x40026048	SWU4_HIST1	SWU4 Bandwidth History Register n	0x00000000
0x4002604C	SWU4_CUR1	SWU4 Current Register n	0x00000000
0x40026050	SWU4_CTL2	SWU4 Control Register n	0x00000000
0x40026054	SWU4_LA2	SWU4 Lower Address Register n	0x00000000
0x40026058	SWU4_UA2	SWU4 Upper Address Register n	0x00000000
0x4002605C	SWU4_ID2	SWU4 ID Register n	0x00000000
0x40026060	SWU4_CNT2	SWU4 Count Register n	0x00000000
0x40026064	SWU4_TARG2	SWU4 Target Register n	0x00000000
0x40026068	SWU4_HIST2	SWU4 Bandwidth History Register n	0x00000000

Table A-64: ADSP-CM40x SWU4 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4002606C	SWU4_CUR2	SWU4 Current Register n	0x00000000
0x40026070	SWU4_CTL3	SWU4 Control Register n	0x00000000
0x40026074	SWU4_LA3	SWU4 Lower Address Register n	0x00000000
0x40026078	SWU4_UA3	SWU4 Upper Address Register n	0x00000000
0x4002607C	SWU4_ID3	SWU4 ID Register n	0x00000000
0x40026080	SWU4_CNT3	SWU4 Count Register n	0x00000000
0x40026084	SWU4_TARG3	SWU4 Target Register n	0x00000000
0x40026088	SWU4_HIST3	SWU4 Bandwidth History Register n	0x00000000
0x4002608C	SWU4_CUR3	SWU4 Current Register n	0x00000000

Table A-65: ADSP-CM40x TAPC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40029000	TAPC0_IDCODE	TAPC0 IDCODE Register	0x028050CB
0x40029004	TAPC0_USERCODE	TAPC0 USERCODE Register	0x00000000
0x40029008	TAPC0_SDBGKEY_CTL	TAPC0 SDBGKEY Control Register	0x00000000
0x4002900C	TAPC0_SDBGKEY_STAT	TAPC0 SDBGKEY Status Register	0x00000000
0x40029010	TAPC0_SDBGKEY0	TAPC0 SDBGKEY0 Register	0x00000000
0x40029014	TAPC0_SDBGKEY1	TAPC0 SDBGKEY1 Register	0x00000000
0x40029018	TAPC0_SDBGKEY2	TAPC0 SDBGKEY2 Register	0x00000000
0x4002901C	TAPC0_SDBGKEY3	TAPC0 SDBGKEY3 Register	0x00000000
0x40029050	TAPC0_USERKEYCMP0	TAPC0 User Key Compare0 Register	0x00000000
0x40029054	TAPC0_USERKEYCMP1	TAPC0 User Key Compare1 Register	0x00000000
0x40029058	TAPC0_USERKEYCMP2	TAPC0 User Key Compare2 Register	0x00000000
0x4002905C	TAPC0_USERKEYCMP3	TAPC0 User Key Compare3 Register	0x00000000
0x400290F0	TAPC0_RCMSG	TAPC0 Run Control Message Register	0x00000000
0x400290F4	TAPC0_RCMSG_CLR	TAPC0 Run Control Message Clear Register	0x00000000
0x400290F8	TAPC0_RCMSG_SET	TAPC0 Run Control Message Set Register	0x00000000
0x400290FC	TAPC0_RCMSG_TOG	TAPC0 Run Control Message Toggle Register	0x00000000

Table A-66: ADSP-CM40x TIMER0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40000804	TIMER0_RUN	TIMER0 Run Register	0x00000000
0x40000808	TIMER0_RUN_SET	TIMER0 Run Set Register	0x00000000
0x4000080C	TIMER0_RUN_CLR	TIMER0 Run Clear Register	0x00000000
0x40000810	TIMER0_STOP_CFG	TIMER0 Stop Configuration Register	0x00000000
0x40000814	TIMER0_STOP_CFG_SET	TIMER0 Stop Configuration Set Register	0x00000000
0x40000818	TIMER0_STOP_CFG_CLR	TIMER0 Stop Configuration Clear Register	0x00000000
0x4000081C	TIMER0_DATA_IMSK	TIMER0 Data Interrupt Mask Register	0x000000FF
0x40000820	TIMER0_STAT_IMSK	TIMER0 Status Interrupt Mask Register	0x000000FF
0x40000824	TIMER0_TRG_MSK	TIMER0 Trigger Master Mask Register	0x000000FF
0x40000828	TIMER0_TRG_IE	TIMER0 Trigger Slave Enable Register	0x00000000
0x4000082C	TIMER0_DATA_ILAT	TIMER0 Data Interrupt Latch Register	0x00000000
0x40000830	TIMER0_STAT_ILAT	TIMER0 Status Interrupt Latch Register	0x00000000
0x40000834	TIMER0_ERR_TYPE	TIMER0 Error Type Status Register	0x00000000
0x40000838	TIMER0_BCAST_PER	TIMER0 Broadcast Period Register	0x00000000
0x4000083C	TIMER0_BCAST_WID	TIMER0 Broadcast Width Register	0x00000000
0x40000840	TIMER0_BCAST_DLY	TIMER0 Broadcast Delay Register	0x00000000
0x40000860	TIMER0_TMR0_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x40000864	TIMER0_TMR0_CNT	TIMER0 Timer n Counter Register	0x00000001
0x40000868	TIMER0_TMR0_PER	TIMER0 Timer n Period Register	0x00000000
0x4000086C	TIMER0_TMR0_WID	TIMER0 Timer n Width Register	0x00000000
0x40000870	TIMER0_TMR0_DLY	TIMER0 Timer n Delay Register	0x00000000
0x40000880	TIMER0_TMR1_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x40000884	TIMER0_TMR1_CNT	TIMER0 Timer n Counter Register	0x00000001
0x40000888	TIMER0_TMR1_PER	TIMER0 Timer n Period Register	0x00000000
0x4000088C	TIMER0_TMR1_WID	TIMER0 Timer n Width Register	0x00000000
0x40000890	TIMER0_TMR1_DLY	TIMER0 Timer n Delay Register	0x00000000
0x400008A0	TIMER0_TMR2_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x400008A4	TIMER0_TMR2_CNT	TIMER0 Timer n Counter Register	0x00000001
0x400008A8	TIMER0_TMR2_PER	TIMER0 Timer n Period Register	0x00000000
0x400008AC	TIMER0_TMR2_WID	TIMER0 Timer n Width Register	0x00000000
0x400008B0	TIMER0_TMR2_DLY	TIMER0 Timer n Delay Register	0x00000000

Table A-66: ADSP-CM40x TIMER0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x400008C0	TIMER0_TMR3_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x400008C4	TIMER0_TMR3_CNT	TIMER0 Timer n Counter Register	0x00000001
0x400008C8	TIMER0_TMR3_PER	TIMER0 Timer n Period Register	0x00000000
0x400008CC	TIMER0_TMR3_WID	TIMER0 Timer n Width Register	0x00000000
0x400008D0	TIMER0_TMR3_DLY	TIMER0 Timer n Delay Register	0x00000000
0x400008E0	TIMER0_TMR4_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x400008E4	TIMER0_TMR4_CNT	TIMER0 Timer n Counter Register	0x00000001
0x400008E8	TIMER0_TMR4_PER	TIMER0 Timer n Period Register	0x00000000
0x400008EC	TIMER0_TMR4_WID	TIMER0 Timer n Width Register	0x00000000
0x400008F0	TIMER0_TMR4_DLY	TIMER0 Timer n Delay Register	0x00000000
0x40000900	TIMER0_TMR5_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x40000904	TIMER0_TMR5_CNT	TIMER0 Timer n Counter Register	0x00000001
0x40000908	TIMER0_TMR5_PER	TIMER0 Timer n Period Register	0x00000000
0x4000090C	TIMER0_TMR5_WID	TIMER0 Timer n Width Register	0x00000000
0x40000910	TIMER0_TMR5_DLY	TIMER0 Timer n Delay Register	0x00000000
0x40000920	TIMER0_TMR6_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x40000924	TIMER0_TMR6_CNT	TIMER0 Timer n Counter Register	0x00000001
0x40000928	TIMER0_TMR6_PER	TIMER0 Timer n Period Register	0x00000000
0x4000092C	TIMER0_TMR6_WID	TIMER0 Timer n Width Register	0x00000000
0x40000930	TIMER0_TMR6_DLY	TIMER0 Timer n Delay Register	0x00000000
0x40000940	TIMER0_TMR7_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x40000944	TIMER0_TMR7_CNT	TIMER0 Timer n Counter Register	0x00000001
0x40000948	TIMER0_TMR7_PER	TIMER0 Timer n Period Register	0x00000000
0x4000094C	TIMER0_TMR7_WID	TIMER0 Timer n Width Register	0x00000000
0x40000950	TIMER0_TMR7_DLY	TIMER0 Timer n Delay Register	0x00000000

Table A-67: ADSP-CM40x TRU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40015000	TRU0_SSR0	TRU0 Slave Select Register	0x00000000
0x40015004	TRU0_SSR1	TRU0 Slave Select Register	0x00000000

Table A-67: ADSP-CM40x TRU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40015008	TRU0_SSR2	TRU0 Slave Select Register	0x00000000
0x4001500C	TRU0_SSR3	TRU0 Slave Select Register	0x00000000
0x40015010	TRU0_SSR4	TRU0 Slave Select Register	0x00000000
0x40015014	TRU0_SSR5	TRU0 Slave Select Register	0x00000000
0x40015018	TRU0_SSR6	TRU0 Slave Select Register	0x00000000
0x4001501C	TRU0_SSR7	TRU0 Slave Select Register	0x00000000
0x40015020	TRU0_SSR8	TRU0 Slave Select Register	0x00000000
0x40015024	TRU0_SSR9	TRU0 Slave Select Register	0x00000000
0x40015028	TRU0_SSR10	TRU0 Slave Select Register	0x00000000
0x4001502C	TRU0_SSR11	TRU0 Slave Select Register	0x00000000
0x40015030	TRU0_SSR12	TRU0 Slave Select Register	0x00000000
0x40015034	TRU0_SSR13	TRU0 Slave Select Register	0x00000000
0x40015038	TRU0_SSR14	TRU0 Slave Select Register	0x00000000
0x4001503C	TRU0_SSR15	TRU0 Slave Select Register	0x00000000
0x40015040	TRU0_SSR16	TRU0 Slave Select Register	0x00000000
0x40015044	TRU0_SSR17	TRU0 Slave Select Register	0x00000000
0x40015048	TRU0_SSR18	TRU0 Slave Select Register	0x00000000
0x4001504C	TRU0_SSR19	TRU0 Slave Select Register	0x00000000
0x40015050	TRU0_SSR20	TRU0 Slave Select Register	0x00000000
0x40015054	TRU0_SSR21	TRU0 Slave Select Register	0x00000000
0x40015058	TRU0_SSR22	TRU0 Slave Select Register	0x00000000
0x4001505C	TRU0_SSR23	TRU0 Slave Select Register	0x00000000
0x40015060	TRU0_SSR24	TRU0 Slave Select Register	0x00000000
0x40015064	TRU0_SSR25	TRU0 Slave Select Register	0x00000000
0x40015068	TRU0_SSR26	TRU0 Slave Select Register	0x00000000
0x4001506C	TRU0_SSR27	TRU0 Slave Select Register	0x00000000
0x40015070	TRU0_SSR28	TRU0 Slave Select Register	0x00000000
0x40015074	TRU0_SSR29	TRU0 Slave Select Register	0x00000000
0x40015078	TRU0_SSR30	TRU0 Slave Select Register	0x00000000
0x4001507C	TRU0_SSR31	TRU0 Slave Select Register	0x00000000
0x40015080	TRU0_SSR32	TRU0 Slave Select Register	0x00000000

Table A-67: ADSP-CM40x TRU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40015084	TRU0_SSR33	TRU0 Slave Select Register	0x00000000
0x40015088	TRU0_SSR34	TRU0 Slave Select Register	0x00000000
0x4001508C	TRU0_SSR35	TRU0 Slave Select Register	0x00000000
0x40015090	TRU0_SSR36	TRU0 Slave Select Register	0x00000000
0x40015094	TRU0_SSR37	TRU0 Slave Select Register	0x00000000
0x40015098	TRU0_SSR38	TRU0 Slave Select Register	0x00000000
0x4001509C	TRU0_SSR39	TRU0 Slave Select Register	0x00000000
0x400150A0	TRU0_SSR40	TRU0 Slave Select Register	0x00000000
0x400150A4	TRU0_SSR41	TRU0 Slave Select Register	0x00000000
0x400150A8	TRU0_SSR42	TRU0 Slave Select Register	0x00000000
0x400150AC	TRU0_SSR43	TRU0 Slave Select Register	0x00000000
0x400150B0	TRU0_SSR44	TRU0 Slave Select Register	0x00000000
0x400150B4	TRU0_SSR45	TRU0 Slave Select Register	0x00000000
0x400150B8	TRU0_SSR46	TRU0 Slave Select Register	0x00000000
0x400150BC	TRU0_SSR47	TRU0 Slave Select Register	0x00000000
0x400150C0	TRU0_SSR48	TRU0 Slave Select Register	0x00000000
0x400150C4	TRU0_SSR49	TRU0 Slave Select Register	0x00000000
0x400150C8	TRU0_SSR50	TRU0 Slave Select Register	0x00000000
0x400150CC	TRU0_SSR51	TRU0 Slave Select Register	0x00000000
0x400150D0	TRU0_SSR52	TRU0 Slave Select Register	0x00000000
0x400150D4	TRU0_SSR53	TRU0 Slave Select Register	0x00000000
0x400150D8	TRU0_SSR54	TRU0 Slave Select Register	0x00000000
0x400150DC	TRU0_SSR55	TRU0 Slave Select Register	0x00000000
0x400150E0	TRU0_SSR56	TRU0 Slave Select Register	0x00000000
0x400150E4	TRU0_SSR57	TRU0 Slave Select Register	0x00000000
0x400150E8	TRU0_SSR58	TRU0 Slave Select Register	0x00000000
0x400150EC	TRU0_SSR59	TRU0 Slave Select Register	0x00000000
0x400150F0	TRU0_SSR60	TRU0 Slave Select Register	0x00000000
0x400150F4	TRU0_SSR61	TRU0 Slave Select Register	0x00000000
0x400150F8	TRU0_SSR62	TRU0 Slave Select Register	0x00000000
0x400150FC	TRU0_SSR63	TRU0 Slave Select Register	0x00000000

Table A-67: ADSP-CM40x TRU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40015100	TRU0_SSR64	TRU0 Slave Select Register	0x00000000
0x40015104	TRU0_SSR65	TRU0 Slave Select Register	0x00000000
0x40015108	TRU0_SSR66	TRU0 Slave Select Register	0x00000000
0x4001510C	TRU0_SSR67	TRU0 Slave Select Register	0x00000000
0x40015110	TRU0_SSR68	TRU0 Slave Select Register	0x00000000
0x400157E0	TRU0_MTR	TRU0 Master Trigger Register	0x00000000
0x400157E8	TRU0_ERRADDR	TRU0 Error Address Register	0x00000000
0x400157EC	TRU0_STAT	TRU0 Status Information Register	0x00000000
0x400157F4	TRU0_GCTL	TRU0 Global Control Register	0x00000000

Table A-68: ADSP-CM40x TWI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40000C00	TWI0_CLKDIV	TWI0 SCL Clock Divider Register	0x00000000
0x40000C04	TWI0_CTL	TWI0 Control Register	0x00000000
0x40000C08	TWI0_SLVCTL	TWI0 Slave Mode Control Register	0x00000000
0x40000C0C	TWI0_SLVSTAT	TWI0 Slave Mode Status Register	0x00000000
0x40000C10	TWI0_SLVADDR	TWI0 Slave Mode Address Register	0x00000000
0x40000C14	TWI0_MSTRCTL	TWI0 Master Mode Control Registers	0x00000000
0x40000C18	TWI0_MSTRSTAT	TWI0 Master Mode Status Register	0x00000000
0x40000C1C	TWI0_MSTRADDR	TWI0 Master Mode Address Register	0x00000000
0x40000C20	TWI0_ISTAT	TWI0 Interrupt Status Register	0x00000000
0x40000C24	TWI0_IMSK	TWI0 Interrupt Mask Register	0x00000000
0x40000C28	TWI0_FIFOCTL	TWI0 FIFO Control Register	0x00000000
0x40000C2C	TWI0_FIFOSTAT	TWI0 FIFO Status Register	0x00000000
0x40000C80	TWI0_TXDATA8	TWI0 Tx Data Single-Byte Register	0x00000000
0x40000C84	TWI0_TXDATA16	TWI0 Tx Data Double-Byte Register	0x00000000
0x40000C88	TWI0_RXDATA8	TWI0 Rx Data Single-Byte Register	0x00000000
0x40000C8C	TWI0_RXDATA16	TWI0 Rx Data Double-Byte Register	0x00000000

Table A-69: ADSP-CM40x UART0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40004004	UART2_CTL	UART0 Control Register	0x00000000
0x40004008	UART2_STAT	UART0 Status Register	0x000000A0
0x4000400C	UART2_SCR	UART0 Scratch Register	0x00000000
0x40004010	UART2_CLK	UART0 Clock Rate Register	0x0000FFFF
0x40004014	UART2_IMSK	UART0 Interrupt Mask Register	0x00000000
0x40004018	UART2_IMSK_SET	UART0 Interrupt Mask Set Register	0x00000000
0x4000401C	UART2_IMSK_CLR	UART0 Interrupt Mask Clear Register	0x00000000
0x40004020	UART2_RBR	UART0 Receive Buffer Register	0x00000000
0x40004024	UART2_THR	UART0 Transmit Hold Register	0x00000000
0x40004028	UART2_TAIP	UART0 Transmit Address/Insert Pulse Register	0x00000000
0x4000402C	UART2_TSR	UART0 Transmit Shift Register	0x000007FF
0x40004030	UART2_RSR	UART0 Receive Shift Register	0x00000000
0x40004034	UART2_TXCNT	UART0 Transmit Counter Register	0x00000000
0x40004038	UART2_RXCNT	UART0 Receive Counter Register	0x00000000

Table A-70: ADSP-CM40x UART1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40003004	UART0_CTL	UART1 Control Register	0x00000000
0x40003008	UART0_STAT	UART1 Status Register	0x000000A0
0x4000300C	UART0_SCR	UART1 Scratch Register	0x00000000
0x40003010	UART0_CLK	UART1 Clock Rate Register	0x0000FFFF
0x40003014	UART0_IMSK	UART1 Interrupt Mask Register	0x00000000
0x40003018	UART0_IMSK_SET	UART1 Interrupt Mask Set Register	0x00000000
0x4000301C	UART0_IMSK_CLR	UART1 Interrupt Mask Clear Register	0x00000000
0x40003020	UART0_RBR	UART1 Receive Buffer Register	0x00000000
0x40003024	UART0_THR	UART1 Transmit Hold Register	0x00000000
0x40003028	UART0_TAIP	UART1 Transmit Address/Insert Pulse Register	0x00000000
0x4000302C	UART0_TSR	UART1 Transmit Shift Register	0x000007FF
0x40003030	UART0_RSR	UART1 Receive Shift Register	0x00000000
0x40003034	UART0_TXCNT	UART1 Transmit Counter Register	0x00000000

Table A-70: ADSP-CM40x UART1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40003038	UART0_RXCNT	UART1 Receive Counter Register	0x00000000

Table A-71: ADSP-CM40x UART2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40003404	UART1_CTL	UART2 Control Register	0x00000000
0x40003408	UART1_STAT	UART2 Status Register	0x000000A0
0x4000340C	UART1_SCR	UART2 Scratch Register	0x00000000
0x40003410	UART1_CLK	UART2 Clock Rate Register	0x0000FFFF
0x40003414	UART1_IMSK	UART2 Interrupt Mask Register	0x00000000
0x40003418	UART1_IMSK_SET	UART2 Interrupt Mask Set Register	0x00000000
0x4000341C	UART1_IMSK_CLR	UART2 Interrupt Mask Clear Register	0x00000000
0x40003420	UART1_RBR	UART2 Receive Buffer Register	0x00000000
0x40003424	UART1_THR	UART2 Transmit Hold Register	0x00000000
0x40003428	UART1_TAIP	UART2 Transmit Address/Insert Pulse Register	0x00000000
0x4000342C	UART1_TSR	UART2 Transmit Shift Register	0x000007FF
0x40003430	UART1_RSR	UART2 Receive Shift Register	0x00000000
0x40003434	UART1_TXCNT	UART2 Transmit Counter Register	0x00000000
0x40003438	UART1_RXCNT	UART2 Receive Counter Register	0x00000000

Table A-72: ADSP-CM40x USB0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x40030000	USB0_FADDR	USB0 Function Address Register	0x00000000
0x40030001	USB0_POWER	USB0 Power and Device Control Register	0x00000020
0x40030002	USB0_INTRTX	USB0 Transmit Interrupt Register	0x00000000
0x40030004	USB0_INTRRX	USB0 Receive Interrupt Register	0x00000000
0x40030006	USB0_INTRTXE	USB0 Transmit Interrupt Enable Register	0x0000000F
0x40030008	USB0_INTRRXE	USB0 Receive Interrupt Enable Register	0x0000000E
0x4003000A	USB0_IRQ	USB0 Common Interrupts Register	0x00000000
0x4003000B	USB0_IEN	USB0 Common Interrupts Enable Register	0x00000000
0x4003000C	USB0_FRAME	USB0 Frame Number Register	0x00000000

Table A-72: ADSP-CM40x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4003000E	USB0_INDEX	USB0 Index Register	0x00000000
0x4003000F	USB0_TESTMODE	USB0 Testmode Register	0x00000000
0x40030010	USB0_EPI[N]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x40030012	USB0_EP0I_CSR[N]_P	USB0 EP0 Configuration and Status (Peripheral) Register	0x00000000
0x40030012	USB0_EP0I_CSR[N]_H	USB0 EP0 Configuration and Status (Host) Register	0x00000000
0x40030012	USB0_EPI[N]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x40030012	USB0_EPI[N]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x40030014	USB0_EPI[N]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x40030016	USB0_EPI[N]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x40030016	USB0_EPI[N]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x40030018	USB0_EP0I_CNT[N]	USB0 EP0 Number of Received Bytes Register	0x00000000
0x40030018	USB0_EPI[N]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x4003001A	USB0_EP0I_TYPE[N]	USB0 EP0 Connection Type Register	0x00000000
0x4003001A	USB0_EPI[N]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x4003001B	USB0_EP0I_NAKLIMIT[N]	USB0 EP0 NAK Limit Register	0x00000000
0x4003001B	USB0_EPI[N]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x4003001C	USB0_EPI[N]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x4003001D	USB0_EPI[N]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x4003001F	USB0_EP0I_CFGDATA[N]	USB0 EP0 Configuration Information Register	0x00000002
0x4003001F	USB0_EPI_FIFOSZ	USB0 FIFO Size	0x00000000
0x40030020	USB0_FIFOB0	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x40030020	USB0_FIFO0	USB0 FIFO Word (32-Bit) Register	0x00000000
0x40030020	USB0_FIFOH0	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x40030024	USB0_FIFOB1	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x40030024	USB0_FIFO1	USB0 FIFO Word (32-Bit) Register	0x00000000
0x40030024	USB0_FIFOH1	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x40030028	USB0_FIFOB2	USB0 FIFO Byte (8-Bit) Register	0x00000000

Table A-72: ADSP-CM40x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40030028	USB0_FIFO2	USB0 FIFO Word (32-Bit) Register	0x00000000
0x40030028	USB0_FIFOH2	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x4003002C	USB0_FIFOB3	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x4003002C	USB0_FIFO3	USB0 FIFO Word (32-Bit) Register	0x00000000
0x4003002C	USB0_FIFOH3	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x40030060	USB0_DEV_CTL	USB0 Device Control Register	0x00000000
0x40030078	USB0_EPINFO	USB0 Endpoint Information Register	0x000000CC
0x40030079	USB0_RAMINFO	USB0 RAM Information Register	0x0000008C
0x4003007A	USB0_LINKINFO	USB0 Link Information Register	0x0000005C
0x4003007B	USB0_VPLEN	USB0 VBUS Pulse Length Register	0x0000003C
0x4003007D	USB0_FS_EOF1	USB0 Full-Speed EOF 1 Register	0x00000077
0x4003007E	USB0_LS_EOF1	USB0 Low-Speed EOF 1 Register	0x00000072
0x4003007F	USB0_SOFT_RST	USB0 Software Reset Register	0x00000000
0x40030080	USB0_MP0_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x40030082	USB0_MP0_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x40030083	USB0_MP0_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x40030084	USB0_MP0_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x40030086	USB0_MP0_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x40030087	USB0_MP0_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x40030088	USB0_MP1_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x4003008A	USB0_MP1_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x4003008B	USB0_MP1_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x4003008C	USB0_MP1_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x4003008E	USB0_MP1_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x4003008F	USB0_MP1_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x40030090	USB0_MP2_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x40030092	USB0_MP2_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000

Table A-72: ADSP-CM40x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x40030093	USB0_MP2_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x40030094	USB0_MP2_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x40030096	USB0_MP2_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x40030097	USB0_MP2_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x40030098	USB0_MP3_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x4003009A	USB0_MP3_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x4003009B	USB0_MP3_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x4003009C	USB0_MP3_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x4003009E	USB0_MP3_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x4003009F	USB0_MP3_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x40030100	USB0_EP0_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x40030102	USB0_EP0_CSR0_P	USB0 EP0 Configuration and Status (Peripheral) Register	0x00000000
0x40030102	USB0_EP0_CSR0_H	USB0 EP0 Configuration and Status (Host) Register	0x00000000
0x40030102	USB0_EP0_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x40030102	USB0_EP0_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x40030104	USB0_EP0_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x40030106	USB0_EP0_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x40030106	USB0_EP0_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x40030108	USB0_EP0_CNT0	USB0 EP0 Number of Received Bytes Register	0x00000000
0x40030108	USB0_EP0_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x4003010A	USB0_EP0_TYPE0	USB0 EP0 Connection Type Register	0x00000000
0x4003010A	USB0_EP0_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x4003010B	USB0_EP0_NAKLIMIT0	USB0 EP0 NAK Limit Register	0x00000000
0x4003010B	USB0_EP0_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x4003010C	USB0_EP0_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x4003010D	USB0_EP0_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000

Table A-72: ADSP-CM40x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4003010F	USB0_EP0_CFGDATA0	USB0 EP0 Configuration Information Register	0x00000002
0x4003010F	USB0_EP0_FIFOSZ	USB0 FIFO Size	0x00000000
0x40030110	USB0_EP1_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x40030112	USB0_EP1_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x40030112	USB0_EP1_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x40030114	USB0_EP1_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x40030116	USB0_EP1_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x40030116	USB0_EP1_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x40030118	USB0_EP1_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x4003011A	USB0_EP1_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x4003011B	USB0_EP1_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x4003011C	USB0_EP1_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x4003011D	USB0_EP1_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x4003011F	USB0_EP1_FIFOSZ	USB0 FIFO Size	0x00000000
0x40030120	USB0_EP2_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x40030122	USB0_EP2_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x40030122	USB0_EP2_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x40030124	USB0_EP2_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x40030126	USB0_EP2_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x40030126	USB0_EP2_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x40030128	USB0_EP2_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x4003012A	USB0_EP2_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x4003012B	USB0_EP2_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x4003012C	USB0_EP2_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x4003012D	USB0_EP2_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000

Table A-72: ADSP-CM40x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4003012F	USB0_EP2_FIFOSZ	USB0 FIFO Size	0x00000000
0x40030130	USB0_EP3_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x40030132	USB0_EP3_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x40030132	USB0_EP3_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x40030134	USB0_EP3_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x40030136	USB0_EP3_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x40030136	USB0_EP3_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x40030138	USB0_EP3_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x4003013A	USB0_EP3_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x4003013B	USB0_EP3_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x4003013C	USB0_EP3_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x4003013D	USB0_EP3_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x4003013F	USB0_EP3_FIFOSZ	USB0 FIFO Size	0x00000000
0x40030200	USB0_DMA_IRQ	USB0 DMA Interrupt Register	0x00000000
0x40030204	USB0_DMA0_CTL	USB0 DMA Channel n Control Register	0x00000000
0x40030208	USB0_DMA0_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x4003020C	USB0_DMA0_CNT	USB0 DMA Channel n Count Register	0x00000000
0x40030214	USB0_DMA1_CTL	USB0 DMA Channel n Control Register	0x00000000
0x40030218	USB0_DMA1_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x4003021C	USB0_DMA1_CNT	USB0 DMA Channel n Count Register	0x00000000
0x40030224	USB0_DMA2_CTL	USB0 DMA Channel n Control Register	0x00000000
0x40030228	USB0_DMA2_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x4003022C	USB0_DMA2_CNT	USB0 DMA Channel n Count Register	0x00000000
0x40030234	USB0_DMA3_CTL	USB0 DMA Channel n Control Register	0x00000000
0x40030238	USB0_DMA3_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x4003023C	USB0_DMA3_CNT	USB0 DMA Channel n Count Register	0x00000000
0x40030244	USB0_DMA4_CTL	USB0 DMA Channel n Control Register	0x00000000
0x40030248	USB0_DMA4_ADDR	USB0 DMA Channel n Address Register	0x00000000

Table A-72: ADSP-CM40x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Address
0x4003024C	USB0_DMA4_CNT	USB0 DMA Channel n Count Register	0x00000000
0x40030254	USB0_DMA5_CTL	USB0 DMA Channel n Control Register	0x00000000
0x40030258	USB0_DMA5_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x4003025C	USB0_DMA5_CNT	USB0 DMA Channel n Count Register	0x00000000
0x40030264	USB0_DMA6_CTL	USB0 DMA Channel n Control Register	0x00000000
0x40030268	USB0_DMA6_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x4003026C	USB0_DMA6_CNT	USB0 DMA Channel n Count Register	0x00000000
0x40030300	USB0_RQPKTCNT0	USB0 EPn Request Packet Count Register	0x00000000
0x40030304	USB0_RQPKTCNT1	USB0 EPn Request Packet Count Register	0x00000000
0x40030308	USB0_RQPKTCNT2	USB0 EPn Request Packet Count Register	0x00000000
0x40030340	USB0_RXDPKTBUFDIS	USB0 RX Double Packet Buffer Disable for Endpoints 1 to 3	0x00000000
0x40030342	USB0_TXDPKTBUFDIS	USB0 TX Double Packet Buffer Disable for Endpoints 1 to 3	0x00000000
0x40030360	USB0_LPM_ATTR	USB0 LPM Attribute Register	0x00000000
0x40030362	USB0_LPM_CTL	USB0 LPM Control Register	0x00000000
0x40030363	USB0_LPM_IEN	USB0 LPM Interrupt Enable Register	0x00000000
0x40030364	USB0_LPM_IRQ	USB0 LPM Interrupt Status Register	0x00000000
0x40030365	USB0_LPM_FADDR	USB0 LPM Function Address Register	0x00000000
0x40030380	USB0_VBUS_CTL	USB0 VBUS Control Register	0x00000000
0x40030382	USB0_IDCTL	USB0 ID Control	0x00000000
0x4003039C	USB0_PHY_CTL	USB0 FS PHY Control	0x00000003
0x4003039E	USB0_PHY_STAT	USB0 FS PHY Status	0x00000000

Table A-73: ADSP-CM40x WDOG0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Address
0x4002D000	WDOG0_CTL	WDOG0 Control Register	0x00000AD0
0x4002D004	WDOG0_CNT	WDOG0 Count Register	0x00000000
0x4002D008	WDOG0_STAT	WDOG0 Watchdog Timer Status Register	0x00000000

Index

Symbols

(Amplitude) Limits for Secondary Filter 0 Register, SINC (SINC_LIMIT0)....29–31
(Amplitude) Limits for Secondary Filter 1 Register, SINC (SINC_LIMIT1)....29–32
(Amplitude) Limits for Secondary Filter 2 Register, SINC (SINC_LIMIT2)....29–33
(Amplitude) Limits for Secondary Filter 3 Register, SINC (SINC_LIMIT3)....29–34
(MaxPktSize).... 22–23

A

Abort Acknowledge 1 Register, CAN (CAN_AA1)....21–28
Abort Acknowledge 2 Register, CAN (CAN_AA2)....21–29
Acceptance Mask (H) Register, CAN (CAN_AM[nn]H).... 21–30
Acceptance Mask (L) Register, CAN (CAN_AM[nn]L)..... 21–31
active mode, DPM.... 6–2-6–4
ADCC_BPTR0 (Base Pointer 0 Register, ADCC)....26–42
ADCC_BPTR1 (DMA Base Pointer 1 Register, ADCC)..... 26–43
ADCC_BWMON0 (Bandwidth Monitor 0 Register, ADCC).... 26–44
ADCC_BWMON1 (Bandwidth Monitor 1 Register, ADCC).... 26–45
ADCC_CBSIZ0 (Circular Buffer Size 0 Register, ADCC)..... 26–46
ADCC_CBSIZ1 (Circular Buffer Size 1 Register, ADCC)..... 26–47
ADCC_CFG (ADC Configuration Register, ADCC)..26–48
ADCC_CTL (Control Register, ADCC).... 26–49
ADCC_ECOL (Event Collision Status Register, ADCC)..... 26–54
ADCC_EIMSK_CLR (Event Interrupt Mask Clear Register, ADCC).... 26–56
ADCC_EIMSK_SET (Event Interrupt Mask Set Register, ADCC).... 26–57
ADCC_EIMSK (Event Interrupt Mask Register, ADCC)..... 26–55
ADCC_EISTAT (Event Interrupt Status Register, ADCC).... 26–58
ADCC_EMISS (Event Miss Status Register, ADCC).. 26–59
ADCC_EPND (Pending Events Status Register, ADCC)..... 26–60
ADCC_ERRMSK_CLR (Error Mask Clear Register, ADCC).... 26–63
ADCC_ERRMSK_SET (Error Mask Set Register, ADCC)... 26–65
ADCC_ERRMSK (Error Mask Register, ADCC).....26–61
ADCC_ERRSTAT (Error Status Register, ADCC).....26–67
ADCC_EVCTL[nn] (Event n Control Register, ADCC)..... 26–69
ADCC_EVDAT[nn] (Event n Data Register, ADCC).26–71
ADCC_EVSTAT[nn] (Event n Status Register, ADCC)..... 26–72
ADCC_EVT[nn] (Event n Time Register, ADCC)..... 26–76
ADCC_EVTEN_CLR (Event Enable Clear Register, ADCC).... 26–74
ADCC_EVTEN_SET (Event Enable Set Register, ADCC)... 26–75
ADCC_EVTEN (Event Enable Register, ADCC).....26–73
ADCC_FIMSK_CLR (Frame Interrupt Mask Clear Register, ADCC).... 26–78
ADCC_FIMSK_SET (Frame Interrupt Mask Set Register, ADCC).... 26–79
ADCC_FIMSK (Frame Interrupt Mask Register, ADCC)..... 26–77
ADCC_FISTAT (Frame Interrupt Status Register, ADCC)... 26–80
ADCC_FRINC0 (Frame Increment 0 Register, ADCC)..... 26–81
ADCC_FRINC1 (Frame Increment 1 Register, ADCC)..... 26–82
ADCC_T0STAT (Timer 0 Status Register, ADCC).... 26–83
ADCC_T1STAT (Timer 1 Status Register, ADCC).... 26–84
ADCC_TCA0 (Timing Control A (ADC0) Register, ADCC).... 26–85
ADCC_TCA1 (Timing Control A (ADC1) Register, ADCC).... 26–86
ADCC_TCB0 (Timing Control B (ADC0) Register, ADCC).... 26–87

ADCC_TCB1 (Timing Control B (ADC1) Register, ADCC)..... 26–88
ADCC_TMR0 (Timer 0 Current Count Register, ADCC)... 26–89
ADCC_TMR1 (Timer 1 Current Count Register, ADCC)... 26–90
ADCC0_ERR interrupt 7–4,26–10
ADCC0_SLV0 trigger 8–5,26–10
ADCC0_SLV1 trigger 8–5,26–10
ADCC0_SLV2 trigger 8–5,26–10
ADCC0_SLV3 trigger 8–5,26–10
ADCC0_SLV4 trigger 8–5,26–10
ADCC0_SLV5 trigger 8–5,26–10
ADCC0_TMR0_EVT interrupt 7–6,26–10
ADCC0_TMR0_EVT trigger 8–3,26–10
ADCC0_TMR1_EVT interrupt 7–6,26–10
ADCC0_TMR1_EVT trigger 8–3,26–10
ADC Configuration Register, ADCC (ADCC_CFG)..26–48
address alignment, SCB, DMA.....12–8
address filtering module (AFM), EMAC CORE..... 23–48,23–49
AFM, EMAC CORE.....23–48
alarm, target time trigger, PTP.....23–71,23–81
arbitration, SCB..... 12–7
arbitration, TWI.....20–6
architectural model, SCB.....3–2
ARDY signal, SMC..... 9–3,9–5,9–10,9–16
asynchronous mode, SMC.....9–6,9–8-9–11,9–13
autobaud detection, UART.....19–7,19–22
autobuffer flow mode, DMA..... 12–21,12–25,12–32,12–35
auto-extend mode, CNT..... 17–11
auto-transmit mode, CAN.....21–10

B

babble interrupt, USB.....22–37
Bandwidth History Register n, SWU (SWU_HIST[n])..... 32–22
Bandwidth Limit Count Current Register, DMA (DMA_BWLCNT_CUR).....12–47
Bandwidth Limit Count Register, DMA (DMA_BWLCNT) 12–46
bandwidth mode, SWU.....32–3,32–5,32–7
Bandwidth Monitor 0 Register, ADCC (ADCC_BWMON0).....26–44
Bandwidth Monitor 1 Register, ADCC (ADCC_BWMON1).....26–45
Bandwidth Monitor Count Current Register, DMA (DMA_BWMCNT_CUR)..... 12–49
Bandwidth Monitor Count Register, DMA (DMA_BWMCNT).....12–48
Bank 0 Control Register, SMC (SMC_B0CTL)..... 9–18
Bank 0 Extended Timing Register, SMC (SMC_B0ETIM)... 9–20
Bank 0 Timing Register, SMC (SMC_B0TIM).....9–22
Bank 1 Control Register, SMC (SMC_B1CTL)..... 9–24
Bank 1 Extended Timing Register, SMC (SMC_B1ETIM)... 9–26
Bank 1 Timing Register, SMC (SMC_B1TIM).....9–28
Bank 2 Control Register, SMC (SMC_B2CTL)..... 9–30
Bank 2 Extended Timing Register, SMC (SMC_B2ETIM)... 9–32
Bank 2 Timing Register, SMC (SMC_B2TIM).....9–34
Bank 3 Control Register, SMC (SMC_B3CTL)..... 9–36
Bank 3 Extended Timing Register, SMC (SMC_B3ETIM)... 9–38
Bank 3 Timing Register, SMC (SMC_B3TIM).....9–40
bank control, SMC.....9–3
Base Pointer 0 Register, ADCC (ADCC_BPTR0)..... 26–42
Base Pointer 0 Register, DACC (DACC_BPTR0)..... 27–18
Base Pointer 1 Register, DACC (DACC_BPTR1)..... 27–19
Bias for Group 0 Register, SINC (SINC_BIAS0).....29–18
Bias for Group 1 Register, SINC (SINC_BIAS1).....29–19
binary encoder mode, CNT.....17–5,17–16
bit rate generation, UART..... 19–6
bit swapping, CRC..... 11–6
Boot Code Register, RCU (RCU_BCODE)..... 30–7
boundary modes, CNT.....17–11,17–17
Broadcast (Write) Control Register, DACC (DACC_BCST_CTL).....27–16
Broadcast Delay Register, TIMER (TIMER_BCAST_DLY).. 14–28
broadcast feature, GP timer..... 14–22
Broadcast Period Register, TIMER (TIMER_BCAST_PER). 14–29
Broadcast Width Register, TIMER (TIMER_BCAST_WID)..... 14–30
burst mode, SMC.....9–16
burst operations, SCB.....23–14,23–16
burst transfers, SCB, DMA.....12–8
bus contention, avoiding, SMC..... 9–3,9–4
bus error, CGU.....4–6

Bus Fault Error Information Register, M4P (M4P_BUSFLT) 2–26
bus grant, SMC..... 9–3
bus interconnection, system, *see* SCB
bypass, PLL..... 4–4

C

Cache Counter Control Register, M4P (M4P_CACHE_CNTCTL)..... 2–33
Cache DCODE Line Fill Counter Register, M4P (M4P_CACHE_DFILL)..... 2–36
Cache DCODE Miss Counter Register, M4P (M4P_CACHE_DMISS)..... 2–37
Cache DCODE Reference Counter Register, M4P (M4P_CACHE_DREF)..... 2–38
Cache ICODE Line Fill Counter Register, M4P (M4P_CACHE_IFILL)..... 2–39
Cache ICODE Miss Counter Register, M4P (M4P_CACHE_IMISS)..... 2–40
Cache ICODE Reference Counter Register, M4P (M4P_CACHE_IREF)..... 2–41
CAN
  architectural concepts..... 21–4
  block diagram..... 21–5
  data transfer modes..... 21–8
  debug modes..... 21–23
  event control..... 21–17, 21–20
  features..... 21–1
  functional description..... 21–2
  interface..... 21–4
  operating modes..... 21–15
  overview..... 21–1
CAN_AA1 (Abort Acknowledge 1 Register, CAN)..... 21–28
CAN_AA2 (Abort Acknowledge 2 Register, CAN)..... 21–29
CAN_AM[nn]H (Acceptance Mask (H) Register, CAN)..... 21–30
CAN_AM[nn]L (Acceptance Mask (L) Register, CAN)..... 21–31
CAN_CEC (Error Counter Register, CAN)..... 21–32
CAN_CLK (Clock Register, CAN)..... 21–33
CAN_CTL (CAN Master Control Register, CAN)..... 21–34
CAN_DBG (Debug Register, CAN)..... 21–36
CAN_ESR (Error Status Register, CAN)..... 21–38
CAN_EWR (Error Counter Warning Level Register, CAN).. 21–40
CAN_GIF (Global CAN Interrupt Flag Register, CAN)..... 21–41
CAN_GIM (Global CAN Interrupt Mask Register, CAN).... 21–44
CAN_GIS (Global CAN Interrupt Status Register, CAN)..... 21–46
CAN_INT (Interrupt Pending Register, CAN)..... 21–49
CAN_MB[nn]_DATA0 (Mailbox Word 0 Register, CAN).... 21–58
CAN_MB[nn]_DATA1 (Mailbox Word 1 Register, CAN).... 21–59
CAN_MB[nn]_DATA2 (Mailbox Word 2 Register, CAN).... 21–60
CAN_MB[nn]_DATA3 (Mailbox Word 3 Register, CAN).... 21–61
CAN_MB[nn]_ID0 (Mailbox ID 0 Register, CAN).... 21–62
CAN_MB[nn]_ID1 (Mailbox ID 1 Register, CAN).... 21–63
CAN_MB[nn]_LENGTH (Mailbox Length Register, CAN) 21–64
CAN_MB[nn]_TIMESTAMP (Mailbox Timestamp Register, CAN)..... 21–65
CAN_MBIM1 (Mailbox Interrupt Mask 1 Register, CAN)... 21–51
CAN_MBIM2 (Mailbox Interrupt Mask 2 Register, CAN)... 21–52
CAN_MBRIF1 (Mailbox Receive Interrupt Flag 1 Register, CAN)..... 21–53
CAN_MBRIF2 (Mailbox Receive Interrupt Flag 2 Register, CAN)..... 21–54
CAN_MBTD (Temporary Mailbox Disable Register, CAN). 21–55
CAN_MBTIF1 (Mailbox Transmit Interrupt Flag 1 Register, CAN)..... 21–56
CAN_MBTIF2 (Mailbox Transmit Interrupt Flag 2 Register, CAN)..... 21–57
CAN_MC1 (Mailbox Configuration 1 Register, CAN)..... 21–66
CAN_MC2 (Mailbox Configuration 2 Register, CAN)..... 21–67
CAN_MD1 (Mailbox Direction 1 Register, CAN)..... 21–68
CAN_MD2 (Mailbox Direction 2 Register, CAN)..... 21–69
CAN_OPSS1 (Overwrite Protection/Single Shot Transmission 1 Register, CAN)..... 21–70
CAN_OPSS2 (Overwrite Protection/Single Shot Transmission 2 Register, CAN)..... 21–71

CAN_RFH1 (Remote Frame Handling 1 Register, CAN)..... 21–72
CAN_RFH2 (Remote Frame Handling 2 Register, CAN)..... 21–73
CAN_RML1 (Receive Message Lost 1 Register, CAN) 21–74
CAN_RML2 (Receive Message Lost 2 Register, CAN) 21–75
CAN_RMP1 (Receive Message Pending 1 Register, CAN).... 21–76
CAN_RMP2 (Receive Message Pending 2 Register, CAN).... 21–77
CAN_STAT (Status Register, CAN)..... 21–78
CAN_TA1 (Transmission Acknowledge 1 Register, CAN)..... 21–80
CAN_TA2 (Transmission Acknowledge 2 Register, CAN)..... 21–81
CAN_TIMING (Timing Register, CAN)..... 21–82
CAN_TRR1 (Transmission Request Reset 1 Register, CAN). 21–83
CAN_TRR2 (Transmission Request Reset 2 Register, CAN). 21–84
CAN_TRS1 (Transmission Request Set 1 Register, CAN)..... 21–85
CAN_TRS2 (Transmission Request Set 2 Register, CAN)..... 21–86
CAN_UCCNF (Universal Counter Configuration Mode Register, CAN)..... 21–87
CAN_UCCNT (Universal Counter Register, CAN).... 21–89
CAN_UCRC (Universal Counter Reload/Capture Register, CAN)..... 21–90
CAN0_RX interrupt 7–7,21–4
CAN0_STAT interrupt 7–7,21–4
CAN0_TX interrupt 7–7,21–4
CAN1_RX interrupt 7–7,21–4
CAN1_STAT interrupt 7–7,21–4
CAN1_TX interrupt 7–7,21–4
CAN Master Control Register, CAN (CAN_CTL)..... 21–34
capture mode *, see* WIDCAP mode
CCLKn, CGU..... 4–9,4–11
CCLKn clock domains..... 3–5
CGU
  block diagram..... 4–4
  definitions..... 4–3
  event control..... 4–5
  features..... 4–1
  functional description..... 4–1
  operating modes..... 4–5
  overview..... 4–1
  PLL..... 4–1,4–4,4–8
  programming..... 4–8
CGU_CLKOUTSEL (CLKOUT Select Register, CGU)..... 4–13
CGU_CTL (Control Register, CGU)..... 4–15
CGU_DIV (Clocks Divisor Register, CGU)..... 4–17
CGU_OSCWDCTL (Oscillator Watchdog Register, CGU). 4–20
CGU_STAT (Status Register, CGU)..... 4–22
CGU_TSCOUNT0 (Time Stamp Counter 32 LSB Register, CGU)..... 4–26
CGU_TSCOUNT1 (Time Stamp Counter 32 MSB Register, CGU)..... 4–27
CGU_TSCTL (Timestamp Control Register, CGU)..... 4–28
CGU_TSVALUE0 (Timestamp Counter Initial 32 LSB Value Register, CGU)..... 4–29
CGU_TSVALUE1 (Time Stamp Counter Initial MSB Value Register, CGU)..... 4–30
CGU0_EVT interrupt 4–2,7–3
CGU0_EVT trigger 4–3,8–2
CGU bus error..... 4–6
CGU event..... 4–6
chaining, EMAC DMA..... 23–18
Channel A Control Register, PWM (PWM_ACTL).... 18–44
Channel A Delay Register, PWM (PWM_DLYA)..... 18–110
Channel A-High Duty-0 Register, PWM (PWM_AH0)..... 18–46
Channel A-High Duty-1 Register, PWM (PWM_AH1)..... 18–48
Channel A-High Full Duty0 Register, PWM (PWM_AH_DUTY0)..... 18–50
Channel A-High Full Duty1 Register, PWM (PWM_AH_DUTY1)..... 18–51
Channel A-High Heightened-Precision Duty-0 Register, PWM (PWM_AH0_HP)..... 18–47
Channel A-High Heightened-Precision Duty-1 Register, PWM (PWM_AH1_HP)..... 18–49
Channel A-Low Duty-0 Register, PWM (PWM_AL0) 18–52
Channel A-Low Duty-1 Register, PWM (PWM_AL1) 18–54
Channel A-Low Full Duty0 Register, PWM (PWM_AL_DUTY0)..... 18–56
Channel A-Low Full Duty1 Register, PWM (PWM_AL_DUTY1)..... 18–57
Channel A-Low Heightened-Precision Duty-0 Register, PWM (PWM_AL0_HP)..... 18–53

Channel A-Low Heightened-Precision Duty-1 Register, PWM (PWM_AL1_HP)....18–55
Channel B Control Register, PWM (PWM_BCTL)....18–58
Channel B Delay Register, PWM (PWM_DLYB).....18–111
Channel B-High Duty-0 Register, PWM (PWM_BH0).....18–60
Channel B-High Duty-1 Register, PWM (PWM_BH1).....18–62
Channel B-High Full Duty0 Register, PWM (PWM_BH_DUTY0)....18–64
Channel B-High Full Duty1 Register, PWM (PWM_BH_DUTY1)....18–65
Channel B-High Heightened-Precision Duty-0 Register, PWM (PWM_BH0_HP)....18–61
Channel B-High Heightened-Precision Duty-1 Register, PWM (PWM_BH1_HP)....18–63
Channel B-Low Duty-0 Register, PWM (PWM_BL0) 18–66
Channel B-Low Duty-1 Register, PWM (PWM_BL1) 18–68
Channel B-Low Full Duty0 Register, PWM (PWM_BL_DUTY0)....18–70
Channel B-Low Full Duty1 Register, PWM (PWM_BL_DUTY1)....18–71
Channel B-Low Heightened-Precision Duty-0 Register, PWM (PWM_BL0_HP)....18–67
Channel B-Low Heightened-Precision Duty-1 Register, PWM (PWM_BL1_HP)....18–69
Channel C Control Register, PWM (PWM_CCTL)... 18–72
Channel C Delay Register, PWM (PWM_DLYC).....18–112
Channel C-High Full Duty0 Register, PWM (PWM_CH_DUTY0)....18–86
Channel C-High Full Duty1 Register, PWM (PWM_CH_DUTY1)....18–87
Channel C-High Pulse Duty Register 0, PWM (PWM_CH0)....18–74
Channel C-High Pulse Duty Register 1, PWM (PWM_CH1)....18–76
Channel C-High Pulse Heightened-Precision Duty Register 0, PWM (PWM_CH0_HP)....18–75
Channel C-High Pulse Heightened-Precision Duty Register 1, PWM (PWM_CH1_HP)....18–77
Channel C-Low Duty-1 Register, PWM (PWM_CL1)18–90
Channel C-Low Full Duty0 Register, PWM (PWM_CL_DUTY0)....18–92
Channel C-Low Full Duty1 Register, PWM (PWM_CL_DUTY1)....18–93
Channel C-Low Heightened-Precision Duty-1 Register, PWM (PWM_CL1_HP)....18–91
Channel C-Low Pulse Duty Register 0, PWM (PWM_CL0)....18–88
Channel C-Low Pulse Duty Register 1, PWM (PWM_CL0_HP)....18–89
Channel Configuration Register, PWM (PWM_CHANCFG)....18–78
channel control, PWM....18–12
Channel D Control Register, PWM (PWM_DCTL)...18–98
Channel D Delay Register, PWM (PWM_DLYD).... 18–113
Channel D-High Duty-0 Register, PWM (PWM_DH0).....18–100
Channel D-High Full Duty0 Register, PWM (PWM_DH_DUTY0)....18–104
Channel D-High Full Duty1 Register, PWM (PWM_DH_DUTY1)....18–105
Channel D-High Pulse Duty Register 1, PWM (PWM_DH1)....18–102
Channel D-High Pulse Heightened-Precision Duty Register 0, PWM (PWM_DH0_HP)....18–101
Channel D High Pulse Heightened-Precision Duty Register 1, PWM (PWM_DH1_HP)....18–103
Channel D-Low Full Duty0 Register, PWM (PWM_DL_DUTY0)....18–114
Channel D-Low Full Duty1 Register, PWM (PWM_DL_DUTY1)....18–115
Channel D-Low Heightened-Precision Duty-0 Register, PWM (PWM_DL0_HP)....18–107
Channel D-Low Heightened-Precision Duty-1 Register, PWM (PWM_DL1_HP)....18–109
Channel D-Low Pulse Duty Register 0, PWM (PWM_DL0)....18–106
Channel D-Low Pulse Duty Register 1, PWM (PWM_DL1)....18–108
checksum, EMAC CORE23–40,23–41,23–45,23–47,23–81
checksum offload, EMAC....23–20
Chop Configuration Register, PWM (PWM_CHOPCFG)....18–85
Circular Buffer Size 0 Register, ADCC (ADCC_CBSIZ0).....26–46
Circular Buffer Size 1 Register, ADCC (ADCC_CBSIZ1).....26–47
CLKOUT Select Register, CGU (CGU_CLKOUTSEL).....4–13
clock, TWI.... 20–3,20–5,20–14

clock alignment, CGU....4–11
Clock Control Register, SINC (SINC_CLK)....29–20
clock domain, SCB....3–5
clock frequency configuration, CGU....4–8
clock generation unit , *see* CGU
clock multiplier/divisor....4–4
Clock Rate Register, SPI (SPI_CLK)....24–40
Clock Rate Register, UART (UART_CLK)....19–25
Clock Register, CAN (CAN_CLK)....21–33
Clocks Divisor Register, CGU (CGU_DIV)....4–17
clock stretching mode, TWI....20–9
CNT
event control....17–16
features....17–1
functional description....17–2
operating modes....17–4
overview....17–1
programming....17–9,17–13
CNT_CFG (Configuration Register, CNT)....17–18
CNT_CMD (Command Register, CNT)....17–21
CNT_CNTR (Counter Register, CNT)....17–23
CNT_DEBNCE (Debounce Register, CNT)....17–24
CNT_IMSK (Interrupt Mask Register, CNT)....17–26
CNT_MAX (Maximum Count Register, CNT)....17–29
CNT_MDIV (M Value for Divider, CNT)....17–30
CNT_MIN (Minimum Count Register, CNT)....17–31
CNT_NDIV (N Value for Divider, CNT)....17–32
CNT_STAT (Status Register, CNT)....17–33
CNT0_STAT interrupt....7–4,17–3
CNT0_STAT trigger....8–3,17–3
CNT1_STAT interrupt....7–4,17–3
CNT1_STAT trigger....8–3,17–3
CNT1_UD trigger....8–5,17–4
CNT2_STAT interrupt....7–5,17–3
CNT2_STAT trigger....8–3,17–3
CNT3_STAT interrupt....7–5,17–3
CNT3_STAT trigger....8–3,17–3
Code Cache Configuration and Status Register, M4P (M4P_CACHE_CFG)....2–27
Code Cache Parity Error Address Register, M4P (M4P_CACHE_PEADDR)....2–46
Command Register, CNT (CNT_CMD)....17–21
Common Interrupts Enable Register, USB (USB_IEN)....22–164
Common Interrupts Register, USB (USB_IRQ)....22–171
companding, SPORT....25–13,25–33
Configuration 0 Register, HAE (HAE_CFG0)....28–18
Configuration 1 Register, HAE (HAE_CFG1)....28–19
Configuration 2 Register, HAE (HAE_CFG2)....28–20
Configuration 3 Register, HAE (HAE_CFG3)....28–22
Configuration 4 Register, HAE (HAE_CFG4)....28–23
Configuration Register, CNT (CNT_CFG)....17–18
Configuration Register, CPTMR (CPTMR_CFG[n])....15–7
Configuration Register, DMA (DMA_CFG)....12–50
Control 0 Register, DACC (DACC_CTL0)....27–24
Control 1 Register, DACC (DACC_CTL1)....27–27
controller area network (CAN), *see* CAN
Control Register, ADCC (ADCC_CTL)....26–49
Control Register, CGU (CGU_CTL)....4–15
Control Register, CRC (CRC_CTL)....11–26
Control Register, DPM (DPM_CTL)....6–12
Control Register, PWM (PWM_CTL)....18–94
Control Register, RCU (RCU_CTL)....30–12
Control Register, SINC (SINC_CTL)....29–22
Control Register, SPI (SPI_CTL)....24–41
Control Register, SPU (SPU_CTL)....5–7
Control Register, TWI (TWI_CTL)....20–18
Control Register, UART (UART_CTL)....19–26
Control Register, WDOG (WDOG_CTL)....16–5
Control Register n, SWU (SWU_CTL[n])....32–11
COP pin....7–13
Core Clock Buffer Disable Register, DPM (DPM_CCBF_DIS)....6–8
Core Clock Buffer Enable Register, DPM (DPM_CCBF_EN)....6–9
Core Clock Buffer Status Register, DPM (DPM_CCBF_STAT)....6–10
Core Clock Buffer Status Sticky Register, DPM (DPM_CCBF_STAT_STKY)....6–11
core clock n (CCLKn)....4–3,4–4,4–9
core mode, UART....19–11,19–12,19–17
core reset, RCU....30–5
Core Reset outputs Control Register, RCU (RCU_CRCTL)....30–10
Core Reset outputs Status Register, RCU (RCU_CRSTAT)....30–11
core subblocks, EMAC....23–10
Count 0 Register, DACC (DACC_CNT0)....27–20
Count 1 Register, DACC (DACC_CNT1)....27–21
counter, general purpose , *see* CNT
counter mode, *see* EXTCLK mode
Counter Register, CNT (CNT_CNTR)....17–23

Counter Register, CPTMR (CPTMR_CNT[n]) 15–8
Count Register, WDOG (WDOG_CNT) 16–4
Count Register n, SWU (SWU_CNT[n]) 32–10
CPTMR_CFG[n] (Configuration Register, CPTMR) ... 15–7
CPTMR_CNT[n] (Counter Register, CPTMR) 15–8
CPTMR_DATA_ILAT (Data Interrupt Latch Status Register, CPTMR) 15–9
CPTMR_DATA_IMSK_CLR (Data Interrupt Mask Clear Register, CPTMR) 15–11
CPTMR_DATA_IMSK_SET (Data Interrupt Mask Set Register, CPTMR) 15–12
CPTMR_DATA_IMSK (Data Interrupt Mask Register, CPTMR) 15–10
CPTMR_RUN_CLR (Run Clear Register, CPTMR) .. 15–14
CPTMR_RUN_SET (Run Set Register, CPTMR) 15–15
CPTMR_RUN (Run Register, CPTMR) 15–13
CPTMR_STAT_ILAT (Interrupt Latch Status Register, CPTMR) 15–16
CPTMR_STAT_IMSK_CLR (Status Interrupt Mask Clear Register, CPTMR) 15–18
CPTMR_STAT_IMSK_SET (Status Interrupt Mask Set Register, CPTMR) 15–19
CPTMR_STAT_IMSK (Status Interrupt Mask Register, CPTMR) 15–17
CPTMR_TON[n] (On-time Capture Register, CPTMR) 15–20
CPTMR0_CPT0_ERR interrupt 7–7,15–3
CPTMR0_CPT0_MEAS interrupt 7–7,15–3
CPTMR0_CPT0_SLV0 trigger 8–7,15–3
CPTMR0_CPT0_SLV1 trigger 8–7,15–3
CPTMR0_CPT1_ERR interrupt 7–7,15–3
CPTMR0_CPT1_MEAS interrupt 7–7,15–3
CPTMR0_CPT1_SLV0 trigger 8–7,15–3
CPTMR0_CPT1_SLV1 trigger 8–7,15–3
CPTMR0_CPT2_ERR interrupt 7–7,15–3
CPTMR0_CPT2_MEAS interrupt 7–7,15–3
CPTMR0_CPT2_SLV0 trigger 8–7,15–3
CPTMR0_CPT2_SLV1 trigger 8–7,15–3
CRC
  block diagram 11–4
  data transfer modes 11–9
  DMA transfers 11–2
  EMAC CORE 23–44,23–45
  event control 11–11
  mode configuration 11–12
  operating modes 11–9
  overview 11–1
  programming 11–12
CRC_COMP (Data Compare Register, CRC) 11–25
CRC_CTL (Control Register, CRC) 11–26
CRC_DCNT (Data Word Count Register, CRC) 11–29
CRC_DCNTCAP (Data Count Capture Register, CRC) 11–30
CRC_DCNTRLD (Data Word Count Reload Register, CRC) 11–31
CRC_DFIFO (Data FIFO Register, CRC) 11–32
CRC_FILLVAL (Fill Value Register, CRC) 11–33
CRC_INEN_CLR (Interrupt Enable Clear Register, CRC) 11–35
CRC_INEN_SET (Interrupt Enable Set Register, CRC) 11–36
CRC_INEN (Interrupt Enable Register, CRC) 11–34
CRC_POLY (Polynomial Register, CRC) 11–37
CRC_RESULT_CUR (CRC Current Result Register, CRC) 11–38
CRC_RESULT_FIN (CRC Final Result Register, CRC) 11–39
CRC_STAT (Status Register, CRC) 11–40
CRC0_DCNTEXP interrupt 7–6,11–3
CRC0_ERR interrupt 7–6,11–3
CRC Current Result Register, CRC (CRC_RESULT_CUR) 11–38
CRC Final Result Register, CRC (CRC_RESULT_FIN) 11–39
CTX, EMAC CORE 23–44
Current Address Register, DMA (DMA_ADDR_CUR) 12–45
Current Count(1D) or intra-row XCNT (2D) Register, DMA (DMA_XCNT_CUR) 12–65
Current Count 0 Register, DACC (DACC_CNTCUR0) 27–22
Current Count 1 Register, DACC (DACC_CNTCUR1) 27–23
Current Descriptor Pointer Register, DMA (DMA_DSCPTR_CUR) 12–58
Current Register n, SWU (SWU_CUR[n]) 32–16
Current Row Count (2D only) Register, DMA (DMA_YCNT_CUR) 12–68
cut through mode, *see* threshold mode, EMAC FIFO
cyclic redundancy check, *see* CRC
cyclic redundancy check (CRC) 3–5

D

DACC_BCST_CTL (Broadcast (Write) Control Register, DACC)..... 27–16
DACC_BPTR0 (Base Pointer 0 Register, DACC)..... 27–18
DACC_BPTR1 (Base Pointer 1 Register, DACC)..... 27–19
DACC_CNT0 (Count 0 Register, DACC)..... 27–20
DACC_CNT1 (Count 1 Register, DACC)..... 27–21
DACC_CNTCUR0 (Current Count 0 Register, DACC)..... 27–22
DACC_CNTCUR1 (Current Count 1 Register, DACC)..... 27–23
DACC_CTL0 (Control 0 Register, DACC)..... 27–24
DACC_CTL1 (Control 1 Register, DACC)..... 27–27
DACC_DAT0 (Data FIFO 0 Register, DACC)..... 27–30
DACC_DAT1 (Data FIFO 1 Register, DACC)..... 27–31
DACC_ERRMSK_CLR (Error Mask Clear Register, DACC)..... 27–34
DACC_ERRMSK_SET (Error Mask Set Register, DACC)... 27–35
DACC_ERRMSK (Error Mask Register, DACC)..... 27–32
DACC_ERRSTAT (Error Status Register, DACC)..... 27–36
DACC_IMSK_CLR (Interrupt Mask Clear Register, DACC)..... 27–40
DACC_IMSK_SET (Interrupt Mask Set Register, DACC)... 27–41
DACC_IMSK (Interrupt Mask Register, DACC)..... 27–38
DACC_ISTAT (Interrupt Status Register, DACC)..... 27–42
DACC_MOD0 (Modify 0 Register, DACC)..... 27–44
DACC_MOD1 (Modify 1 Register, DACC)..... 27–45
DACC_STAT (Status Register, DACC)..... 27–46
DACC_TC0 (Timing Control 0 Register, DACC)..... 27–48
DACC_TC1 (Timing Control 1 Register, DACC)..... 27–49
DACC0_DAC0 interrupt 7–6
DACC0_DAC1 interrupt 7–6
DACC0_ERR interrupt 7–4
data chaining, EMAC DMA..... 23–18
Data Compare Register, CRC (CRC_COMP)..... 11–25
Data Count Capture Register, CRC (CRC_DCNTCAP)..... 11–30
Data FIFO 0 Register, DACC (DACC_DAT0)..... 27–30
Data FIFO 1 Register, DACC (DACC_DAT1)..... 27–31
Data FIFO Register, CRC (CRC_DFIFO)..... 11–32
Data Interrupt Latch Register, TIMER (TIMER_DATA_ILAT)..... 14–31
Data Interrupt Latch Status Register, CPTMR (CPTMR_DATA_ILAT)..... 15–9
Data Interrupt Mask Clear Register, CPTMR (CPTMR_DATA_IMSK_CLR)..... 15–11
Data Interrupt Mask Register, CPTMR (CPTMR_DATA_IMSK)..... 15–10
Data Interrupt Mask Register, TIMER (TIMER_DATA_IMSK)..... 14–32
Data Interrupt Mask Set Register, CPTMR (CPTMR_DATA_IMSK_SET)..... 15–12
data mirroring, CRC..... 11–6,11–7
data packing, multichannel DMA(MCPDE), SPORT. 25–33
data signals, SPORT..... 25–5,25–6
data types, SPORT..... 25–13
Data Word Count Register, CRC (CRC_DCNT)..... 11–29
Data Word Count Reload Register, CRC (CRC_DCNTRLD)..... 11–31
DCLK clock domain..... 3–5
DCLK frequency, CGU..... 4–9,4–11
dead-time control, PWM..... 18–19,18–20,18–22
Dead Time Register, PWM (PWM_DT)..... 18–116
Debounce Register, CNT (CNT_DEBNCE)..... 17–24
debug features, CAN..... 21–23
debugging, UART..... 19–8
Debug Register, CAN (CAN_DBG)..... 21–36
Debug Register, EMAC (EMAC_DBG)..... 23–94
deep sleep mode, configuring..... 6–5
deep sleep mode, DPM..... 6–2-6–5
Delay Register, SPI (SPI_DLY)..... 24–47
descriptor-array mode, DMA..... 12–16,12–26,12–36
descriptor fetch, DMA..... 12–4,12–16
descriptor-list flow mode, DMA. 12–16,12–26,12–28,12–37
descriptor-on-demand mode, DMA..... 12–27
descriptor queues, DMA..... 12–40,12–42
descriptors, EMAC DMA 23–17,23–20,23–27,23–33,23–86
Device Control Register, USB (USB_DEV_CTL)..... 22–73
DeviceNet mode, CAN..... 21–12
device requests, USB..... 22–15,22–30
DIDT Coefficient Register, HAE (HAE_DIDT_COEF)..... 28–24
DIDT Gain Register, HAE (HAE_DIDT_GAIN)..... 28–25
direction counter, CNT..... 17–5
direct memory access, *see* DMA
DMA
- and EMAC..... 23–24
- architectural concepts..... 12–7
- block diagram..... 12–6
- CRC..... 11–2

data transfer modes....12–28
definitions.... 12–4
EMAC....23–2,23–17,23–18,23–32,23–86
errors.... 12–21,12–23,12–24
event control....12–29,12–39,12–41
features.... 12–1
functional description....12–3
modes, configuring SPI.... 24–36
operating modes.... 12–24,12–25
operation flow.... 12–15
overview.... 12–1
programming....12–33,12–39
SCB interface.... 12–7,12–8
transfers, SPORT....25–40
triggers....12–32,12–39
UART.... 19–16
USB.... 22–46,22–65
DMA_ADDR_CUR (Current Address Register, DMA).....
....12–45
DMA_ADDRSTART (Start Address of Current Buffer Register, DMA).... 12–44
DMA_BWLCNT_CUR (Bandwidth Limit Count Current Register, DMA).... 12–47
DMA_BWLCNT (Bandwidth Limit Count Register, DMA)
....12–46
DMA_BWMCNT_CUR (Bandwidth Monitor Count Current Register, DMA).... 12–49
DMA_BWMCNT (Bandwidth Monitor Count Register, DMA).... 12–48
DMA_CFG (Configuration Register, DMA).... 12–50
DMA_DSCPTR_CUR (Current Descriptor Pointer Register, DMA).... 12–58
DMA_DSCPTR_NXT (Pointer to Next Initial Descriptor Register, DMA).... 12–59
DMA_DSCPTR_PRV (Previous Initial Descriptor Pointer Register, DMA).... 12–60
DMA_STAT (Status Register, DMA).... 12–61
DMA_XCNT_CUR (Current Count(1D) or intra-row XCNT (2D) Register, DMA).... 12–65
DMA_XCNT (Inner Loop Count Start Value Register, DMA).... 12–64
DMA_XMOD (Inner Loop Address Increment Register, DMA).... 12–66
DMA_YCNT_CUR (Current Row Count (2D only) Register, DMA).... 12–68
DMA_YCNT (Outer Loop Count Start Value (2D only) Register, DMA).... 12–67
DMA_YMOD (Outer Loop Address Increment (2D only) Register, DMA).... 12–69
DMA Base Pointer 1 Register, ADCC (ADCC_BPTR1).....
....26–43
DMA Bus Mode Register, EMAC (EMAC_DMA_BUSMODE).... 23–100
DMA Channel n Address Register, USB (USB_DMA[n]_ADDR)....22–75
DMA Channel n Control Register, USB (USB_DMA[n]_CTL).... 22–77
DMA Channel n Count Register, USB (USB_DMA[n]_CNT).... 22–76
DMA channels, SCB.... 3–4
DMA Interrupt Enable Register, EMAC (EMAC_DMA_IEN)....23–103
DMA Interrupt Register, USB (USB_DMA_IRQ)..... 22–80
DMA Missed Frame Register, EMAC (EMAC_DMA_MISS_FRM)....23–106
DMA Operation Mode Register, EMAC (EMAC_DMA_OPMODE).... 23–107
DMA Rx Buffer Current Register, EMAC (EMAC_DMA_RXBUF_CUR)....23–111
DMA Rx Descriptor Current Register, EMAC (EMAC_DMA_RXDSC_CUR).... 23–113
DMA Rx Descriptor List Address Register, EMAC (EMAC_DMA_RXDSC_ADDR)....23–112
DMA Rx Interrupt Watch Dog Register, EMAC (EMAC_DMA_RXIWDOG).... 23–114
DMA Rx Poll Demand register, EMAC (EMAC_DMA_RXPOLL)....23–115
DMA SCB Bus Mode Register, EMAC (EMAC_DMA_BMMODE).... 23–97
DMA SCB Status Register, EMAC (EMAC_DMA_BMSTAT)....23–99
DMA Status Register, EMAC (EMAC_DMA_STAT).....
....23–116
DMA Tx Buffer Current Register, EMAC (EMAC_DMA_TXBUF_CUR)....23–120
DMA Tx Descriptor Current Register, EMAC (EMAC_DMA_TXDSC_CUR).... 23–122
DMA Tx Descriptor List Address Register, EMAC (EMAC_DMA_TXDSC_ADDR).... 23–121
DMA Tx Poll Demand Register, EMAC (EMAC_DMA_TXPOLL)....23–123

down, up, counter mode, CNT..... 17–5,17–16
DPM
CGU interface.....4–3
event control.....6–5
features.....6–1
functional description.....6–1
operating modes.....6–3
overview.....6–1
programming.....6–5
DPM_CCBF_DIS (Core Clock Buffer Disable Register, DPM).....6–8
DPM_CCBF_EN (Core Clock Buffer Enable Register, DPM).....6–9
DPM_CCBF_STAT_STKY (Core Clock Buffer Status Sticky Register, DPM).....6–11
DPM_CCBF_STAT (Core Clock Buffer Status Register, DPM).....6–10
DPM_CTL (Control Register, DPM).....6–12
DPM_SCBF_DIS (System Clock Buffer Disable Register, DPM).....6–14
DPM_STAT (Status Register, DPM).....6–15
DPM_WAKE_EN (Wakeup Enable Register, DPM).....6–17
DPM_WAKE_POL (Wakeup Polarity Register, DPM). 6–18
DPM_WAKE_STAT (Wakeup Status Register, DPM).. 6–19
DPM0_EVT interrupt 6–1,7–7
dual I/O mode, SPI 24–12
duty cycle, PWM.....18–12,18–20,18–40
dynamic memory clock (DCLK)..... 4–4
dynamic power management unit, *see* DPM

E

EMAC
architectural concepts..... 23–13
block diagram.....23–8
clock.....23–12,23–50,23–51
core..... 23–10
CORE.....23–38,23–78
DMA controller.....
.....23–17,23–20,23–24,23–27,23–32,23–86
DMA register summary table.....23–18
event control.....23–17,23–37,23–55,23–80
features.....23–1
FIFO.....23–36
functional description.....23–2
interfaces..... 23–8
management counters (MMC)..... 23–53
MDIO management interface..... 23–50
overview..... 23–1
precision time protocol engine (PTP)..... 23–2,23–55
programming.....23–14,23–77,23–84
receive modules..... 23–44
station management interface (SMI).....23–2,23–50
system crossbar interface (SCB)..... 23–13
transmit modules.....23–39
EMAC_ADDR0_HI (MAC Address 0 High Register, EMAC)..... 23–92
EMAC_ADDR0_LO (MAC Address 0 Low Register, EMAC)..... 23–93
EMAC_DBG (Debug Register, EMAC).....23–94
EMAC_DMA_BMMODE (DMA SCB Bus Mode Register, EMAC)..... 23–97
EMAC_DMA_BMSTAT (DMA SCB Status Register, EMAC)..... 23–99
EMAC_DMA_BUSMODE (DMA Bus Mode Register, EMAC)..... 23–100
EMAC_DMA_IEN (DMA Interrupt Enable Register, EMAC)..... 23–103
EMAC_DMA_MISS_FRM (DMA Missed Frame Register, EMAC)..... 23–106
EMAC_DMA_OPMODE (DMA Operation Mode Register, EMAC)..... 23–107
EMAC_DMA_RXBUF_CUR (DMA Rx Buffer Current Register, EMAC)..... 23–111
EMAC_DMA_RXDSC_ADDR (DMA Rx Descriptor List Address Register, EMAC)..... 23–112
EMAC_DMA_RXDSC_CUR (DMA Rx Descriptor Current Register, EMAC).....23–113
EMAC_DMA_RXIWDOG (DMA Rx Interrupt Watch Dog Register, EMAC)..... 23–114
EMAC_DMA_RXPOLL (DMA Rx Poll Demand register, EMAC)..... 23–115
EMAC_DMA_STAT (DMA Status Register, EMAC).....
.....23–116
EMAC_DMA_TXBUF_CUR (DMA Tx Buffer Current Register, EMAC)..... 23–120
EMAC_DMA_TXDSC_ADDR (DMA Tx Descriptor List Address Register, EMAC)..... 23–121
EMAC_DMA_TXDSC_CUR (DMA Tx Descriptor Current Register, EMAC).....23–122
EMAC_DMA_TXPOLL (DMA Tx Poll Demand Register, EMAC)..... 23–123

EMAC_FLOWCTL (FLow Control Register, EMAC)..... 23–124
EMAC_HASHTBL_HI (Hash Table High Register, EMAC) 23–126
EMAC_HASHTBL_LO (Hash Table Low Register, EMAC) 23–127
EMAC_IMSK (Interrupt Mask Register, EMAC)..... 23–128
EMAC_IPC_RXIMSK (MMC IPC Rx Interrupt Mask Register, EMAC)..... 23–130
EMAC_IPC_RXINT (MMC IPC Rx Interrupt Register, EMAC)..... 23–136
EMAC_ISTAT (Interrupt Status Register, EMAC).... 23–140
EMAC_MACCFG (MAC Configuration Register, EMAC).. 23–142
EMAC_MACFRMFILT (MAC Rx Frame Filter Register, EMAC)..... 23–146
EMAC_MMC_CTL (MMC Control Register, EMAC)..... 23–149
EMAC_MMC_RXIMSK (MMC Rx Interrupt Mask Register, EMAC)..... 23–151
EMAC_MMC_RXINT (MMC Rx Interrupt Register, EMAC)..... 23–154
EMAC_MMC_TXIMSK (MMC TX Interrupt Mask Register, EMAC)..... 23–157
EMAC_MMC_TXINT (MMC Tx Interrupt Register, EMAC)..... 23–161
EMAC_RX1024TOMAX_GB (Rx 1024- to Max-Byte Frames (Good/Bad) Register, EMAC)..... 23–165
EMAC_RX128TO255_GB (Rx 128- to 255-Byte Frames (Good/Bad) Register, EMAC)..... 23–166
EMAC_RX256TO511_GB (Rx 256- to 511-Byte Frames (Good/Bad) Register, EMAC)..... 23–167
EMAC_RX512TO1023_GB (Rx 512- to 1023-Byte Frames (Good/Bad) Register, EMAC)..... 23–168
EMAC_RX64_GB (Rx 64-Byte Frames (Good/Bad) Register, EMAC)..... 23–169
EMAC_RX65TO127_GB (Rx 65- to 127-Byte Frames (Good/Bad) Register, EMAC)..... 23–170
EMAC_RXALIGN_ERR (Rx alignment Error Register, EMAC)..... 23–171
EMAC_RXBCASTFRM_G (Rx Broadcast Frames (Good) Register, EMAC)..... 23–172
EMAC_RXCRC_ERR (Rx CRC Error Register, EMAC)..... 23–173
EMAC_RXFIFO_OVF (Rx FIFO Overflow Register, EMAC)..... 23–174
EMAC_RXFRMCNT_GB (Rx Frame Count (Good/Bad) Register, EMAC)..... 23–175
EMAC_RXICMP_ERR_FRM (Rx ICMP Error Frames Register, EMAC)..... 23–176
EMAC_RXICMP_ERR_OCT (Rx ICMP Error Octets Register, EMAC)..... 23–177
EMAC_RXICMP_GD_FRM (Rx ICMP Good Frames Register, EMAC)..... 23–178
EMAC_RXICMP_GD_OCT (Rx ICMP Good Octets Register, EMAC)..... 23–179
EMAC_RXIPV4_FRAG_FRM (Rx IPv4 Datagrams Fragmented Frames Register, EMAC).....23–180
EMAC_RXIPV4_FRAG_OCT (Rx IPv4 Datagrams Fragmented Octets Register, EMAC)..... 23–181
EMAC_RXIPV4_GD_FRM (Rx IPv4 Datagrams (Good) Register, EMAC)..... 23–182
EMAC_RXIPV4_GD_OCT (Rx IPv4 Datagrams Good Octets Register, EMAC)..... 23–183
EMAC_RXIPV4_HDR_ERR_FRM (Rx IPv4 Datagrams Header Errors Register, EMAC)..... 23–184
EMAC_RXIPV4_HDR_ERR_OCT (Rx IPv4 Datagrams Header Errors Register, EMAC)..... 23–185
EMAC_RXIPV4_NOPAY_FRM (Rx IPv4 Datagrams No Payload Frame Register, EMAC).....23–186
EMAC_RXIPV4_NOPAY_OCT (Rx IPv4 Datagrams No Payload Octets Register, EMAC)..... 23–187
EMAC_RXIPV4_UDSBL_FRM (Rx IPv4 UDP Disabled Frames Register, EMAC)..... 23–188
EMAC_RXIPV4_UDSBL_OCT (Rx IPv4 UDP Disabled Octets Register, EMAC)..... 23–189
EMAC_RXIPV6_GD_FRM (Rx IPv6 Datagrams Good Frames Register, EMAC)..... 23–190
EMAC_RXIPV6_GD_OCT (Rx IPv6 Good Octets Register, EMAC)..... 23–191
EMAC_RXIPV6_HDR_ERR_FRM (Rx IPv6 Datagrams Header Error Frames Register, EMAC).....23–192
EMAC_RXIPV6_HDR_ERR_OCT (Rx IPv6 Header Errors Register, EMAC)..... 23–193
EMAC_RXIPV6_NOPAY_FRM (Rx IPv6 Datagrams No Payload Frames Register, EMAC)..... 23–194
EMAC_RXIPV6_NOPAY_OCT (Rx IPv6 No Payload Octets Register, EMAC).....23–195
EMAC_RXJAB_ERR (Rx Jab Error Register, EMAC)..... 23–196
EMAC_RXLEN_ERR (Rx Length Error Register, EMAC)... 23–197

EMAC_RXMCASTFRM_G (Rx Multicast Frames (Good) Register, EMAC)..... 23–198
EMAC_RXOCTCNT_G (Rx Octet Count (Good) Register, EMAC)..... 23–199
EMAC_RXOCTCNT_GB (Rx Octet Count (Good/Bad) Register, EMAC)..... 23–200
EMAC_RXOORTYPE (Rx Out Of Range Type Register, EMAC)..... 23–201
EMAC_RXOSIZE_G (Rx Oversize (Good) Register, EMAC)..... 23–202
EMAC_RXPAUSEFRM (Rx Pause Frames Register, EMAC) 23–203
EMAC_RXRUNT_ERR (Rx Runt Error Register, EMAC)... 23–204
EMAC_RXTCP_ERR_FRM (Rx TCP Error Frames Register, EMAC)..... 23–205
EMAC_RXTCP_ERR_OCT (Rx TCP Error Octets Register, EMAC)..... 23–206
EMAC_RXTCP_GD_FRM (Rx TCP Good Frames Register, EMAC)..... 23–207
EMAC_RXTCP_GD_OCT (Rx TCP Good Octets Register, EMAC)..... 23–208
EMAC_RXUCASTFRM_G (Rx Unicast Frames (Good) Register, EMAC)..... 23–209
EMAC_RXUDP_ERR_FRM (Rx UDP Error Frames Register, EMAC)..... 23–210
EMAC_RXUDP_ERR_OCT (Rx UDP Error Octets Register, EMAC)..... 23–211
EMAC_RXUDP_GD_FRM (Rx UDP Good Frames Register, EMAC)..... 23–212
EMAC_RXUDP_GD_OCT (Rx UDP Good Octets Register, EMAC)..... 23–213
EMAC_RXUSIZE_G (Rx Undersize (Good) Register, EMAC)..... 23–214
EMAC_RXVLANFRM_GB (Rx VLAN Frames (Good/Bad) Register, EMAC)..... 23–215
EMAC_RXWDOG_ERR (Rx Watch Dog Error Register, EMAC)..... 23–216
EMAC_SMI_ADDR (SMI Address Register, EMAC)..... 23–217
EMAC_SMI_DATA (SMI Data Register, EMAC).... 23–219
EMAC_TM_ADDEND (Time Stamp Addend Register, EMAC)..... 23–220
EMAC_TM_AUXSTMP_NSEC (Time Stamp Auxiliary TS Nano Seconds Register, EMAC)..... 23–221
EMAC_TM_AUXSTMP_SEC (Time Stamp Auxiliary TM Seconds Register, EMAC).....23–222
EMAC_TM_CTL (Time Stamp Control Register, EMAC)... 23–223
EMAC_TM_HISEC (Time Stamp High Second Register, EMAC)..... 23–227
EMAC_TM_NSEC (Time Stamp Nanoseconds Register, EMAC)..... 23–228
EMAC_TM_NSECUPDT (Time Stamp Nanoseconds Update Register, EMAC).....23–229
EMAC_TM_NTGTM (Time Stamp Target Time Nanoseconds Register, EMAC).....23–230
EMAC_TM_PPSCTL (PPS Control Register, EMAC)..... 23–231
EMAC_TM_PPSINTVL (Time Stamp PPS Interval Register, EMAC)..... 23–234
EMAC_TM_PPSWIDTH (PPS Width Register, EMAC)..... 23–235
EMAC_TM_SEC (Time Stamp Low Seconds Register, EMAC)..... 23–236
EMAC_TM_SECUPDT (Time Stamp Seconds Update Register, EMAC)..... 23–237
EMAC_TM_STMPSTAT (Time Stamp Status Register, EMAC)..... 23–238
EMAC_TM_SUBSEC (Time Stamp Sub Second Increment Register, EMAC)..... 23–240
EMAC_TM_TGTM (Time Stamp Target Time Seconds Register, EMAC)..... 23–241
EMAC_TX1024TOMAX_GB (Tx 1024- to Max-Byte Frames (Good/Bad) Register, EMAC)..... 23–242
EMAC_TX128TO255_GB (Tx 128- to 255-Byte Frames (Good/Bad) Register, EMAC)..... 23–243
EMAC_TX256TO511_GB (Tx 256- to 511-Byte Frames (Good/Bad) Register, EMAC)..... 23–244
EMAC_TX512TO1023_GB (Tx 512- to 1023-Byte Frames (Good/Bad) Register, EMAC)..... 23–245
EMAC_TX64_GB (Tx 64-Byte Frames (Good/Bad) Register, EMAC)..... 23–246
EMAC_TX65TO127_GB (Tx 65- to 127-Byte Frames (Good/Bad) Register, EMAC)..... 23–247
EMAC_TXBCASTFRM_G (Tx Broadcast Frames (Good) Register, EMAC)..... 23–248
EMAC_TXBCASTFRM_GB (Tx Broadcast Frames (Good/Bad) Register, EMAC).....23–249
EMAC_TXCARR_ERR (Tx Carrier Error Register, EMAC) 23–250

EMAC_TXDEFERRED (Tx Deferred Register, EMAC)..... 23–251
EMAC_TXEXCESSCOL (Tx Excess Collision Register, EMAC)..... 23–252
EMAC_TXEXCESSDEF (Tx Excess Deferral Register, EMAC)..... 23–253
EMAC_TXFRMCNT_G (Tx Frame Count (Good) Register, EMAC)..... 23–254
EMAC_TXFRMCNT_GB (Tx Frame Count (Good/Bad) Register, EMAC)..... 23–255
EMAC_TXLATECOL (Tx Late Collision Register, EMAC). 23–256
EMAC_TXMCASTFRM_G (Tx Multicast Frames (Good) Register, EMAC)..... 23–257
EMAC_TXMCASTFRM_GB (Tx Multicast Frames (Good/Bad) Register, EMAC).....23–258
EMAC_TXMULTCOL_G (Tx Multiple Collision (Good) Register, EMAC)..... 23–259
EMAC_TXOCTCNT_G (Tx Octet Count (Good) Register, EMAC)..... 23–260
EMAC_TXOCTCNT_GB (Tx OCT Count (Good/Bad) Register, EMAC)..... 23–261
EMAC_TXPAUSEFRM (Tx Pause Frame Register, EMAC). 23–262
EMAC_TXSNGCOL_G (Tx Single Collision (Good) Register, EMAC)..... 23–263
EMAC_TXUCASTFRM_GB (Tx Unicast Frames (Good/Bad) Register, EMAC).....23–264
EMAC_TXUNDR_ERR (Tx Underflow Error Register, EMAC)..... 23–265
EMAC_TXVLANFRM_G (Tx VLAN Frames (Good) Register, EMAC)..... 23–266
EMAC_VLANTAG (VLAN Tag Register, EMAC)... 23–267
EMAC0_STAT interrupt 7–6,23–7
EMAC0_STAT trigger 8–3,23–7
encoder mode, binary, CNT.....17–5
Endpoint Information Register, USB (USB_EPINFO)..... 22–104
endpoints, USB.... 22–12,22–18,22–30,22–35,22–64,22–66
EP0 Configuration and Status (Host) Register, USB (USB_EP0_CSR[n]_H)..... 22–96
EP0 Configuration and Status (Host) Register, USB (USB_EP0I_CSR[N]_H).....22–85
EP0 Configuration and Status (Peripheral) Register, USB (USB_EP0_CSR[n]_P)..... 22–99
EP0 Configuration and Status (Peripheral) Register, USB (USB_EP0I_CSR[N]_P).....22–88
EP0 Configuration Information Register, USB (USB_EP0_CFGDATA[n]).....22–93
EP0 Configuration Information Register, USB (USB_EP0I_CFGDATA[N])..... 22–82
EP0 Connection Type Register, USB (USB_EP0_TYPE[n]).. 22–103
EP0 Connection Type Register, USB (USB_EP0I_TYPE[N]).....22–92
EP0 NAK Limit Register, USB (USB_EP0_NAKLIMIT[n]) 22–102
EP0 NAK Limit Register, USB (USB_EP0I_NAKLIMIT[N])..... 22–91
EP0 Number of Received Bytes Register, USB (USB_EP0_CNT[n])..... 22–95
EP0 Number of Received Bytes Register, USB (USB_EP0I_CNT[N]).....22–84
EPn Number of Bytes Received Register, USB (USB_EP[n]_RXCNT).....22–132
EPn Number of Bytes Received Register, USB (USB_EPI[N]_RXCNT).....22–105
EPn Receive Configuration and Status (Host) Register, USB (USB_EP[n]_RXCSR_H)..... 22–133
EPn Receive Configuration and Status (Host) Register, USB (USB_EPI[N]_RXCSR_H).....22–106
EPn Receive Configuration and Status (Peripheral) Register, USB (USB_EP[n]_RXCSR_P).....22–138
EPn Receive Configuration and Status (Peripheral) Register, USB (USB_EPI[N]_RXCSR_P)..... 22–111
EPn Receive Maximum Packet Length Register, USB (USB_EP[n]_RXMAXP).....22–143
EPn Receive Maximum Packet Length Register, USB (USB_EPI[N]_RXMAXP)..... 22–116
EPn Receive Polling Interval Register, USB (USB_EP[n]_RXINTERVAL).....22–142
EPn Receive Polling Interval Register, USB (USB_EPI[N]_RXINTERVAL)..... 22–115
EPn Receive Type Register, USB (USB_EP[n]_RXTYPE)..... 22–144
EPn Receive Type Register, USB (USB_EPI[N]_RXTYPE)... 22–117
EPn Request Packet Count Register, USB (USB_RQPKTCNT[n]).....22–196
EPn Transmit Configuration and Status (Host) Register, USB (USB_EP[n]_TXCSR_H).....22–146

EPn Transmit Configuration and Status (Host) Register, USB (USB_EPI[N]_TXCSR_H)..... 22–119
EPn Transmit Configuration and Status (Peripheral) Register, USB (USB_EP[n]_TXCSR_P).....22–150
EPn Transmit Configuration and Status (Peripheral) Register, USB (USB_EPI[N]_TXCSR_P)..... 22–123
EPn Transmit Maximum Packet Length Register, USB (USB_EP[n]_TXMAXP).....22–154
EPn Transmit Maximum Packet Length Register, USB (USB_EPI[N]_TXMAXP)..... 22–127
EPn Transmit Polling Interval Register, USB (USB_EP[n]_TXINTERVAL).....22–153
EPn Transmit Polling Interval Register, USB (USB_EPI[N]_TXINTERVAL)..... 22–126
EPn Transmit Type Register, USB (USB_EP[n]_TXTYPE)... 22–155
EPn Transmit Type Register, USB (USB_EPI[N]_TXTYPE) 22–128
Error Address Register, TRU (TRU_ERRADDR)..... 8–10
Error Counter Register, CAN (CAN_CEC)..... 21–32
Error Counter Warning Level Register, CAN (CAN_EWR).. 21–40
Error Mask Clear Register, ADCC (ADCC_ERRMSK_CLR).....26–63
Error Mask Clear Register, DACC (DACC_ERRMSK_CLR).....27–34
Error Mask Register, ADCC (ADCC_ERRMSK).....26–61
Error Mask Register, DACC (DACC_ERRMSK).....27–32
Error Mask Set Register, ADCC (ADCC_ERRMSK_SET)... 26–65
Error Mask Set Register, DACC (DACC_ERRMSK_SET)... 27–35
errors, DMA..... 12–21,12–23,12–24
see also DMA event control
Error Status Register, ADCC (ADCC_ERRSTAT).....26–67
Error Status Register, CAN (CAN_ESR)..... 21–38
Error Status Register, DACC (DACC_ERRSTAT).....27–36
Error Type Status Register, TIMER (TIMER_ERR_TYPE).. 14–33
Ethernet frame size, EMAC..... 23–85
ethernet media access controller, *see* EMAC
Ethernet PTP frame.....23–67
event, CGU..... 4–6
Event Collision Status Register, ADCC (ADCC_ECOL)..... 26–54
event controller, *see* SEC
Event Enable Clear Register, ADCC (ADCC_EVTEN_CLR).....26–74
Event Enable Register, ADCC (ADCC_EVTEN).....26–73
Event Enable Set Register, ADCC (ADCC_EVTEN_SET)... 26–75
Event Interrupt Mask Clear Register, ADCC (ADCC_EIMSK_CLR).....26–56
Event Interrupt Mask Register, ADCC (ADCC_EIMSK)..... 26–55
Event Interrupt Mask Set Register, ADCC (ADCC_EIMSK_SET)..... 26–57
Event Interrupt Status Register, ADCC (ADCC_EISTAT).... 26–58
Event Miss Status Register, ADCC (ADCC_EMISS).. 26–59
Event n Control Register, ADCC (ADCC_EVCTL[nn])..... 26–69
Event n Data Register, ADCC (ADCC_EVDAT[nn]).26–71
Event n Status Register, ADCC (ADCC_EVSTAT[nn])..... 26–72
Event n Time Register, ADCC (ADCC_EVT[nn])..... 26–76
events, binary, CNT..... 17–5
EXTCLK mode, GP timer.....14–5,14–18,14–25

F

fast mode, TWI..... 20–7
Fault Control Register, SEC (SEC_FCTL)..... 7–20
Fault COP Period Current Register, SEC (SEC_FCOPP_CUR)..... 7–19
Fault COP Period Register, SEC (SEC_FCOPP).....7–18
Fault Delay Current Register, SEC (SEC_FDLY_CUR) 7–24
Fault Delay Register, SEC (SEC_FDLY).....7–23
Fault End Register, SEC (SEC_FEND)..... 7–25
fault management interface, SEC, *see* SFI
Fault Source ID Register, SEC (SEC_FSID).....7–26
Fault Status Register, SEC (SEC_FSTAT).....7–29
Fault System Reset Delay Current Register, SEC (SEC_FSRDLY_CUR).....7–28
Fault System Reset Delay Register, SEC (SEC_FSRDLY)..... 7–27
FIFO Byte (8-Bit) Register, USB (USB_FIFOB[n])...22–158
FIFO configuration, USB.....22–4
FIFO Control Register, TWI (TWI_FIFOCTL)..... 20–20
FIFO Half-Word (16-Bit) Register, USB (USB_FIFOH[n]).. 22–159
FIFO reads and writes, SMC..... 9–15
FIFO Size, USB (USB_EP[n]_FIFOSZ)..... 22–131

FIFO Size, USB (USB_EPI_FIFOSZ).....22–130
FIFO Status Register, TWI (TWI_FIFOSTAT)..... 20–22
FIFO Word (32-Bit) Register, USB (USB_FIFO[n]). 22–160
Fill Value Register, CRC (CRC_FILLVAL)..... 11–33
filtering, AFM, EMAC CORE..... 23–48
finish command, DMA.....12–11,12–12
flash mode, SMC.....9–6,9–11,9–13,9–16
flash page mode, SMC.....9–14
FLow Control Register, EMAC (EMAC_FLOWCTL).....
.....23–124
flush operation, EMAC FIFO.....23–37
frame control, EMAC CORE..... 23–40,23–45
frame delay (MFD), multichannel, SPORT 25–32
Frame Increment 0 Register, ADCC (ADCC_FRINC0).....
.....26–81
Frame Increment 1 Register, ADCC (ADCC_FRINC1).....
.....26–82
Frame Interrupt Mask Clear Register, ADCC (ADCC_FIMSK_CLR).....26–78
Frame Interrupt Mask Register, ADCC (ADCC_FIMSK).....
.....26–77
Frame Interrupt Mask Set Register, ADCC (ADCC_FIMSK_SET)..... 26–79
Frame Interrupt Status Register, ADCC (ADCC_FISTAT)...
.....26–80
Frame Number Register, USB (USB_FRAME).....22–161
frame packet, start of, USB..... 22–27
frame sync
combinations table, SPORT.....25–11
early vs. late, SPORT.....25–23
mode, SPORT.....25–5,25–22,25–31
signal, SPORT.....25–4,25–10
FS PHY Control, USB (USB_PHY_CTL)..... 22–190
FS PHY Status, USB (USB_PHY_STAT).....22–191
FTX, EMAC CORE.....23–44
full-on mode, DPM.....6–2,6–3
Full-Speed EOF 1 Register, USB (USB_FS_EOF1)...22–162
Function Address Register, USB (USB_FADDR)..... 22–157

G

gate drive unit, PWM..... 18–23
general-purpose input/output, *see* GPIO
general purpose timer modulator , *see* GP timer
Global CAN Interrupt Flag Register, CAN (CAN_GIF).....
.....21–41
Global CAN Interrupt Mask Register, CAN (CAN_GIM)....
.....21–44
Global CAN Interrupt Status Register, CAN (CAN_GIS).....
.....21–46
Global Control Register, SEC (SEC_GCTL).....7–31
Global Control Register, SWU (SWU_GCTL)..... 32–17
Global Control Register, TRU (TRU_GCTL).....8–11
global locking, SPU..... 5–3
Global Raise Register, SEC (SEC_RAISE).....7–34
Global Status Register, SEC (SEC_GSTAT)..... 7–32
Global Status Register, SWU (SWU_GSTAT).....32–18
GPIO..... 13–1
functionality.....13–5
operating modes..... 13–2,13–5
pin interrupt signal flow figure..... 13–1
see also PORT
GP timer
event control.....14–13,14–22
features..... 14–1
illegal states.....14–22
interface.....14–4
operating modes..... 14–5,14–14,14–18
overview..... 14–1
programming.....14–21

H

HAE_CFG0 (Configuration 0 Register, HAE)..... 28–18
HAE_CFG1 (Configuration 1 Register, HAE)..... 28–19
HAE_CFG2 (Configuration 2 Register, HAE)..... 28–20
HAE_CFG3 (Configuration 3 Register, HAE)..... 28–22
HAE_CFG4 (Configuration 4 Register, HAE)..... 28–23
HAE_DIDT_COEF (DIDT Coefficient Register, HAE).....
.....28–24
HAE_DIDT_GAIN (DIDT Gain Register, HAE)..... 28–25
HAE_H[nn]_INDX (Harmonic n Index Register, HAE).....
.....28–26
HAE_ISAMPLE (I (Current) Sample Register, HAE)..28–27
HAE_IWAVEFORM (I (Current) Waveform Register, HAE).....28–28
HAE_RESULTS_RAM (Results RAM Register, HAE)28–29
HAE_RUN (Run Register, HAE).....28–30
HAE_STAT (Status Register, HAE).....28–31
HAE_VLEVEL (Voltage Level Register, HAE)..... 28–32
HAE_VSAMPLE (V (Voltage) Sample Register, HAE)28–33
HAE_VWAVEFORM (V (Voltage) Waveform Register, HAE).....28–34

HAE0_RXDMA_CH0_ERR interrupt 7–4,28–3
HAE0_RXDMA_CH0 interrupt 7–6,28–3
HAE0_RXDMA_CH0 trigger 8–3,8–6
HAE0_RXDMA_CH1_ERR interrupt 7–4,28–3
HAE0_RXDMA_CH1 interrupt 7–6,28–3
HAE0_RXDMA_CH1 trigger 8–3,8–6
HAE0_STAT interrupt 7–6,28–3
HAE0_TXDMA_ERR interrupt 7–4,28–3
HAE0_TXDMA interrupt 7–6,28–3
HAE0_TXDMA trigger 8–3,8–6
half duplex mode, EMAC.....23–22,23–36,23–43
Half SPORT 'A' Control 2 Register, SPORT (SPORT_CTL2_A).....25–56
Half SPORT 'A' Control Register, SPORT (SPORT_CTL_A).....25–58
Half SPORT 'A' Divisor Register, SPORT (SPORT_DIV_A)..... 25–75
Half SPORT 'A' Error Register, SPORT (SPORT_ERR_A).. 25–77
Half SPORT 'A' Multi-channel 0-31 Select Register, SPORT (SPORT_CS0_A).....25–48
Half SPORT 'A' Multi-channel 32-63 Select Register, SPORT (SPORT_CS1_A)..... 25–50
Half SPORT 'A' Multi-channel 64-95 Select Register, SPORT (SPORT_CS2_A)..... 25–52
Half SPORT 'A' Multi-channel 96-127 Select Register, SPORT (SPORT_CS3_A)..... 25–54
Half SPORT 'A' Multi-channel Control Register, SPORT (SPORT_MCTL_A)..... 25–81
Half SPORT 'A' Multi-channel Status Register, SPORT (SPORT_MSTAT_A).....25–85
Half SPORT 'A' Rx Buffer (Primary) Register, SPORT (SPORT_RXPRI_A)..... 25–87
Half SPORT 'A' Rx Buffer (Secondary) Register, SPORT (SPORT_RXSEC_A)..... 25–89
Half SPORT 'A' Tx Buffer (Primary) Register, SPORT (SPORT_TXPRI_A).....25–91
Half SPORT 'A' Tx Buffer (Secondary) Register, SPORT (SPORT_TXSEC_A).....25–93
Half SPORT 'B' Control 2 Register, SPORT (SPORT_CTL2_B).....25–57
Half SPORT 'B' Control Register, SPORT (SPORT_CTL_B).....25–66
Half SPORT 'B' Divisor Register, SPORT (SPORT_DIV_B)..... 25–76
Half SPORT 'B' Error Register, SPORT (SPORT_ERR_B).. 25–79
Half SPORT 'B' Multi-channel 0-31 Select Register, SPORT (SPORT_CS0_B).....25–49
Half SPORT 'B' Multi-channel 32-63 Select Register, SPORT (SPORT_CS1_B)..... 25–51
Half SPORT 'B' Multichannel 64-95 Select Register, SPORT (SPORT_CS2_B)..... 25–53
Half SPORT 'B' Multichannel 96-127 Select Register, SPORT (SPORT_CS3_B)..... 25–55
Half SPORT 'B' Multi-channel Control Register, SPORT (SPORT_MCTL_B)..... 25–83
Half SPORT 'B' Multi-channel Status Register, SPORT (SPORT_MSTAT_B).....25–86
Half SPORT 'B' Rx Buffer (Primary) Register, SPORT (SPORT_RXPRI_B)..... 25–88
Half SPORT 'B' Rx Buffer (Secondary) Register, SPORT (SPORT_RXSEC_B)..... 25–90
Half SPORT 'B' Tx Buffer (Primary) Register, SPORT (SPORT_TXPRI_B).....25–92
Half SPORT 'B' Tx Buffer (Secondary) Register, SPORT (SPORT_TXSEC_B)..... 25–94
hardware flow control, UART.....19–5,19–17,19–23
hardware reset, RCU.....30–4
Harmonic n Index Register, HAE (HAE_H[nn]_INDX)..... 28–26
hash filtering mode, AFM, EMAC CORE.....23–48,23–78
Hash Table High Register, EMAC (EMAC_HASHTBL_HI) 23–126
Hash Table Low Register, EMAC (EMAC_HASHTBL_LO) 23–127

hierarchical model, SCB..... 3–3
History Status Register, SINC (SINC_HIS_STAT)..... 29–25
host mode, USB..... 22–2,22–11,22–29,22–34,22–46
host negotiation, USB.....22–63
HSPORT block, SPORT.....25–9

I

I (Current) Sample Register, HAE (HAE_ISAMPLE)..28–27
I (Current) Waveform Register, HAE (HAE_IWAVE-FORM).....28–28
I2S mode, SPORT.....25–25,25–34
IDCODE Register, TAPC (TAPC_IDCODE).....33–3
ID Control, USB (USB_IDCTL).....22–163
identifier value (SID), SEC.....7–8

idle command, DMA....12–10
ID Register n, SWU (SWU_ID[n])....32–23
IEEE1588, PTP, EMAC....23–55,23–56
IEEE 802.3 Ethernet packet structure, EMAC....23–84
INCR burst type, SCB....23–14
Index Register, USB (USB_INDEX)....22–166
infrared data association, *see* IrDA
Inner Loop Address Increment Register, DMA (DMA_XMOD)....12–66
Inner Loop Count Start Value Register, DMA (DMA_XCNT)....12–64
input mode, PORT....13–6
interfaces, SCB....3–5
interrupt acknowledge, SEC....7–10
interrupt command, DMA....12–11
interrupt configuration, SEC....7–15
interrupt controller, *see* SEC
Interrupt Enable Clear Register, CRC (CRC_INEN_CLR)....11–35
Interrupt Enable Register, CRC (CRC_INEN)....11–34
Interrupt Enable Set Register, CRC (CRC_INEN_SET)....11–36
interrupt flow, SEC....7–1
interrupt flow diagram, EMAC....23–76
Interrupt Latch Register, PWM (PWM_ILAT)....18–117
Interrupt Latch Status Register, CPTMR (CPTMR_STAT_ILAT)....15–16
Interrupt Mask Clear Register, DACC (DACC_IMSK_CLR)....27–40
Interrupt Mask Clear Register, SPI (SPI_IMSK_CLR) 24–55
Interrupt Mask Clear Register, UART (UART_IMSK_CLR)....19–36
Interrupt Mask Register, CNT (CNT_IMSK)....17–26
Interrupt Mask Register, DACC (DACC_IMSK)....27–38
Interrupt Mask Register, EMAC (EMAC_IMSK)....23–128
Interrupt Mask Register, PWM (PWM_IMSK)....18–119
Interrupt Mask Register, SPI (SPI_IMSK)....24–53
Interrupt Mask Register, TWI (TWI_IMSK)....20–23
Interrupt Mask Register, UART (UART_IMSK)....19–33
Interrupt Mask Set Register, DACC (DACC_IMSK_SET)....27–41
Interrupt Mask Set Register, SPI (SPI_IMSK_SET)....24–58
Interrupt Mask Set Register, UART (UART_IMSK_SET)....19–38
Interrupt Pending Register, CAN (CAN_INT)....21–49
Interrupt Status Register, DACC (DACC_ISTAT)....27–42
Interrupt Status Register, EMAC (EMAC_ISTAT)....23–140
Interrupt Status Register, TWI (TWI_ISTAT)....20–25
IP header, EMAC CORE....23–41
IPv4-UDP PTP frame....23–64
IPv6-UDP PTP frame....23–66
IrDA SIR mode, UART....19–9,19–13

L

latency, peripheral SCB....3–4
left-justified mode, SPORT....25–26
Level Control for Group 0 Register, SINC (SINC_LEVEL0)....29–27
Level Control for Group 1 Register, SINC (SINC_LEVEL1)....29–29
Link Information Register, USB (USB_LINKINFO) 22–173
LIN protocol, UART....19–11
local interconnect network (LIN) protocol, UART....19–11
locking, global, SPU....5–3,5–5
lookup table (LUT), CRC....11–1,11–7,11–12
loopback mode, UART....19–9
Lower Address Register n, SWU (SWU_LA[n])....32–24
Low-Speed EOF 1 Register, USB (USB_LS_EOF1)..22–183
LPM Attribute Register, USB (USB_LPM_ATTR)...22–174
LPM Control Register, USB (USB_LPM_CTL)....22–175
LPM Function Address Register, USB (USB_LPM_FADDR)....22–177
LPM Interrupt Enable Register, USB (USB_LPM_IEN)....22–178
LPM Interrupt Status Register, USB (USB_LPM_IRQ)....22–180

M

M4_SCS0_BusFault interrupt....7–2
M4_SCS0_DebugMonitor interrupt....7–2
M4_SCS0_HardFault interrupt....7–2
M4_SCS0_MemoryManagement interrupt....7–2
M4_SCS0_NonMaskableint interrupt....7–2
M4_SCS0_PendSV interrupt....7–2
M4_SCS0_RESET interrupt....7–2
M4_SCS0_SVCall interrupt....7–2
M4_SCS0_SysTick interrupt....7–2
M4_SCS0_UsageFault interrupt....7–2
M4P_BUSFLT (Bus Fault Error Information Register, M4P)....2–26
M4P_CACHE_CFG (Code Cache Configuration and Status Register, M4P)....2–27

M4P_CACHE_CNTCTL (Cache Counter Control Register, M4P)....2–33
M4P_CACHE_DFILL (Cache DCODE Line Fill Counter Register, M4P)....2–36
M4P_CACHE_DMISS (Cache DCODE Miss Counter Register, M4P)....2–37
M4P_CACHE_DREF (Cache DCODE Reference Counter Register, M4P)....2–38
M4P_CACHE_IFILL (Cache ICODE Line Fill Counter Register, M4P)....2–39
M4P_CACHE_IMISS (Cache ICODE Miss Counter Register, M4P)....2–40
M4P_CACHE_IREF (Cache ICODE Reference Counter Register, M4P)....2–41
M4P_CACHE_MEMX (MEMX Space Configuration Register, M4P)....2–42
M4P_CACHE_MEMY (MEMY Space Configuration Register, M4P)....2–44
M4P_CACHE_PEADDR (Code Cache Parity Error Address Register, M4P).... 2–46
M4P_SRAM_CFG (SRAM Configuration Register, M4P)... 2–47
M4P_SRAM_PEADDR_CORE (SRAM Parity Error Address (Core) Register, M4P).... 2–49
M4P_SRAM_PEADDR_DMA (SRAM Parity Error Address (DMA) Register, M4P).... 2–51
M4P_STCALIB (SysTick Calibration Register, M4P)... 2–53
M4P0_BUS_FAULT interrupt 2–3,7–3
M4P0_CORE_SRAM_PERR interrupt 2–3,7–3
M4P0_DMA_SRAM_PERR interrupt 2–3,7–3
M4P0_L1CC_PERR interrupt 2–3,7–3
M4P0_LOCKUP interrupt 2–3,7–3
M4P0_SRAM_PERR_FLT interrupt 2–3,7–3
MAC Address 0 High Register, EMAC (EMAC_ADDR0_HI).... 23–92
MAC Address 0 Low Register, EMAC (EMAC_ADDR0_LO).... 23–93
MAC Configuration Register, EMAC (EMAC_MACCFG).. 23–142
MAC FIFO layer, *see* EMAC, FIFO
MAC management counters, *see* MMC
MAC Rx Frame Filter Register, EMAC (EMAC_MACFRMFILT)....23–146
Mailbox Configuration 1 Register, CAN (CAN_MC1)..... 21–66
Mailbox Configuration 2 Register, CAN (CAN_MC2)..... 21–67
Mailbox Direction 1 Register, CAN (CAN_MD1)..... 21–68
Mailbox Direction 2 Register, CAN (CAN_MD2)..... 21–69
mailbox feature, CAN....21–4,21–6,21–11,21–14
Mailbox ID 0 Register, CAN (CAN_MB[nn]_ID0).... 21–62
Mailbox ID 1 Register, CAN (CAN_MB[nn]_ID1).... 21–63
Mailbox Interrupt Mask 1 Register, CAN (CAN_MBIM1)... 21–51
Mailbox Interrupt Mask 2 Register, CAN (CAN_MBIM2)... 21–52
Mailbox Length Register, CAN (CAN_MB[nn]_LENGTH) 21–64
Mailbox Receive Interrupt Flag 1 Register, CAN (CAN_MBRIF1)....21–53
Mailbox Receive Interrupt Flag 2 Register, CAN (CAN_MBRIF2)....21–54
Mailbox Timestamp Register, CAN (CAN_MB[nn]_TIMESTAMP)....21–65
Mailbox Transmit Interrupt Flag 1 Register, CAN (CAN_MBTIF1)....21–56
Mailbox Transmit Interrupt Flag 2 Register, CAN (CAN_MBTIF2)....21–57
Mailbox Word 0 Register, CAN (CAN_MB[nn]_DATA0).... 21–58
Mailbox Word 1 Register, CAN (CAN_MB[nn]_DATA1).... 21–59
Mailbox Word 2 Register, CAN (CAN_MB[nn]_DATA2).... 21–60
Mailbox Word 3 Register, CAN (CAN_MB[nn]_DATA3).... 21–61
management counters (MMC).... 23–53
Masked Interrupt Clear Register, SPI (SPI_ILAT_CLR)..... 24–50
Masked Interrupt Condition Register, SPI (SPI_ILAT)24–48
Master 0 IB Sync Mode, SCB (SCB_MST00_IB_SYNC)..... 3–17
Master 0 Read Quality of Service, SCB (SCB_MST00_IB_RQOS).... 3–16
Master 0 Write Quality of Service, SCB (SCB_MST00_IB_WQOS)....3–18
Master 10 Read Quality of Service, SCB (SCB_MST10_RQOS).... 3–38
Master 10 Write Quality of Service, SCB (SCB_MST10_WQOS)....3–39

Master 11 Read Quality of Service, SCB (SCB_MST11_RQOS)..... 3–40
Master 11 Write Quality of Service, SCB (SCB_MST11_WQOS)..... 3–41
Master 12 Read Quality of Service, SCB (SCB_MST12_RQOS)..... 3–42
Master 12 Write Quality of Service, SCB (SCB_MST12_WQOS)..... 3–43
Master 13 Read Quality of Service, SCB (SCB_MST13_RQOS)..... 3–44
Master 13 Write Quality of Service, SCB (SCB_MST13_WQOS)..... 3–45
Master 14 Read Quality of Service, SCB (SCB_MST14_RQOS)..... 3–46
Master 14 Write Quality of Service, SCB (SCB_MST14_WQOS)..... 3–47
Master 15 Read Quality of Service, SCB (SCB_MST15_RQOS)..... 3–48
Master 15 Write Quality of Service, SCB (SCB_MST15_WQOS)..... 3–49
Master 16 Read Quality of Service, SCB (SCB_MST16_RQOS)..... 3–50
Master 16 Write Quality of Service, SCB (SCB_MST16_WQOS)..... 3–51
Master 17 Read Quality of Service, SCB (SCB_MST17_RQOS)..... 3–52
Master 17 Write Quality of Service, SCB (SCB_MST17_WQOS)..... 3–53
Master 18 Read Quality of Service, SCB (SCB_MST18_RQOS)..... 3–54
Master 18 Write Quality of Service, SCB (SCB_MST18_WQOS)..... 3–55
Master 19 Read Quality of Service, SCB (SCB_MST19_RQOS)..... 3–56
Master 19 Write Quality of Service, SCB (SCB_MST19_WQOS)..... 3–57
Master 1 IB Sync Mode, SCB (SCB_MST01_IB_SYNC)..... 3–20
Master 1 Read Quality of Service, SCB (SCB_MST01_IB_RQOS)..... 3–19
Master 1 Write Quality of Service, SCB (SCB_MST01_IB_WQOS)..... 3–21
Master20 Read Quality of Service, SCB (SCB_MST20_RQOS)..... 3–58
Master 20 Write Quality of Service, SCB (SCB_MST20_WQOS)..... 3–59
Master 21 Read Quality of Service, SCB (SCB_MST21_RQOS)..... 3–60
Master 21 Write Quality of Service, SCB (SCB_MST21_WQOS)..... 3–61
Master 22 Read Quality of Service, SCB (SCB_MST22_RQOS)..... 3–62
Master 22 Write Quality of Service, SCB (SCB_MST22_WQOS)..... 3–63
Master 23 Read Quality of Service, SCB (SCB_MST23_RQOS)..... 3–64
Master 23 Write Quality of Service, SCB (SCB_MST23_WQOS)..... 3–65
Master 24 Read Quality of Service, SCB (SCB_MST24_RQOS)..... 3–66
Master 24 Write Quality of Service, SCB (SCB_MST24_WQOS)..... 3–67
Master 25 Read Quality of Service, SCB (SCB_MST25_RQOS)..... 3–68
Master 25 Write Quality of Service, SCB (SCB_MST25_WQOS)..... 3–69
Master 26 Read Quality of Service, SCB (SCB_MST26_RQOS)..... 3–70
Master 26 Write Quality of Service, SCB (SCB_MST26_WQOS)..... 3–71
Master 2 Read Quality of Service, SCB (SCB_MST02_RQOS)..... 3–22
Master 2 Write Quality of Service, SCB (SCB_MST02_WQOS)..... 3–23
Master 3 Read Quality of Service, SCB (SCB_MST03_RQOS)..... 3–24
Master 3 Write Quality of Service, SCB (SCB_MST03_WQOS)..... 3–25
Master 4 Read Quality of Service, SCB (SCB_MST04_RQOS)..... 3–26
Master 4 Write Quality of Service, SCB (SCB_MST04_WQOS)..... 3–27
Master 5 Read Quality of Service, SCB (SCB_MST05_RQOS)..... 3–28
Master 5 Write Quality of Service, SCB (SCB_MST05_WQOS)..... 3–29
Master 6 Read Quality of Service, SCB (SCB_MST06_RQOS)..... 3–30
Master 6 Write Quality of Service, SCB (SCB_MST06_WQOS)..... 3–31
Master 7 Read Quality of Service, SCB (SCB_MST07_RQOS)..... 3–32

Master 7 Write Quality of Service, SCB (SCB_MST07_WQOS)....3–33
Master 8 Read Quality of Service, SCB (SCB_MST08_RQOS)....3–34
Master 8 Write Quality of Service, SCB (SCB_MST08_WQOS)....3–35
Master 9 Read Quality of Service, SCB (SCB_MST09_RQOS)....3–36
Master 9 Write Quality of Service, SCB (SCB_MST09_WQOS)....3–37
Master Mode Address Register, TWI (TWI_MSTRADDR).. 20–28
Master Mode Control Registers, TWI (TWI_MSTRCTL).... 20–29
Master Mode Status Register, TWI (TWI_MSTRSTAT)..... 20–32
Master Trigger Register, TRU (TRU_MTR)....8–13
match block, SWU....32–6
Maximum Count Register, CNT (CNT_MAX)....17–29
maximum individual packet size (MaxPktSize). 22–23-22–26
MaxPktSize (maximum individual packet size). 22–23-22–26
MDB mode, UART....19–10,19–15,19–22
MDMA
configuration....12–14
definition....12–5
triggering....12–13
MDMA0_DST interrupt 7–6
MDMA0_SRC interrupt 7–6
MDMA1_DST interrupt 7–6
MDMA1_SRC interrupt 7–6
measurement report, GP timer....14–11
memory access, direct , *see* DMA
memory controller, static , *see* SMC
memory DMA (MDMA)....3–5
memory mapped mode, SPI....24–14
Memory Mapped Read Header , SPI (SPI_MMRDH) 24–61
memory mapped register (MMR)....3–6
memory scan mode, CRC....11–9,11–13,11–14,11–16
memory-to-memory DMA , *see* MDMA
memory transfer mode, CRC....11–9,11–19,11–20
MEMX Space Configuration Register, M4P (M4P_CACHE_MEMX)....2–42
MEMY Space Configuration Register, M4P (M4P_CACHE_MEMY)....2–44
Message Clear Bits Register, RCU (RCU_MSG_CLR) 30–16
Message Register, RCU (RCU_MSG)....30–14
Message Set Bits Register, RCU (RCU_MSG_SET).... 30–17
Minimum Count Register, CNT (CNT_MIN)....17–31
mirror options, CRC.... 11–6,11–7
MMC, EMAC....23–2,23–53
MMC Control Register, EMAC (EMAC_MMC_CTL)..... 23–149
MMC IPC Rx Interrupt Mask Register, EMAC (EMAC_IPC_RXIMSK)....23–130
MMC IPC Rx Interrupt Register, EMAC (EMAC_IPC_RXINT).... 23–136
MMC Rx Interrupt Mask Register, EMAC (EMAC_MMC_RXIMSK).... 23–151
MMC Rx Interrupt Register, EMAC (EMAC_MMC_RXINT)....23–154
MMC TX Interrupt Mask Register, EMAC (EMAC_MMC_TXIMSK).... 23–157
MMC Tx Interrupt Register, EMAC (EMAC_MMC_TXINT)....23–161
MMR
access bus....11–5
CRC....11–1
MMR access bus, DMA....12–15
MMR block, SWU....32–5
Modify 0 Register, DACC (DACC_MOD0)....27–44
Modify 1 Register, DACC (DACC_MOD1)....27–45
modulation, PWM.... 18–21
motor control, PWM....18–26,18–36
MPn Receive Function Address Register, USB (USB_MP[n]_RXFUNCADDR)....22–184
MPn Receive Hub Address Register, USB (USB_MP[n]_RXHUBADDR).... 22–185
MPn Receive Hub Port Register, USB (USB_MP[n]_RXHUBPORT).... 22–186
MPn Transmit Function Address Register, USB (USB_MP[n]_TXFUNCADDR)....22–187
MPn Transmit Hub Address Register, USB (USB_MP[n]_TXHUBADDR).... 22–188
MPn Transmit Hub Port Register, USB (USB_MP[n]_TXHUBPORT).... 22–189
multicast addressing, AFM, EMAC CORE.... 23–48
multichannel mode, SPORT....25–29
multi-drop bus mode, UART....19–10,19–15,19–22
multiplexer logic, *see* SPORT, SPMUX
multiplexing, PORT.... 13–2,13–6
multi-point configuration, USB....22–3,22–35,22–36
M Value for Divider, CNT (CNT_MDIV).... 17–30

N

negotiation protocol, host (HNP), USB.....22–63
nested vectored interrupt controller, SEC, *see* NVIC
noise filtering, CNT..... 17–13
N Value for Divider, CNT (CNT_NDIV)..... 17–32
NVIC
    features.....7–1
    non-maskable interrupt (NMI).....7–10

O

OCLK clock domain..... 3–5
OFS mode, EMAC DMA.....23–25
on-the-go (OTG), *see* OTG
On-time Capture Register, CPTMR (CPTMR_TON[n]).....
   15–20
open-drain mode, PORT.....13–6
oscillator watchdog, CGU..... 4–7
Oscillator Watchdog Register, CGU (CGU_OSCWDCTL).
   4–20
OSCW_EVT interrupt 7–2
OTG
    functional description, USB..... 22–2
    session request, USB.....22–1,22–62
OUTCLK frequency, CGU..... 4–10,4–11
Outer Loop Address Increment (2D only) Register, DMA (DMA_YMOD).....12–69
Outer Loop Count Start Value (2D only) Register, DMA (DMA_YCNT)..... 12–67
output clock (OCLK).....4–4
output mode, PORT..... 13–6
Overwrite Protection/Single Shot Transmission 1 Register, CAN (CAN_OPSS1)..... 21–70
Overwrite Protection/Single Shot Transmission 2 Register, CAN (CAN_OPSS2)..... 21–71
ownership (OWN), EMAC DMA..... 23–34

P

packet buffering, USB.....22–12
packet size, USB 22–23
packet structure, EMAC programming.....23–84,23–86
PADS_PCFG0 (Peripheral Configuration0 Register, PADS).
   13–99
page mode, SMC.....9–6,9–16
Pair 0 Secondary (Filter) History n Register, SINC (SINC_P0SEC_HIST[n]).....29–35
Pair 1 Secondary (Filter) History n Register, SINC (SINC_P1SEC_HIST[n]).....29–36
Pair 2 Secondary (Filter) History n Register, SINC (SINC_P2SEC_HIST[n]).....29–37
Pair 3 Secondary (Filter) History n Register, SINC (SINC_P3SEC_HIST[n]).....29–38
parity behavior, UART.....19–8
peer-to-peer (P2P) PTP message support..... 23–58
Pending Events Status Register, ADCC (ADCC_EPND).....
   26–60
perfect filtering mode, AFM, EMAC CORE..... 23–48
Peripheral Configuration0 Register, PADS (PADS_PCFG0).
   13–99
peripheral control, DMA..... 12–10,12–12
peripheral latency.....3–4
peripheral mode, USB..... 22–2,22–11,22–46
phase-locked loop, *see* PLL
phase-locked loop, CGU, *see* PLL
phase-locked loop clock (PLLCLK)..... 4–3,4–8
phase offset, PWM..... 18–9
PHY device, EMAC.....23–2,23–38,23–76,23–78
PHY interface, EMAC.....23–11,23–50
pin descriptions table, SPORT.....25–3
pin interrupt mode, *see* PININT mode
PININT mode, GP timer..... 14–5,14–18
pin multiplexing, PORT.....13–2,13–6
PinMux logic, *see* SPORT, SPMUX
PINT_ASSIGN (Pint Assign Register, PINT)..... 13–67
PINT_EDGE_CLR (Pint Edge Clear Register, PINT) 13–68
PINT_EDGE_SET (Pint Edge Set Register, PINT).... 13–71
PINT_INV_CLR (Pint Invert Clear Register, PINT).. 13–74
PINT_INV_SET (Pint Invert Set Register, PINT)..... 13–77
PINT_LATCH (Pint Latch Register, PINT).....13–81
PINT_MSK_CLR (Pint Mask Clear Register, PINT)..13–84
PINT_MSK_SET (Pint Mask Set Register, PINT)..... 13–87
PINT_PINSTATE (Pint Pinstate Register, PINT)..... 13–91
PINT_REQ (Pint Request Register, PINT)..... 13–95
PINT0_BLOCK interrupt 7–3,13–4
PINT0_BLOCK trigger 8–2,13–4
PINT1_BLOCK interrupt 7–3,13–4
PINT1_BLOCK trigger 8–2,13–4
PINT2_BLOCK interrupt 7–3,13–4
PINT2_BLOCK trigger 8–2,13–4
PINT3_BLOCK interrupt 7–3,13–4
PINT3_BLOCK trigger 8–2,13–4
PINT4_BLOCK interrupt 7–3,13–4

PINT4_BLOCK trigger 8–3,13–4
Pint Assign Register, PINT (PINT_ASSIGN)..... 13–67
Pint Edge Clear Register, PINT (PINT_EDGE_CLR) 13–68
Pint Edge Set Register, PINT (PINT_EDGE_SET).... 13–71
Pint Invert Clear Register, PINT (PINT_INV_CLR).. 13–74
Pint Invert Set Register, PINT (PINT_INV_SET)..... 13–77
Pint Latch Register, PINT (PINT_LATCH).....13–81
Pint Mask Clear Register, PINT (PINT_MSK_CLR)..13–84
Pint Mask Set Register, PINT (PINT_MSK_SET)..... 13–87
Pint Pinstate Register, PINT (PINT_PINSTATE)..... 13–91
Pint Request Register, PINT (PINT_REQ)..... 13–95
PINTx block, PORT..... 13–2,13–7,13–10
PLL
    block diagram..... 4–4
    frequency configuration..... 4–8
PLL bypass..... 4–4
PLLCLK frequency, CGU..... 4–11
PLL control unit (PCU)..... 4–3
Pointer to Next Initial Descriptor Register, DMA (DMA_DSCPTR_NXT)..... 12–59
Polynomial Register, CRC (CRC_POLY)..... 11–37
PORT
    architectural concepts..... 13–5
    event control..... 13–7
    features..... 13–1
    functional description..... 13–2
    GPIO functionality..... 13–2
    interface..... 13–5
    overview..... 13–1
    programming..... 13–9
PORT_DATA_CLR (Port x GPIO Data Clear Register, PORT)..... 13–17
PORT_DATA_SET (Port x GPIO Data Set Register, PORT)..... 13–20
PORT_DATA_TGL (Port x GPIO Output Toggle Register, PORT)..... 13–23
PORT_DATA (Port x GPIO Data Register, PORT).... 13–14
PORT_DIR_CLR (Port x GPIO Direction Clear Register, PORT)..... 13–30
PORT_DIR_SET (Port x GPIO Direction Set Register, PORT)..... 13–33
PORT_DIR (Port x GPIO Direction Register, PORT) 13–27
PORT_FER_CLR (Port x Function Enable Clear Register, PORT)..... 13–39
PORT_FER_SET (Port x Function Enable Set Register, PORT)..... 13–42
PORT_FER (Port x Function Enable Register, PORT) 13–36
PORT_INEN_CLR (Port x GPIO Input Enable Clear Register, PORT)..... 13–48
PORT_INEN_SET (Port x GPIO Input Enable Set Register, PORT)..... 13–51
PORT_INEN (Port x GPIO Input Enable Register, PORT). 13–45
PORT_LOCK (Port x GPIO Lock Register, PORT)... 13–54
PORT_MUX (Port x Multiplexer Control Register, PORT).. 13–56
PORT_POL_CLR (Port x GPIO Polarity Invert Clear Register, PORT)..... 13–61
PORT_POL_SET (Port x GPIO Polarity Invert Set Register, PORT)..... 13–64
PORT_POL (Port x GPIO Polarity Invert Register, PORT).. 13–58
Port x Function Enable Clear Register, PORT (PORT_FER_CLR)..... 13–39
Port x Function Enable Register, PORT (PORT_FER) 13–36
Port x Function Enable Set Register, PORT (PORT_FER_SET)..... 13–42
Port x GPIO Data Clear Register, PORT (PORT_DATA_CLR)..... 13–17
Port x GPIO Data Register, PORT (PORT_DATA).... 13–14
Port x GPIO Data Set Register, PORT (PORT_DATA_SET)..... 13–20
Port x GPIO Direction Clear Register, PORT (PORT_DIR_CLR)..... 13–30
Port x GPIO Direction Register, PORT (PORT_DIR) 13–27
Port x GPIO Direction Set Register, PORT (PORT_DIR_SET)..... 13–33
Port x GPIO Input Enable Clear Register, PORT (PORT_INEN_CLR)..... 13–48
Port x GPIO Input Enable Register, PORT (PORT_INEN). 13–45
Port x GPIO Input Enable Set Register, PORT (PORT_INEN_SET)..... 13–51
Port x GPIO Lock Register, PORT (PORT_LOCK)... 13–54
Port x GPIO Output Toggle Register, PORT (PORT_DATA_TGL)..... 13–23
Port x GPIO Polarity Invert Clear Register, PORT (PORT_POL_CLR)..... 13–61
Port x GPIO Polarity Invert Register, PORT (PORT_POL).. 13–58
Port x GPIO Polarity Invert Set Register, PORT (PORT_POL_SET)..... 13–64

Port x Multiplexer Control Register, PORT (PORT_MUX).. 13–56
Power and Device Control Register, USB (USB_POWER).... 22–192
power dissipation, DPM..... 6–4
power management unit, dynamic , *see* DPM
power modes, DPM..... 6–2
powersaving modes, CAN..... 21–16
PPS
command..... 23–73
event control..... 23–74
modes..... 23–71,23–83
output frequencies..... 23–71
system time..... 23–71
PPS Control Register, EMAC (EMAC_TM_PPSCTL)..... 23–231
PPS Width Register, EMAC (EMAC_TM_PPSWIDTH)..... 23–235
precision time protocol engine, *see* PTP
Previous Initial Descriptor Pointer Register, DMA (DMA_DSCPTR_PRV)..... 12–60
Primary (Filters) Head for Group 0 Register, SINC (SINC_PHEAD0)..... 29–39
Primary (Filters) Head for Group 1 Register, SINC (SINC_PHEAD1)..... 29–40
Primary (Filters) Pointer for Group 0 Register, SINC (SINC_PPTR0)..... 29–41
Primary (Filters) Pointer for Group 1 Register, SINC (SINC_PPTR1)..... 29–42
Primary (Filters) Tail for Group 0 Register, SINC (SINC_PTAIL0)..... 29–43
Primary (Filters) Tail for Group 1 Register, SINC (SINC_PTAIL1)..... 29–44
priority grouping, SEC..... 7–11
programming restrictions, SEC..... 7–16
propagation delay, PTP..... 23–58
protection, peripheral, SPU..... 5–6
protection unit, system , *see* SPU
PTP
block diagram..... 23–59
clock..... 23–60
frame detection..... 23–62,23–82
header structure..... 23–87
IEEE1588..... 23–55
messaging..... 23–63
processing..... 23–64
programming..... 23–81
pulse-per-second (PPS)..... 23–59
system time..... 23–68,23–71,23–81
timestamping..... 23–61,23–67,23–82
pulse mode, PWM..... 18–12
pulse-per-second (PPS), PTP, *see* PPS
pulse-width modulator , *see* PWM
push-button operation, CNT..... 17–10,17–13
PWM
architectural concepts..... 18–5
block diagram..... 18–5
crossover operations..... 18–25
event control..... 18–33
features..... 18–1
functional description..... 18–1
interface..... 18–36
output disable function..... 18–25
overview..... 18–1
programming..... 18–36
sync operation..... 18–24
timing unit..... 18–6,18–12
waveforms..... 18–31
PWM_ACTL (Channel A Control Register, PWM).... 18–44
PWM_AH_DUTY0 (Channel A-High Full Duty0 Register, PWM)..... 18–50
PWM_AH_DUTY1 (Channel A-High Full Duty1 Register, PWM)..... 18–51
PWM_AH0_HP (Channel A-High Heightened-Precision Duty-0 Register, PWM)..... 18–47
PWM_AH0 (Channel A-High Duty-0 Register, PWM)..... 18–46
PWM_AH1_HP (Channel A-High Heightened-Precision Duty-1 Register, PWM)..... 18–49
PWM_AH1 (Channel A-High Duty-1 Register, PWM)..... 18–48
PWM_AL_DUTY0 (Channel A-Low Full Duty0 Register, PWM)..... 18–56
PWM_AL_DUTY1 (Channel A-Low Full Duty1 Register, PWM)..... 18–57
PWM_AL0_HP (Channel A-Low Heightened-Precision Duty-0 Register, PWM)..... 18–53
PWM_AL0 (Channel A-Low Duty-0 Register, PWM) 18–52
PWM_AL1_HP (Channel A-Low Heightened-Precision Duty-1 Register, PWM)..... 18–55
PWM_AL1 (Channel A-Low Duty-1 Register, PWM) 18–54
PWM_BCTL (Channel B Control Register, PWM).... 18–58

PWM_BH_DUTY0 (Channel B-High Full Duty0 Register, PWM)....18–64
PWM_BH_DUTY1 (Channel B-High Full Duty1 Register, PWM)....18–65
PWM_BH0_HP (Channel B-High Heightened-Precision Duty-0 Register, PWM).... 18–61
PWM_BH0 (Channel B-High Duty-0 Register, PWM)..... 18–60
PWM_BH1_HP (Channel B-High Heightened-Precision Duty-1 Register, PWM).... 18–63
PWM_BH1 (Channel B-High Duty-1 Register, PWM)..... 18–62
PWM_BL_DUTY0 (Channel B-Low Full Duty0 Register, PWM)....18–70
PWM_BL_DUTY1 (Channel B-Low Full Duty1 Register, PWM)....18–71
PWM_BL0_HP (Channel B-Low Heightened-Precision Duty-0 Register, PWM)....18–67
PWM_BL0 (Channel B-Low Duty-0 Register, PWM) 18–66
PWM_BL1_HP (Channel B-Low Heightened-Precision Duty-1 Register, PWM)....18–69
PWM_BL1 (Channel B-Low Duty-1 Register, PWM) 18–68
PWM_CCTL (Channel C Control Register, PWM)... 18–72
PWM_CH_DUTY0 (Channel C-High Full Duty0 Register, PWM)....18–86
PWM_CH_DUTY1 (Channel C-High Full Duty1 Register, PWM)....18–87
PWM_CH0_HP (Channel C-High Pulse Heightened-Precision Duty Register 0, PWM)....18–75
PWM_CH0 (Channel C-High Pulse Duty Register 0, PWM)....18–74
PWM_CH1_HP (Channel C-High Pulse Heightened-Precision Duty Register 1, PWM)....18–77
PWM_CH1 (Channel C-High Pulse Duty Register 1, PWM)....18–76
PWM_CHANCFG (Channel Configuration Register, PWM)....18–78
PWM_CHOPCFG (Chop Configuration Register, PWM)... 18–85
PWM_CL_DUTY0 (Channel C-Low Full Duty0 Register, PWM)....18–92
PWM_CL_DUTY1 (Channel C-Low Full Duty1 Register, PWM)....18–93
PWM_CL0_HP (Channel C-Low Pulse Duty Register 1, PWM)....18–89
PWM_CL0 (Channel C-Low Pulse Duty Register 0, PWM). 18–88
PWM_CL1_HP (Channel C-Low Heightened-Precision Duty-1 Register, PWM).... 18–91
PWM_CL1 (Channel C-Low Duty-1 Register, PWM)18–90
PWM_CTL (Control Register, PWM).... 18–94
PWM_DCTL (Channel D Control Register, PWM)...18–98
PWM_DH_DUTY0 (Channel D-High Full Duty0 Register, PWM)....18–104
PWM_DH_DUTY1 (Channel D-High Full Duty1 Register, PWM)....18–105
PWM_DH0_HP (Channel D-High Pulse Heightened-Precision Duty Register 0, PWM).... 18–101
PWM_DH0 (Channel D-High Duty-0 Register, PWM)..... 18–100
PWM_DH1_HP (Channel D High Pulse Heightened-Precision Duty Register 1, PWM)....18–103
PWM_DH1 (Channel D-High Pulse Duty Register 1, PWM)....18–102
PWM_DL_DUTY0 (Channel D-Low Full Duty0 Register, PWM)....18–114
PWM_DL_DUTY1 (Channel D-Low Full Duty1 Register, PWM)....18–115
PWM_DL0_HP (Channel D-Low Heightened-Precision Duty-0 Register, PWM).... 18–107
PWM_DL0 (Channel D-Low Pulse Duty Register 0, PWM) 18–106
PWM_DL1_HP (Channel D-Low Heightened-Precision Duty-1 Register, PWM).... 18–109
PWM_DL1 (Channel D-Low Pulse Duty Register 1, PWM) 18–108
PWM_DLYA (Channel A Delay Register, PWM)..... 18–110
PWM_DLYB (Channel B Delay Register, PWM).....18–111
PWM_DLYC (Channel C Delay Register, PWM).....18–112
PWM_DLYD (Channel D Delay Register, PWM).... 18–113
PWM_DT (Dead Time Register, PWM).....18–116
PWM_ILAT (Interrupt Latch Register, PWM).....18–117
PWM_IMSK (Interrupt Mask Register, PWM).....18–119
PWM_STAT (Status Register, PWM).....18–121
PWM_SYNC_WID (Sync Pulse Width Register, PWM)..... 18–126
PWM_TM0 (Timer 0 Period Register, PWM)..... 18–127
PWM_TM1 (Timer 1 Period Register, PWM)..... 18–128
PWM_TM2 (Timer 2 Period Register, PWM)..... 18–129
PWM_TM3 (Timer 3 Period Register, PWM)..... 18–130
PWM_TM4 (Timer 4 Period Register, PWM)..... 18–131

PWM_TRIPCFG (Trip Configuration Register, PWM)..... 18–132
PWM0_SYNC interrupt 7–3,18–2
PWM0_SYNC trigger 8–3,18–2
PWM0_TRIP_TRIG0 trigger 8–6,18–2
PWM0_TRIP_TRIG1 trigger 8–6,18–2
PWM0_TRIP_TRIG2 trigger 8–6,18–2
PWM0_TRIP interrupt 7–3,18–2
PWM1_SYNC interrupt 7–3,18–3
PWM1_SYNC trigger 8–3,18–2
PWM1_TRIP_TRIG0 trigger 8–6,18–2
PWM1_TRIP_TRIG1 trigger 8–6,18–2
PWM1_TRIP_TRIG2 trigger 8–6,18–2
PWM1_TRIP interrupt 7–3,18–2
PWM2_SYNC interrupt 7–3,18–3
PWM2_SYNC trigger 8–3,18–2
PWM2_TRIP_TRIG0 trigger 8–6,18–2
PWM2_TRIP_TRIG1 trigger 8–6,18–2
PWM2_TRIP_TRIG2 trigger 8–6,18–2
PWM2_TRIP interrupt 7–3,18–2
PWMOUT mode, GP timer. 14–5,14–7,14–8,14–21,14–24

Q

quad mode, SPI..... 24–12
quadrature encoder mode, CNT..... 17–4,17–16
queues, descriptor, DMA..... 12–40,12–42

R

RAM Information Register, USB (USB_RAMINFO)22–195
Rate Control for Group 0 Register, SINC (SINC_RATE0)... 29–45
Rate Control for Group 1 Register, SINC (SINC_RATE1)... 29–47
RCOE, EMAC CORE..... 23–47
RCU
  architectural concepts..... 30–4
  core reset..... 30–5
  definitions..... 30–3
  event control..... 30–4
  features..... 30–1
  functional description..... 30–1
  overview..... 30–1
RCU_BCODE (Boot Code Register, RCU)..... 30–7
RCU_CRCTL (Core Reset outputs Control Register, RCU). 30–10
RCU_CRSTAT (Core Reset outputs Status Register, RCU)... 30–11
RCU_CTL (Control Register, RCU)..... 30–12
RCU_MSG_CLR (Message Clear Bits Register, RCU) 30–16
RCU_MSG_SET (Message Set Bits Register, RCU).... 30–17
RCU_MSG (Message Register, RCU)..... 30–14
RCU_REVID (Revision ID Register, RCU)..... 30–18
RCU_SRRQSTAT (System Reset Request Status, RCU)..... 30–19
RCU_STAT (Status Register, RCU)..... 30–20
RCU_SVECT_LCK (SVECT Lock Register, RCU).... 30–23
RCU_SVECT0 (Software Vector Register 0, RCU)..... 30–22
RCU0_SYSRST0 trigger 8–4,30–2
RCU0_SYSRST1 trigger 8–4,30–2
Receive Buffer Register, UART (UART_RBR)..... 19–40
receive checksum offload engine (RCOE), EMAC CORE..... 23–47
Receive Control Register, SPI (SPI_RXCTL)..... 24–69
Receive Counter Register, UART (UART_RXCNT)... 19–42
Received Word Count Register, SPI (SPI_RWC)..... 24–67
Received Word Count Reload Register, SPI (SPI_RWCR)..... 24–68
Receive FIFO Data Register, SPI (SPI_RFIFO)..... 24–66
receive frame controller module (RFC), EMAC CORE..... 23–45
Receive Interrupt Enable Register, USB (USB_INTRRXE)... 22–168
Receive Interrupt Register, USB (USB_INTRRX)..... 22–167
Receive Message Lost 1 Register, CAN (CAN_RML1) 21–74
Receive Message Lost 2 Register, CAN (CAN_RML2) 21–75
Receive Message Pending 1 Register, CAN (CAN_RMP1).... 21–76
Receive Message Pending 2 Register, CAN (CAN_RMP2).... 21–77
receive protocol engine module (RPE), EMAC CORE 23–44
Receive Shift Register, UART (UART_RSR)..... 19–41
redundancy check *, see* CRC
refresh operation, DMA..... 12–17
registers
  diagram conventions..... 1–4
Remote Frame Handling 1 Register, CAN (CAN_RFH1)..... 21–72
Remote Frame Handling 2 Register, CAN (CAN_RFH2)..... 21–73
repeated start mode, TWI..... 20–7,20–11
request data command, DMA..... 12–12

request data urgent command, DMA.....12–12
reset, USB.....22–45
reset control unit, *see* RCU
reset mode, DPM..... 6–3
resetting a core, RCU.....30–5
restart command, DMA.....12–10,12–12
Results RAM Register, HAE (HAE_RESULTS_RAM)28–29
retransmission, CAN..... 21–9
return-to-zero-inverted (RZI) modulation, UART.....19–9
Revision ID Register, RCU (RCU_REVID)..... 30–18
RFC, EMAC CORE..... 23–45
right-justified mode, SPORT.....25–27
ring mode, EMAC DMA.....23–18
RMII interface, EMAC.....23–2,23–11,23–12
rollover mode, PTP.....23–68,23–72
RPE, EMAC CORE.....23–44
Run Clear Register, CPTMR (CPTMR_RUN_CLR)..15–14
Run Clear Register, TIMER (TIMER_RUN_CLR).... 14–36
Run Control Message Clear Register, TAPC (TAPC_RCMSG_CLR).....33–6
Run Control Message Register, TAPC (TAPC_RCMSG)..... 33–4
Run Control Message Set Register, TAPC (TAPC_RCMSG_SET).....33–7
Run Control Message Toggle Register, TAPC (TAPC_RCMSG_TOG).....33–8
Run Register, CPTMR (CPTMR_RUN)..... 15–13
Run Register, HAE (HAE_RUN).....28–30
Run Register, TIMER (TIMER_RUN).....14–35
Run Set Register, CPTMR (CPTMR_RUN_SET).....15–15
Run Set Register, TIMER (TIMER_RUN_SET)..... 14–37
Rx 1024- to Max-Byte Frames (Good/Bad) Register, EMAC (EMAC_RX1024TOMAX_GB)..... 23–165
Rx 128- to 255-Byte Frames (Good/Bad) Register, EMAC (EMAC_RX128TO255_GB).....23–166
Rx 256- to 511-Byte Frames (Good/Bad) Register, EMAC (EMAC_RX256TO511_GB).....23–167
Rx 512- to 1023-Byte Frames (Good/Bad) Register, EMAC (EMAC_RX512TO1023_GB).....23–168
Rx 64-Byte Frames (Good/Bad) Register, EMAC (EMAC_RX64_GB).....23–169
Rx 65- to 127-Byte Frames (Good/Bad) Register, EMAC (EMAC_RX65TO127_GB).....23–170
Rx alignment Error Register, EMAC (EMAC_RXALIGN_ERR)..... 23–171
Rx Broadcast Frames (Good) Register, EMAC (EMAC_RXBCASTFRM_G)..... 23–172
Rx CRC Error Register, EMAC (EMAC_RXCRC_ERR)..... 23–173
Rx Data Double-Byte Register, TWI (TWI_RXDATA16)..... 20–35
Rx Data Single-Byte Register, TWI (TWI_RXDATA8)..... 20–36
RX Double Packet Buffer Disable for Endpoints 1 to 3, USB (USB_RXDPKTBUFDIS)..... 22–197
Rx FIFO Overflow Register, EMAC (EMAC_RXFIFO_OVF)..... 23–174
Rx Frame Count (Good/Bad) Register, EMAC (EMAC_RXFRMCNT_GB).....23–175
Rx ICMP Error Frames Register, EMAC (EMAC_RXICMP_ERR_FRM)..... 23–176
Rx ICMP Error Octets Register, EMAC (EMAC_RXICMP_ERR_OCT).....23–177
Rx ICMP Good Frames Register, EMAC (EMAC_RXICMP_GD_FRM)..... 23–178
Rx ICMP Good Octets Register, EMAC (EMAC_RXICMP_GD_OCT)..... 23–179
Rx IPv4 Datagrams (Good) Register, EMAC (EMAC_RXIPV4_GD_FRM)..... 23–182
Rx IPv4 Datagrams Fragmented Frames Register, EMAC (EMAC_RXIPV4_FRAG_FRM)..... 23–180
Rx IPv4 Datagrams Fragmented Octets Register, EMAC (EMAC_RXIPV4_FRAG_OCT).....23–181
Rx IPv4 Datagrams Good Octets Register, EMAC (EMAC_RXIPV4_GD_OCT).....23–183
Rx IPv4 Datagrams Header Errors Register, EMAC (EMAC_RXIPV4_HDR_ERR_FRM).....23–184
Rx IPv4 Datagrams Header Errors Register, EMAC (EMAC_RXIPV4_HDR_ERR_OCT).....23–185
Rx IPv4 Datagrams No Payload Frame Register, EMAC (EMAC_RXIPV4_NOPAY_FRM)..... 23–186
Rx IPv4 Datagrams No Payload Octets Register, EMAC (EMAC_RXIPV4_NOPAY_OCT)..... 23–187
Rx IPv4 UDP Disabled Frames Register, EMAC (EMAC_RXIPV4_UDSBL_FRM).....23–188
Rx IPv4 UDP Disabled Octets Register, EMAC (EMAC_RXIPV4_UDSBL_OCT)..... 23–189
Rx IPv6 Datagrams Good Frames Register, EMAC (EMAC_RXIPV6_GD_FRM)..... 23–190
Rx IPv6 Datagrams Header Error Frames Register, EMAC (EMAC_RXIPV6_HDR_ERR_FRM).....23–192

Rx IPv6 Datagrams No Payload Frames Register, EMAC (EMAC_RXIPV6_NOPAY_FRM)..... 23–194
Rx IPv6 Good Octets Register, EMAC (EMAC_RXIPV6_GD_OCT).....23–191
Rx IPv6 Header Errors Register, EMAC (EMAC_RXIPV6_HDR_ERR_OCT).....23–193
Rx IPv6 No Payload Octets Register, EMAC (EMAC_RXIPV6_NOPAY_OCT)..... 23–195
Rx Jab Error Register, EMAC (EMAC_RXJAB_ERR)..... 23–196
Rx Length Error Register, EMAC (EMAC_RXLEN_ERR)... 23–197
Rx Multicast Frames (Good) Register, EMAC (EMAC_RXMCASTFRM_G)..... 23–198
Rx Octet Count (Good/Bad) Register, EMAC (EMAC_RXOCTCNT_GB).....23–200
Rx Octet Count (Good) Register, EMAC (EMAC_RXOCTCNT_G).....23–199
Rx Out Of Range Type Register, EMAC (EMAC_RXOOR-TYPE)..... 23–201
Rx Oversize (Good) Register, EMAC (EMAC_RXO-SIZE_G)..... 23–202
Rx Pause Frames Register, EMAC (EMAC_RXPAUSEFRM) 23–203
Rx Runt Error Register, EMAC (EMAC_RXRUNT_ERR)... 23–204
Rx TCP Error Frames Register, EMAC (EMAC_RXTCP_ERR_FRM).....23–205
Rx TCP Error Octets Register, EMAC (EMAC_RXTCP_ERR_OCT)..... 23–206
Rx TCP Good Frames Register, EMAC (EMAC_RXTCP_GD_FRM).....23–207
Rx TCP Good Octets Register, EMAC (EMAC_RXTCP_GD_OCT).....23–208
Rx UDP Error Frames Register, EMAC (EMAC_RXUDP_ERR_FRM).....23–210
Rx UDP Error Octets Register, EMAC (EMAC_RXUDP_ERR_OCT)..... 23–211
Rx UDP Good Frames Register, EMAC (EMAC_RXUDP_GD_FRM)..... 23–212
Rx UDP Good Octets Register, EMAC (EMAC_RXUDP_GD_OCT).....23–213
Rx Undersize (Good) Register, EMAC (EMAC_RXU-SIZE_G)..... 23–214
Rx Unicast Frames (Good) Register, EMAC (EMAC_RXU-CASTFRM_G)..... 23–209
Rx VLAN Frames (Good/Bad) Register, EMAC (EMAC_RXVLANFRM_GB).....23–215
Rx Watch Dog Error Register, EMAC (EMAC_RXWDOG_ERR)..... 23–216
RZI modulation, UART..... 19–9

S

scan mode, CRC..... 11–9
SCB
architectural concepts..... 3–2
block diagram..... 3–2
data transfer.....23–13
DMA.....23–14,23–16
DMA interface..... 12–7,12–8
EMAC interface..... 23–13
features..... 3–1
functional description..... 3–1
overview..... 3–1
programming.....23–14
SCB_MST00_IB_RQOS (Master 0 Read Quality of Service, SCB)..... 3–16
SCB_MST00_IB_SYNC (Master 0 IB Sync Mode, SCB)..... 3–17
SCB_MST00_IB_WQOS (Master 0 Write Quality of Service, SCB)..... 3–18
SCB_MST01_IB_RQOS (Master 1 Read Quality of Service, SCB)..... 3–19
SCB_MST01_IB_SYNC (Master 1 IB Sync Mode, SCB)..... 3–20
SCB_MST01_IB_WQOS (Master 1 Write Quality of Service, SCB)..... 3–21
SCB_MST02_RQOS (Master 2 Read Quality of Service, SCB)..... 3–22
SCB_MST02_WQOS (Master 2 Write Quality of Service, SCB)..... 3–23
SCB_MST03_RQOS (Master 3 Read Quality of Service, SCB)..... 3–24
SCB_MST03_WQOS (Master 3 Write Quality of Service, SCB)..... 3–25
SCB_MST04_RQOS (Master 4 Read Quality of Service, SCB)..... 3–26
SCB_MST04_WQOS (Master 4 Write Quality of Service, SCB)..... 3–27
SCB_MST05_RQOS (Master 5 Read Quality of Service, SCB)..... 3–28

SCB_MST05_WQOS (Master 5 Write Quality of Service, SCB)..... 3–29
SCB_MST06_RQOS (Master 6 Read Quality of Service, SCB)..... 3–30
SCB_MST06_WQOS (Master 6 Write Quality of Service, SCB)..... 3–31
SCB_MST07_RQOS (Master 7 Read Quality of Service, SCB)..... 3–32
SCB_MST07_WQOS (Master 7 Write Quality of Service, SCB)..... 3–33
SCB_MST08_RQOS (Master 8 Read Quality of Service, SCB)..... 3–34
SCB_MST08_WQOS (Master 8 Write Quality of Service, SCB)..... 3–35
SCB_MST09_RQOS (Master 9 Read Quality of Service, SCB)..... 3–36
SCB_MST09_WQOS (Master 9 Write Quality of Service, SCB)..... 3–37
SCB_MST10_RQOS (Master 10 Read Quality of Service, SCB)..... 3–38
SCB_MST10_WQOS (Master 10 Write Quality of Service, SCB)..... 3–39
SCB_MST11_RQOS (Master 11 Read Quality of Service, SCB)..... 3–40
SCB_MST11_WQOS (Master 11 Write Quality of Service, SCB)..... 3–41
SCB_MST12_RQOS (Master 12 Read Quality of Service, SCB)..... 3–42
SCB_MST12_WQOS (Master 12 Write Quality of Service, SCB)..... 3–43
SCB_MST13_RQOS (Master 13 Read Quality of Service, SCB)..... 3–44
SCB_MST13_WQOS (Master 13 Write Quality of Service, SCB)..... 3–45
SCB_MST14_RQOS (Master 14 Read Quality of Service, SCB)..... 3–46
SCB_MST14_WQOS (Master 14 Write Quality of Service, SCB)..... 3–47
SCB_MST15_RQOS (Master 15 Read Quality of Service, SCB)..... 3–48
SCB_MST15_WQOS (Master 15 Write Quality of Service, SCB)..... 3–49
SCB_MST16_RQOS (Master 16 Read Quality of Service, SCB)..... 3–50
SCB_MST16_WQOS (Master 16 Write Quality of Service, SCB)..... 3–51
SCB_MST17_RQOS (Master 17 Read Quality of Service, SCB)..... 3–52
SCB_MST17_WQOS (Master 17 Write Quality of Service, SCB)..... 3–53
SCB_MST18_RQOS (Master 18 Read Quality of Service, SCB)..... 3–54
SCB_MST18_WQOS (Master 18 Write Quality of Service, SCB)..... 3–55
SCB_MST19_RQOS (Master 19 Read Quality of Service, SCB)..... 3–56
SCB_MST19_WQOS (Master 19 Write Quality of Service, SCB)..... 3–57
SCB_MST20_RQOS (Master20 Read Quality of Service, SCB)..... 3–58
SCB_MST20_WQOS (Master 20 Write Quality of Service, SCB)..... 3–59
SCB_MST21_RQOS (Master 21 Read Quality of Service, SCB)..... 3–60
SCB_MST21_WQOS (Master 21 Write Quality of Service, SCB)..... 3–61
SCB_MST22_RQOS (Master 22 Read Quality of Service, SCB)..... 3–62
SCB_MST22_WQOS (Master 22 Write Quality of Service, SCB)..... 3–63
SCB_MST23_RQOS (Master 23 Read Quality of Service, SCB)..... 3–64
SCB_MST23_WQOS (Master 23 Write Quality of Service, SCB)..... 3–65
SCB_MST24_RQOS (Master 24 Read Quality of Service, SCB)..... 3–66
SCB_MST24_WQOS (Master 24 Write Quality of Service, SCB)..... 3–67
SCB_MST25_RQOS (Master 25 Read Quality of Service, SCB)..... 3–68
SCB_MST25_WQOS (Master 25 Write Quality of Service, SCB)..... 3–69
SCB_MST26_RQOS (Master 26 Read Quality of Service, SCB)..... 3–70
SCB_MST26_WQOS (Master 26 Write Quality of Service, SCB)..... 3–71
SCB interface, SWU.....32–4
SCB interfaces..... 3–5
SCB latency.....3–4
SCL Clock Divider Register, TWI (TWI_CLKDIV)...20–17
SCLK clock domain..... 3–5
Scratch Register, UART (UART_SCR).....19–43

SDBGKEY0 Register, TAPC (TAPC_SDBGKEY0)..... 33–9
SDBGKEY1 Register, TAPC (TAPC_SDBGKEY1).... 33–10
SDBGKEY2 Register, TAPC (TAPC_SDBGKEY2).... 33–11
SDBGKEY3 Register, TAPC (TAPC_SDBGKEY3).... 33–12
SDBGKEY Control Register, TAPC (TAPC_SDBGKEY_CTL)..... 33–13
SDBGKEY Status Register, TAPC (TAPC_SDBGKEY_STAT)..... 33–14
SEC
architectural concepts..... 7–10
block diagram..... 7–8
definitions..... 7–8
features..... 7–1
functional description..... 7–1
overview..... 7–1
programming..... 7–14
SEC_FCOPP_CUR (Fault COP Period Current Register, SEC)..... 7–19
SEC_FCOPP (Fault COP Period Register, SEC)..... 7–18
SEC_FCTL (Fault Control Register, SEC)..... 7–20
SEC_FDLY_CUR (Fault Delay Current Register, SEC) 7–24
SEC_FDLY (Fault Delay Register, SEC)..... 7–23
SEC_FEND (Fault End Register, SEC)..... 7–25
SEC_FSID (Fault Source ID Register, SEC)..... 7–26
SEC_FSRDLY_CUR (Fault System Reset Delay Current Register, SEC)..... 7–28
SEC_FSRDLY (Fault System Reset Delay Register, SEC)..... 7–27
SEC_FSTAT (Fault Status Register, SEC)..... 7–29
SEC_GCTL (Global Control Register, SEC)..... 7–31
SEC_GSTAT (Global Status Register, SEC)..... 7–32
SEC_RAISE (Global Raise Register, SEC)..... 7–34
SEC_SCTL[n] (Source Control Register n, SEC)..... 7–35
SEC_SSTAT[n] (Source Status Register n, SEC)..... 7–37
SEC0_ERR interrupt 7–3
SEC0_FAULT trigger 7–8,8–4
serial bus, *see* USB
serial clock signal, SPORT..... 25–3,25–11
serial peripheral interface, *see* SPI
serial peripheral interface (SPI)..... 3–5
serial port, *see* SPORT
serial port (SPORT)..... 3–5
SFI
block diagram..... 7–9
definition..... 7–8
SID value, SEC..... 7–8
SINC_BIAS0 (Bias for Group 0 Register, SINC)..... 29–18
SINC_BIAS1 (Bias for Group 1 Register, SINC)..... 29–19
SINC_CLK (Clock Control Register, SINC)..... 29–20
SINC_CTL (Control Register, SINC)..... 29–22
SINC_HIS_STAT (History Status Register, SINC)..... 29–25
SINC_LEVEL0 (Level Control for Group 0 Register, SINC)..... 29–27
SINC_LEVEL1 (Level Control for Group 1 Register, SINC)..... 29–29
SINC_LIMIT0 ((Amplitude) Limits for Secondary Filter 0 Register, SINC)..... 29–31
SINC_LIMIT1 ((Amplitude) Limits for Secondary Filter 1 Register, SINC)..... 29–32
SINC_LIMIT2 ((Amplitude) Limits for Secondary Filter 2 Register, SINC)..... 29–33
SINC_LIMIT3 ((Amplitude) Limits for Secondary Filter 3 Register, SINC)..... 29–34
SINC_P0SEC_HIST[n] (Pair 0 Secondary (Filter) History n Register, SINC)..... 29–35
SINC_P1SEC_HIST[n] (Pair 1 Secondary (Filter) History n Register, SINC)..... 29–36
SINC_P2SEC_HIST[n] (Pair 2 Secondary (Filter) History n Register, SINC)..... 29–37
SINC_P3SEC_HIST[n] (Pair 3 Secondary (Filter) History n Register, SINC)..... 29–38
SINC_PHEAD0 (Primary (Filters) Head for Group 0 Register, SINC)..... 29–39
SINC_PHEAD1 (Primary (Filters) Head for Group 1 Register, SINC)..... 29–40
SINC_PPTR0 (Primary (Filters) Pointer for Group 0 Register, SINC)..... 29–41
SINC_PPTR1 (Primary (Filters) Pointer for Group 1 Register, SINC)..... 29–42
SINC_PTAIL0 (Primary (Filters) Tail for Group 0 Register, SINC)..... 29–43
SINC_PTAIL1 (Primary (Filters) Tail for Group 1 Register, SINC)..... 29–44
SINC_RATE0 (Rate Control for Group 0 Register, SINC)..... 29–45
SINC_RATE1 (Rate Control for Group 1 Register, SINC)..... 29–47
SINC_STAT (Status Register, SINC)..... 29–49
SINC0_DATA0 trigger 8–3,29–4
SINC0_DATA1 trigger 8–3,29–4
SINC0_P0_OVLD trigger 8–3,29–3
SINC0_P1_OVLD trigger 8–3,29–3

SINC0_P2_OVLD trigger 8–3,29–3
SINC0_P3_OVLD trigger 8–3,29–3
SINC0_STAT interrupt 7–5,29–3
SINC0_SYNC0 trigger 8–7,29–4
SINC0_SYNC1 trigger 8–7,29–4
single-shot transmission, CAN.....21–10
Slave Mode Address Register, TWI (TWI_SLVADDR).....
.....20–37
Slave Mode Control Register, TWI (TWI_SLVCTL).. 20–38
Slave Mode Status Register, TWI (TWI_SLVSTAT)....20–40
slave select, SPI..... 24–9
Slave Select Register, SPI (SPI_SLVSEL).....24–72
Slave Select Register, TRU (TRU_SSR[n])..... 8–14
sleep mode, DPM.....6–3,6–4
SMC
architectural concepts..... 9–3
definitions..... 9–1
event control.....9–6
features..... 9–1
functional description.....9–3
operating modes..... 9–5
overview..... 9–1
programming.....9–6,9–16
SMC_B0CTL (Bank 0 Control Register, SMC)..... 9–18
SMC_B0ETIM (Bank 0 Extended Timing Register, SMC)...
.....9–20
SMC_B0TIM (Bank 0 Timing Register, SMC).....9–22
SMC_B1CTL (Bank 1 Control Register, SMC)..... 9–24
SMC_B1ETIM (Bank 1 Extended Timing Register, SMC)...
.....9–26
SMC_B1TIM (Bank 1 Timing Register, SMC).....9–28
SMC_B2CTL (Bank 2 Control Register, SMC)..... 9–30
SMC_B2ETIM (Bank 2 Extended Timing Register, SMC)...
.....9–32
SMC_B2TIM (Bank 2 Timing Register, SMC).....9–34
SMC_B3CTL (Bank 3 Control Register, SMC)..... 9–36
SMC_B3ETIM (Bank 3 Extended Timing Register, SMC)...
.....9–38
SMC_B3TIM (Bank 3 Timing Register, SMC).....9–40
SMI, EMAC.....23–2,23–50,23–52
SMI Address Register, EMAC (EMAC_SMI_ADDR).....
.....23–217
SMI Data Register, EMAC (EMAC_SMI_DATA).... 23–219
SOF packet, USB..... 22–27
SOFT0 interrupt 7–7
SOFT1 interrupt 7–7
SOFT2 interrupt 7–7
SOFT3 interrupt 7–7
soft reset, CAN.....21–17
Software Reset Register, USB (USB_SOFT_RST)..... 22–198
Software Vector Register 0, RCU (RCU_SVECT0)..... 30–22
Source Control Register n, SEC (SEC_SCTL[n]).....7–35
source interface (SSI), SEC, *see* SSI
Source Status Register n, SEC (SEC_SSTAT[n])..... 7–37
SPI
event control.....24–32
features..... 24–1
functional description.....24–2
overview..... 24–1
programming.....24–34
slave select..... 24–9
transfers.....24–5,24–10
SPI_CLK (Clock Rate Register, SPI).....24–40
SPI_CTL (Control Register, SPI).....24–41
SPI_DLY (Delay Register, SPI)..... 24–47
SPI_ILAT_CLR (Masked Interrupt Clear Register, SPI).....
.....24–50
SPI_ILAT (Masked Interrupt Condition Register, SPI)24–48
SPI_IMSK_CLR (Interrupt Mask Clear Register, SPI) 24–55
SPI_IMSK_SET (Interrupt Mask Set Register, SPI).....24–58
SPI_IMSK (Interrupt Mask Register, SPI)..... 24–53
SPI_MMRDH (Memory Mapped Read Header , SPI) 24–61
SPI_MMTOP (SPI Memory Top Address , SPI)..... 24–65
SPI_RFIFO (Receive FIFO Data Register, SPI)..... 24–66
SPI_RWC (Received Word Count Register, SPI)..... 24–67
SPI_RWCR (Received Word Count Reload Register, SPI).....
.....24–68
SPI_RXCTL (Receive Control Register, SPI).....24–69
SPI_SLVSEL (Slave Select Register, SPI).....24–72
SPI_STAT (Status Register, SPI).....24–75
SPI_TFIFO (Transmit FIFO Data Register, SPI).....24–80
SPI_TWC (Transmitted Word Count Register, SPI)....24–81
SPI_TWCR (Transmitted Word Count Reload Register, SPI)..... 24–82
SPI_TXCTL (Transmit Control Register, SPI)..... 24–83
SPI0_ERR interrupt 7–4,24–3
SPI0_RXDMA_ERR interrupt 7–3,24–3
SPI0_RXDMA interrupt 7–5,24–4
SPI0_RXDMA trigger 8–3,8–5
SPI0_STAT interrupt 7–5,24–4
SPI0_TXDMA_ERR interrupt 7–3,24–3
SPI0_TXDMA interrupt 7–5,24–4

SPI0_TXDMA trigger 8–3,8–5
SPI1_ERR interrupt 7–4,24–3
SPI1_RXDMA_ERR interrupt 7–4,24–3
SPI1_RXDMA interrupt 7–5,24–4
SPI1_RXDMA trigger 8–3,8–5
SPI1_STAT interrupt 7–5,24–4
SPI1_TXDMA_ERR interrupt 7–4,24–3
SPI1_TXDMA interrupt 7–5,24–4
SPI1_TXDMA trigger 8–3,8–5
SPI2_ERR interrupt 7–4,24–3
SPI2_RXDMA interrupt 7–6,24–4
SPI2_STAT interrupt 7–6,24–4
SPI2_TXDMA interrupt 7–6,24–4
SPI Memory Top Address , SPI (SPI_MMTOP)..... 24–65
SPI modes
DMA..... 24–36
dual I/O 24–12
memory mapped..... 24–14
non-DMA..... 24–10,24–11,24–34
quad..... 24–12
SPORT
architectural concepts..... 25–9
block diagram..... 25–8
data types and companding..... 25–13,25–33
event control..... 25–41
features..... 25–1
frame sync options..... 25–22,25–31
functional description..... 25–6
gated clock mode..... 25–36
operating modes..... 25–19
overview..... 25–1
pin descriptions table..... 25–3
programming..... 25–42
signal descriptions..... 25–3,25–31
SPMUX..... 25–11
SPORT_CS0_A (Half SPORT 'A' Multi-channel 0-31 Select Register, SPORT)..... 25–48
SPORT_CS0_B (Half SPORT 'B' Multi-channel 0-31 Select Register, SPORT)..... 25–49
SPORT_CS1_A (Half SPORT 'A' Multi-channel 32-63 Select Register, SPORT)..... 25–50
SPORT_CS1_B (Half SPORT 'B' Multi-channel 32-63 Select Register, SPORT)..... 25–51
SPORT_CS2_A (Half SPORT 'A' Multi-channel 64-95 Select Register, SPORT)..... 25–52
SPORT_CS2_B (Half SPORT 'B' Multichannel 64-95 Select Register, SPORT)..... 25–53
SPORT_CS3_A (Half SPORT 'A' Multi-channel 96-127 Select Register, SPORT)..... 25–54
SPORT_CS3_B (Half SPORT 'B' Multichannel 96-127 Select Register, SPORT)..... 25–55
SPORT_CTL_A (Half SPORT 'A' Control Register, SPORT)..... 25–58
SPORT_CTL_B (Half SPORT 'B' Control Register, SPORT)..... 25–66
SPORT_CTL2_A (Half SPORT 'A' Control 2 Register, SPORT)..... 25–56
SPORT_CTL2_B (Half SPORT 'B' Control 2 Register, SPORT)..... 25–57
SPORT_DIV_A (Half SPORT 'A' Divisor Register, SPORT)..... 25–75
SPORT_DIV_B (Half SPORT 'B' Divisor Register, SPORT)..... 25–76
SPORT_ERR_A (Half SPORT 'A' Error Register, SPORT).. 25–77
SPORT_ERR_B (Half SPORT 'B' Error Register, SPORT).. 25–79
SPORT_MCTL_A (Half SPORT 'A' Multi-channel Control Register, SPORT)..... 25–81
SPORT_MCTL_B (Half SPORT 'B' Multi-channel Control Register, SPORT)..... 25–83
SPORT_MSTAT_A (Half SPORT 'A' Multi-channel Status Register, SPORT)..... 25–85
SPORT_MSTAT_B (Half SPORT 'B' Multi-channel Status Register, SPORT)..... 25–86
SPORT_RXPRI_A (Half SPORT 'A' Rx Buffer (Primary) Register, SPORT)..... 25–87
SPORT_RXPRI_B (Half SPORT 'B' Rx Buffer (Primary) Register, SPORT)..... 25–88
SPORT_RXSEC_A (Half SPORT 'A' Rx Buffer (Secondary) Register, SPORT)..... 25–89
SPORT_RXSEC_B (Half SPORT 'B' Rx Buffer (Secondary) Register, SPORT)..... 25–90
SPORT_TXPRI_A (Half SPORT 'A' Tx Buffer (Primary) Register, SPORT)..... 25–91
SPORT_TXPRI_B (Half SPORT 'B' Tx Buffer (Primary) Register, SPORT)..... 25–92
SPORT_TXSEC_A (Half SPORT 'A' Tx Buffer (Secondary) Register, SPORT)..... 25–93
SPORT_TXSEC_B (Half SPORT 'B' Tx Buffer (Secondary) Register, SPORT)..... 25–94

SPORT0_A_DMA_ERR interrupt 7–3,25–7
SPORT0_A_DMA interrupt 7–5,25–7
SPORT0_A_DMA trigger 8–3,8–5
SPORT0_A_STAT interrupt 7–5,25–7
SPORT0_B_DMA_ERR interrupt 7–3,25–7
SPORT0_B_DMA interrupt 7–5,25–7
SPORT0_B_DMA trigger 8–3,8–5
SPORT0_B_STAT interrupt 7–5,25–7
SPORT1_A_DMA_ERR interrupt 7–3,25–7
SPORT1_A_DMA interrupt 7–5,25–8
SPORT1_A_DMA trigger 8–3,8–5
SPORT1_A_STAT interrupt 7–5,25–7
SPORT1_B_DMA_ERR interrupt 7–4,25–7
SPORT1_B_DMA interrupt 7–5,25–8
SPORT1_B_DMA trigger 8–3,8–5
SPORT1_B_STAT interrupt 7–5,25–8
SPORT modes
I2S mode.....25–25
left-justified mode..... 25–26
multichannel mode.....25–29
packed mode..... 25–34
right-justified mode.....25–27
selection, table.....25–21
standard serial mode.....25–21
stereo modes.....25–24
SPU
architectural concepts..... 5–4
block diagram.....5–4
event control.....5–4
features..... 5–1
functional description.....5–1
mode configuration..... 5–5
overview..... 5–1
programming.....5–4
SPU_CTL (Control Register, SPU)..... 5–7
SPU_STAT (Status Register, SPU).....5–8
SPU_WP[n] (Write Protect Register n, SPU)..... 5–9
SRAM Configuration Register, M4P (M4P_SRAM_CFG)...
.....2–47
SRAM Parity Error Address (Core) Register, M4P (M4P_SRAM_PEADDR_CORE).....2–49
SRAM Parity Error Address (DMA) Register, M4P (M4P_SRAM_PEADDR_DMA).....2–51
SRAM reads and writes, SMC..... 9–6,9–8,9–9
SSI
definition.....7–8
Start Address of Current Buffer Register, DMA (DMA_ADDRSTART).....12–44
startup operation, DMA..... 12–15,12–16
static memory controller (SMC)..... 3–5
static memory controller unit, *see* SMC
Status Information Register, TRU (TRU_STAT).....8–15
Status Interrupt Latch Register, TIMER (TIMER_STAT_ILAT).....14–38
Status Interrupt Mask Clear Register, CPTMR (CPTMR_STAT_IMSK_CLR)..... 15–18
Status Interrupt Mask Register, CPTMR (CPTMR_STAT_IMSK).....15–17
Status Interrupt Mask Register, TIMER (TIMER_STAT_IMSK).....14–39
Status Interrupt Mask Set Register, CPTMR (CPTMR_STAT_IMSK_SET).....15–19
Status Register, CAN (CAN_STAT)..... 21–78
Status Register, CGU (CGU_STAT)..... 4–22
Status Register, CNT (CNT_STAT).....17–33
Status Register, CRC (CRC_STAT).....11–40
Status Register, DACC (DACC_STAT).....27–46
Status Register, DMA (DMA_STAT)..... 12–61
Status Register, DPM (DPM_STAT)..... 6–15
Status Register, HAE (HAE_STAT).....28–31
Status Register, PWM (PWM_STAT).....18–121
Status Register, RCU (RCU_STAT)..... 30–20
Status Register, SINC (SINC_STAT)..... 29–49
Status Register, SPI (SPI_STAT).....24–75
Status Register, SPU (SPU_STAT).....5–8
Status Register, UART (UART_STAT).....19–44
stereo modes, SPORT.....25–24
Stop Configuration Clear Register, TIMER (TIMER_STOP_CFG_CLR)..... 14–41
Stop Configuration Register, TIMER (TIMER_STOP_CFG).....14–40
Stop Configuration Set Register, TIMER (TIMER_STOP_CFG_SET).....14–42
stop flow mode, DMA..... 12–20,12–25,12–34,12–37
store-and-forward mode, EMAC FIFO..23–36,23–37,23–40
STX, EMAC CORE.....23–43
subblocks, EMAC.....23–10
suspend mode, CAN.....21–17
suspend mode, USB operating modes..... 22–27,22–46
SVECT Lock Register, RCU (RCU_SVECT_LCK)....30–23
SWU
architectural concepts..... 32–3

block diagram....32–4
definitions.... 32–3
event control....32–6
features....32–1
flow diagram....32–3
functional description....32–1
operating modes.... 32–5
programming....32–7
SWU_CNT[n] (Count Register n, SWU).... 32–10
SWU_CTL[n] (Control Register n, SWU)....32–11
SWU_CUR[n] (Current Register n, SWU).... 32–16
SWU_GCTL (Global Control Register, SWU)..... 32–17
SWU_GSTAT (Global Status Register, SWU).....32–18
SWU_HIST[n] (Bandwidth History Register n, SWU).....
.....32–22
SWU_ID[n] (ID Register n, SWU).....32–23
SWU_LA[n] (Lower Address Register n, SWU)..... 32–24
SWU_TARG[n] (Target Register n, SWU).....32–25
SWU_UA[n] (Upper Address Register n, SWU).....32–26
SWU0_EN trigger 8–6,32–2
SWU0_EVT interrupt 7–7,32–2
SWU0_EVT trigger 8–4,32–2
SWU1_EN trigger 8–6,32–3
SWU1_EVT interrupt 7–7,32–2
SWU1_EVT trigger 8–4,32–2
SWU2_EN trigger 8–6,32–3
SWU2_EVT interrupt 7–7,32–2
SWU2_EVT trigger 8–4,32–2
SWU3_EN trigger 8–6,32–3
SWU3_EVT interrupt 7–7,32–2
SWU3_EVT trigger 8–4,32–2
SWU4_EN trigger 8–6,32–3
SWU4_EVT interrupt 7–7,32–2
SWU4_EVT trigger 8–4,32–2
synchronization, DMA..... 12–39,12–40
synchronization, FIFO.....3–13
sync operation, PWM.....18–24
Sync Pulse Width Register, PWM (PWM_SYNC_WID).....
.....18–126
SYS_ECT_EVT0 interrupt 7–3
SYS_ECT_EVT1 interrupt 7–3
SYS_MDMA0_DST_ERR interrupt 7–4
SYS_MDMA0_SRC_ERR interrupt 7–4
SYS_MDMA1_DST_ERR interrupt 7–4
SYS_MDMA1_SRC_ERR interrupt 7–4
SYSCLK clock domain..... 3–5
SYSCLK frequency, CGU.....4–9,4–11
system bus interconnection, *see* SCB
system clock (SYSCLK)..... 4–3,4–4,4–9
System Clock Buffer Disable Register, DPM (DPM_SCBF_DIS).....6–14
system clock input (SYS_CLKIN)..... 4–4
system clock output (SYS_CLKOUT).....4–4
system crossbar block (SCB)..... 32–4,32–5
system crossbar interface, *see* SCB
system crossbars unit, *see* SCB
system event controller, *see* SEC
system peripherals clock (SCLKn).....4–4,4–9
system reset, RCU.....30–5
System Reset Request Status, RCU (RCU_SRRQSTAT).....
.....30–19
system watchpoint unit, *see* SWU
SysTick Calibration Register, M4P (M4P_STCALIB)... 2–53

T

TAPC_IDCODE (IDCODE Register, TAPC).....33–3
TAPC_RCMSG_CLR (Run Control Message Clear Register, TAPC).....33–6
TAPC_RCMSG_SET (Run Control Message Set Register, TAPC).....33–7
TAPC_RCMSG_TOG (Run Control Message Toggle Register, TAPC).....33–8
TAPC_RCMSG (Run Control Message Register, TAPC).....
.....33–4
TAPC_SDBGKEY_CTL (SDBGKEY Control Register, TAPC).....33–13
TAPC_SDBGKEY_STAT (SDBGKEY Status Register, TAPC).....33–14
TAPC_SDBGKEY0 (SDBGKEY0 Register, TAPC)..... 33–9
TAPC_SDBGKEY1 (SDBGKEY1 Register, TAPC).... 33–10
TAPC_SDBGKEY2 (SDBGKEY2 Register, TAPC).... 33–11
TAPC_SDBGKEY3 (SDBGKEY3 Register, TAPC).... 33–12
TAPC_USERCODE (USERCODE Register, TAPC). 33–15
TAPC_USERKEYCMP0 (User Key Compare0 Register, TAPC).....33–16
TAPC_USERKEYCMP1 (User Key Compare1 Register, TAPC).....33–17
TAPC_USERKEYCMP2 (User Key Compare2 Register, TAPC).....33–18
TAPC_USERKEYCMP3 (User Key Compare3 Register, TAPC).....33–19
TAPC0_KEYFAIL interrupt 7–7

Target Register n, SWU (SWU_TARG[n])....32–25
TBU, EMAC CORE....23–40
TCOE, EMAC CORE....23–40
Temporary Mailbox Disable Register, CAN (CAN_MBTD)....21–55
Testmode Register, USB (USB_TESTMODE)....22–199
test modes, CAN....21–23
TFC, EMAC CORE....23–40
threshold mode, EMAC FIFO....23–36,23–37,23–40
time correction, PTP....23–69,23–81
time quantum (TQ), CAN....21–15
TIMER_BCAST_DLY (Broadcast Delay Register, TIMER)....14–28
TIMER_BCAST_PER (Broadcast Period Register, TIMER)....14–29
TIMER_BCAST_WID (Broadcast Width Register, TIMER)....14–30
TIMER_DATA_ILAT (Data Interrupt Latch Register, TIMER)....14–31
TIMER_DATA_IMSK (Data Interrupt Mask Register, TIMER)....14–32
TIMER_ERR_TYPE (Error Type Status Register, TIMER)....14–33
TIMER_RUN_CLR (Run Clear Register, TIMER)....14–36
TIMER_RUN_SET (Run Set Register, TIMER)....14–37
TIMER_RUN (Run Register, TIMER)....14–35
TIMER_STAT_ILAT (Status Interrupt Latch Register, TIMER)....14–38
TIMER_STAT_IMSK (Status Interrupt Mask Register, TIMER)....14–39
TIMER_STOP_CFG_CLR (Stop Configuration Clear Register, TIMER)....14–41
TIMER_STOP_CFG_SET (Stop Configuration Set Register, TIMER)....14–42
TIMER_STOP_CFG (Stop Configuration Register, TIMER)....14–40
TIMER_TMR[n]_CFG (Timer n Configuration Register, TIMER)....14–43
TIMER_TMR[n]_CNT (Timer n Counter Register, TIMER)....14–47
TIMER_TMR[n]_DLY (Timer n Delay Register, TIMER)....14–48
TIMER_TMR[n]_PER (Timer n Period Register, TIMER)....14–49
TIMER_TMR[n]_WID (Timer n Width Register, TIMER)....14–50
TIMER_TRG_IE (Trigger Slave Enable Register, TIMER)....14–51
TIMER_TRG_MSK (Trigger Master Mask Register, TIMER)....14–52
timer, watchdog, *see* WDOG
TIMER0_STAT interrupt....7–4,14–3
TIMER0_TMR0_MST trigger....8–2,14–3
TIMER0_TMR0_SLV0 trigger....8–5,14–3
TIMER0_TMR0 interrupt....7–4,14–3
TIMER0_TMR1_MST trigger....8–2,14–3
TIMER0_TMR1_SLV0 trigger....8–5,14–3
TIMER0_TMR1 interrupt....7–4,14–3
TIMER0_TMR2_MST trigger....8–2,14–3
TIMER0_TMR2_SLV0 trigger....8–5,14–3
TIMER0_TMR2 interrupt....7–4,14–3
TIMER0_TMR3_MST trigger....8–2,14–3
TIMER0_TMR3_SLV0 trigger....8–5,14–3
TIMER0_TMR3 interrupt....7–4,14–3
TIMER0_TMR4_MST trigger....8–2,14–3
TIMER0_TMR4_SLV0 trigger....8–5,14–3
TIMER0_TMR4 interrupt....7–4,14–3
TIMER0_TMR5_MST trigger....8–2,14–3
TIMER0_TMR5_SLV0 trigger....8–5,14–4
TIMER0_TMR5 interrupt....7–4,14–3
TIMER0_TMR6_MST trigger....8–2,14–3
TIMER0_TMR6_SLV0 trigger....8–5,14–4
TIMER0_TMR6 interrupt....7–4,14–3
TIMER0_TMR7_MST trigger....8–2,14–3
TIMER0_TMR7_SLV0 trigger....8–5,14–4
TIMER0_TMR7 interrupt....7–4,14–3
Timer 0 Current Count Register, ADCC (ADCC_TMR0)....26–89
Timer 0 Period Register, PWM (PWM_TM0)....18–127
Timer 0 Status Register, ADCC (ADCC_T0STAT)....26–83
Timer 1 Current Count Register, ADCC (ADCC_TMR1)....26–90
Timer 1 Period Register, PWM (PWM_TM1)....18–128
Timer 1 Status Register, ADCC (ADCC_T1STAT)....26–84
Timer 2 Period Register, PWM (PWM_TM2)....18–129
Timer 3 Period Register, PWM (PWM_TM3)....18–130
Timer 4 Period Register, PWM (PWM_TM4)....18–131
Timer n Configuration Register, TIMER (TIMER_TMR[n]_CFG)....14–43
Timer n Counter Register, TIMER (TIMER_TMR[n]_CNT)....14–47

Timer n Delay Register, TIMER (TIMER_TMR[n]_DLY)... 14–48
Timer n Period Register, TIMER (TIMER_TMR[n]_PER).. 14–49
Timer n Width Register, TIMER (TIMER_TMR[n]_WID). 14–50
Time Stamp Addend Register, EMAC (EMAC_TM_ADDEND)....23–220
Time Stamp Auxiliary TM Seconds Register, EMAC (EMAC_TM_AUXSTMP_SEC).... 23–222
Time Stamp Auxiliary TS Nano Seconds Register, EMAC (EMAC_TM_AUXSTMP_NSEC).... 23–221
Timestamp Control Register, CGU (CGU_TSCTL).....4–28
Time Stamp Control Register, EMAC (EMAC_TM_CTL)... 23–223
Time Stamp Counter 32 LSB Register, CGU (CGU_TSCOUNT0).... 4–26
Time Stamp Counter 32 MSB Register, CGU (CGU_TSCOUNT1).... 4–27
Timestamp Counter Initial 32 LSB Value Register, CGU (CGU_TSVALUE0)....4–29
Time Stamp Counter Initial MSB Value Register, CGU (CGU_TSVALUE1)....4–30
Time Stamp High Second Register, EMAC (EMAC_TM_HISEC).... 23–227
timestamping, PTP.....23–19,23–58,23–61,23–67,23–82
Time Stamp Low Seconds Register, EMAC (EMAC_TM_SEC)....23–236
timestamp mode, CAN....21–13
Time Stamp Nanoseconds Register, EMAC (EMAC_TM_NSEC)....23–228
Time Stamp Nanoseconds Update Register, EMAC (EMAC_TM_NSECUPDT)....23–229
Time Stamp PPS Interval Register, EMAC (EMAC_TM_PPSINTVL).... 23–234
Time Stamp Seconds Update Register, EMAC (EMAC_TM_SECUPDT)....23–237
Time Stamp Status Register, EMAC (EMAC_TM_STMPSTAT).... 23–238
Time Stamp Sub Second Increment Register, EMAC (EMAC_TM_SUBSEC).... 23–240
Time Stamp Target Time Nanoseconds Register, EMAC (EMAC_TM_NTGTM)....23–230
Time Stamp Target Time Seconds Register, EMAC (EMAC_TM_TGTM)....23–241
Timing Control 0 Register, DACC (DACC_TC0)..... 27–48
Timing Control 1 Register, DACC (DACC_TC1)..... 27–49
Timing Control A (ADC0) Register, ADCC (ADCC_TCA0).... 26–85
Timing Control A (ADC1) Register, ADCC (ADCC_TCA1).... 26–86
Timing Control B (ADC0) Register, ADCC (ADCC_TCB0).... 26–87
Timing Control B (ADC1) Register, ADCC (ADCC_TCB1).... 26–88
Timing Register, CAN (CAN_TIMING)....21–82
timing unit, PWM....18–6,18–12
TPE, EMAC CORE....23–42
transfer mode, CRC....11–9
transfer modes, USB.... 22–14,22–64,22–67
transfer size (TxferSize).... 22–23,22–25
transition operation, DMA.... 12–18
Transmission Acknowledge 1 Register, CAN (CAN_TA1)..... 21–80
Transmission Acknowledge 2 Register, CAN (CAN_TA2)..... 21–81
Transmission Request Reset 1 Register, CAN (CAN_TRR1). 21–83
Transmission Request Reset 2 Register, CAN (CAN_TRR2). 21–84
Transmission Request Set 1 Register, CAN (CAN_TRS1)..... 21–85
Transmission Request Set 2 Register, CAN (CAN_TRS2)..... 21–86
Transmit Address/Insert Pulse Register, UART (UART_TAIP).... 19–49
transmit bus interface module (TBU), EMAC CORE. 23–40
transmit checksum offload engine (TCOE), EMAC CORE.. 23–40
Transmit Control Register, SPI (SPI_TXCTL).....24–83
Transmit Counter Register, UART (UART_TXCNT). 19–52
transmit CRC generator module (CTX), EMAC CORE..... 23–44
transmit data valid signal, SPORT.... 25–6
Transmit FIFO Data Register, SPI (SPI_TFIFO).....24–80
transmit flow control module (FTX), EMAC CORE.. 23–44
transmit frame controller module (TFC), EMAC CORE..... 23–40
Transmit Hold Register, UART (UART_THR)..... 19–50
Transmit Interrupt Enable Register, USB (USB_INTRTXE). 22–170
Transmit Interrupt Register, USB (USB_INTRTX)...22–169

transmit protocol engine module (TPE), EMAC CORE..... 23–42
transmit scheduler module (STX), EMAC CORE..... 23–43
Transmit Shift Register, UART (UART_TSR)..... 19–51
Transmitted Word Count Register, SPI (SPI_TWC).... 24–81
Transmitted Word Count Reload Register, SPI (SPI_TWCR)..... 24–82
trigger 8–4-8–7
trigger, target time alarm, PTP..... 23–81
trigger master, TRU..... 8–7,8–8
Trigger Master Mask Register, TIMER (TIMER_TRG_MSK)..... 14–52
trigger routing unit, *see* TRU
trigger routing unit (TRU), SWU..... 32–6
triggers, DMA..... 12–32,12–39
see also DMA event control
trigger slave, TRU..... 8–7,8–8
Trigger Slave Enable Register, TIMER (TIMER_TRG_IE)... 14–51
Trip Configuration Register, PWM (PWM_TRIPCFG)..... 18–132
trip control unit, PWM..... 18–34
TRU
architectural concepts..... 8–8
block diagram..... 8–7
definitions..... 8–7
event control..... 8–9
features..... 8–1
functional description..... 8–1
overview..... 8–1
programming..... 8–8
TRU_ERRADDR (Error Address Register, TRU)..... 8–10
TRU_GCTL (Global Control Register, TRU)..... 8–11
TRU_MTR (Master Trigger Register, TRU)..... 8–13
TRU_SSR[n] (Slave Select Register, TRU)..... 8–14
TRU_STAT (Status Information Register, TRU)..... 8–15
TRU0_INT0 interrupt 7–7,8–2
TRU0_INT1 interrupt 7–7,8–2
TRU0_INT2 interrupt 7–7,8–2
TRU0_INT3 interrupt 7–7,8–2
TRU0_SLV0 trigger 8–5
TRU0_SLV1 trigger 8–5
TRU0_SLV2 trigger 8–5
TRU0_SLV3 trigger 8–5
TWI
architectural concepts..... 20–4
clock..... 20–5,20–14
data transfer..... 20–5
features..... 20–1
functional description..... 20–2
interface..... 20–3,20–4
operating modes..... 20–7
overview..... 20–1
programming..... 20–12
TWI_CLKDIV (SCL Clock Divider Register, TWI)... 20–17
TWI_CTL (Control Register, TWI)..... 20–18
TWI_FIFOCTL (FIFO Control Register, TWI)..... 20–20
TWI_FIFOSTAT (FIFO Status Register, TWI)..... 20–22
TWI_IMSK (Interrupt Mask Register, TWI)..... 20–23
TWI_ISTAT (Interrupt Status Register, TWI)..... 20–25
TWI_MSTRADDR (Master Mode Address Register, TWI).. 20–28
TWI_MSTRCTL (Master Mode Control Registers, TWI).... 20–29
TWI_MSTRSTAT (Master Mode Status Register, TWI)..... 20–32
TWI_RXDATA16 (Rx Data Double-Byte Register, TWI)..... 20–35
TWI_RXDATA8 (Rx Data Single-Byte Register, TWI)..... 20–36
TWI_SLVADDR (Slave Mode Address Register, TWI)..... 20–37
TWI_SLVCTL (Slave Mode Control Register, TWI).. 20–38
TWI_SLVSTAT (Slave Mode Status Register, TWI).... 20–40
TWI_TXDATA16 (Tx Data Double-Byte Register, TWI).... 20–41
TWI_TXDATA8 (Tx Data Single-Byte Register, TWI)..... 20–42
TWI0_DATA interrupt 7–6,20–3
two-dimensional mode, DMA..... 12–28
two-wire interface, *see* TWI
Tx 1024- to Max-Byte Frames (Good/Bad) Register, EMAC (EMAC_TX1024TOMAX_GB)..... 23–242
Tx 128- to 255-Byte Frames (Good/Bad) Register, EMAC (EMAC_TX128TO255_GB)..... 23–243
Tx 256- to 511-Byte Frames (Good/Bad) Register, EMAC (EMAC_TX256TO511_GB)..... 23–244
Tx 512- to 1023-Byte Frames (Good/Bad) Register, EMAC (EMAC_TX512TO1023_GB)..... 23–245
Tx 64-Byte Frames (Good/Bad) Register, EMAC (EMAC_TX64_GB)..... 23–246

Tx 65- to 127-Byte Frames (Good/Bad) Register, EMAC (EMAC_TX65TO127_GB)....23–247
Tx Broadcast Frames (Good/Bad) Register, EMAC (EMAC_TXBCASTFRM_GB)....23–249
Tx Broadcast Frames (Good) Register, EMAC (EMAC_TXBCASTFRM_G)....23–248
Tx Carrier Error Register, EMAC (EMAC_TXCARR_ERR) 23–250
Tx Data Double-Byte Register, TWI (TWI_TXDATA16).... 20–41
Tx Data Single-Byte Register, TWI (TWI_TXDATA8)..... 20–42
Tx Deferred Register, EMAC (EMAC_TXDEFERRED)..... 23–251
TX Double Packet Buffer Disable for Endpoints 1 to 3, USB (USB_TXDPKTBUFDIS)....22–200
Tx Excess Collision Register, EMAC (EMAC_TXEXCESS-COL)....23–252
Tx Excess Deferral Register, EMAC (EMAC_TXEXCESS-DEF)....23–253
TxferSize (transfer size)....22–23,22–25
Tx Frame Count (Good/Bad) Register, EMAC (EMAC_TXFRMCNT_GB)....23–255
Tx Frame Count (Good) Register, EMAC (EMAC_TXFRMCNT_G)....23–254
Tx Late Collision Register, EMAC (EMAC_TXLATECOL). 23–256
Tx Multicast Frames (Good/Bad) Register, EMAC (EMAC_TXMCASTFRM_GB)....23–258
Tx Multicast Frames (Good) Register, EMAC (EMAC_TXMCASTFRM_G)....23–257
Tx Multiple Collision (Good) Register, EMAC (EMAC_TXMULTCOL_G)....23–259
Tx OCT Count (Good/Bad) Register, EMAC (EMAC_TXOCTCNT_GB)....23–261
Tx Octet Count (Good) Register, EMAC (EMAC_TXOCTCNT_G)....23–260
Tx Pause Frame Register, EMAC (EMAC_TXPAUSEFRM). 23–262
Tx Single Collision (Good) Register, EMAC (EMAC_TXSNGCOL_G)....23–263
Tx Underflow Error Register, EMAC (EMAC_TXUNDR_ERR)....23–265
Tx Unicast Frames (Good/Bad) Register, EMAC (EMAC_TXUCASTFRM_GB)....23–264
Tx VLAN Frames (Good) Register, EMAC (EMAC_TXVLANFRM_G)....23–266

U

UART
architectural concepts....19–4
block diagram....19–4
clock....19–6
data transfer modes....19–11,19–23
DMA....19–16
event control....19–17,19–18,19–23
features....19–1
functional description....19–2
interface....19–5
operating modes....19–9
overview....19–1
programming....19–22
UART_CLK (Clock Rate Register, UART)....19–25
UART_CTL (Control Register, UART)....19–26
UART_IMSK_CLR (Interrupt Mask Clear Register, UART) 19–36
UART_IMSK_SET (Interrupt Mask Set Register, UART).... 19–38
UART_IMSK (Interrupt Mask Register, UART)....19–33
UART_RBR (Receive Buffer Register, UART)....19–40
UART_RSR (Receive Shift Register, UART)....19–41
UART_RXCNT (Receive Counter Register, UART)...19–42
UART_SCR (Scratch Register, UART)....19–43
UART_STAT (Status Register, UART)....19–44
UART_TAIP (Transmit Address/Insert Pulse Register, UART)....19–49
UART_THR (Transmit Hold Register, UART)....19–50
UART_TSR (Transmit Shift Register, UART)....19–51
UART_TXCNT (Transmit Counter Register, UART).19–52
UART0_RXDMA_ERR interrupt 7–3,19–3
UART0_RXDMA interrupt 7–5,19–3
UART0_RXDMA trigger 8–3,8–6
UART0_STAT interrupt 7–5,19–3
UART0_TXDMA_ERR interrupt 7–3,19–3
UART0_TXDMA interrupt 7–5,19–3
UART0_TXDMA trigger 8–3,8–6
UART1_RXDMA_ERR interrupt 7–4,19–3
UART1_RXDMA interrupt 7–5,19–4
UART1_RXDMA trigger 8–4,8–6
UART1_STAT interrupt 7–5,19–3
UART1_TXDMA_ERR interrupt 7–4,19–3

UART1_TXDMA interrupt 7–5,19–4
UART1_TXDMA trigger 8–3,8–6
UART2_RXDMA_ERR interrupt 7–4,19–3
UART2_RXDMA interrupt 7–6,19–4
UART2_RXDMA trigger 8–4,8–6
UART2_STAT interrupt 7–6,19–4
UART2_TXDMA_ERR interrupt 7–4,19–3
UART2_TXDMA interrupt 7–6,19–4
UART2_TXDMA trigger 8–4,8–6
unicast addressing, AFM, EMAC CORE..... 23–48
universal asynchronous receiver/transmitter *, see* UART
Universal Counter Configuration Mode Register, CAN (CAN_UCCNF)..... 21–87
Universal Counter Register, CAN (CAN_UCCNT).... 21–89
Universal Counter Reload/Capture Register, CAN (CAN_UCRC)..... 21–90
universal serial bus*, see* USB
universal serial bus (USB)..... 3–5
up down counter mode, CNT..... 17–5,17–16
Upper Address Register n, SWU (SWU_UA[n])..... 32–26
USB
  architectural concepts..... 22–2
  block diagram..... 22–8
  clocking..... 22–4
  event control..... 22–37,22–43,22–66
  features..... 22–1
  functional description..... 22–2
  interface..... 22–2
  operating modes..... 22–11,22–29,22–39
  overview..... 22–1
  programming..... 22–46,22–67
  reset..... 22–45
USB_DEV_CTL (Device Control Register, USB)..... 22–73
USB_DMA_IRQ (DMA Interrupt Register, USB)..... 22–80
USB_DMA[n]_ADDR (DMA Channel n Address Register, USB)..... 22–75
USB_DMA[n]_CNT (DMA Channel n Count Register, USB)..... 22–76
USB_DMA[n]_CTL (DMA Channel n Control Register, USB)..... 22–77
USB_EP[n]_FIFOSZ (FIFO Size, USB)..... 22–131
USB_EP[n]_RXCNT (EPn Number of Bytes Received Register, USB)..... 22–132
USB_EP[n]_RXCSR_H (EPn Receive Configuration and Status (Host) Register, USB)..... 22–133
USB_EP[n]_RXCSR_P (EPn Receive Configuration and Status (Peripheral) Register, USB)..... 22–138
USB_EP[n]_RXINTERVAL (EPn Receive Polling Interval Register, USB)..... 22–142
USB_EP[n]_RXMAXP (EPn Receive Maximum Packet Length Register, USB)..... 22–143
USB_EP[n]_RXTYPE (EPn Receive Type Register, USB)..... 22–144
USB_EP[n]_TXCSR_H (EPn Transmit Configuration and Status (Host) Register, USB)..... 22–146
USB_EP[n]_TXCSR_P (EPn Transmit Configuration and Status (Peripheral) Register, USB)..... 22–150
USB_EP[n]_TXINTERVAL (EPn Transmit Polling Interval Register, USB)..... 22–153
USB_EP[n]_TXMAXP (EPn Transmit Maximum Packet Length Register, USB)..... 22–154
USB_EP[n]_TXTYPE (EPn Transmit Type Register, USB)..... 22–155
USB_EP0_CFGDATA[n] (EP0 Configuration Information Register, USB)..... 22–93
USB_EP0_CNT[n] (EP0 Number of Received Bytes Register, USB)..... 22–95
USB_EP0_CSR[n]_H (EP0 Configuration and Status (Host) Register, USB)..... 22–96
USB_EP0_CSR[n]_P (EP0 Configuration and Status (Peripheral) Register, USB)..... 22–99
USB_EP0_NAKLIMIT[n] (EP0 NAK Limit Register, USB)..... 22–102
USB_EP0_TYPE[n] (EP0 Connection Type Register, USB)..... 22–103
USB_EP0I_CFGDATA[N] (EP0 Configuration Information Register, USB)..... 22–82
USB_EP0I_CNT[N] (EP0 Number of Received Bytes Register, USB)..... 22–84
USB_EP0I_CSR[N]_H (EP0 Configuration and Status (Host) Register, USB)..... 22–85
USB_EP0I_CSR[N]_P (EP0 Configuration and Status (Peripheral) Register, USB)..... 22–88
USB_EP0I_NAKLIMIT[N] (EP0 NAK Limit Register, USB)..... 22–91
USB_EP0I_TYPE[N] (EP0 Connection Type Register, USB)..... 22–92
USB_EPI_FIFOSZ (FIFO Size, USB)..... 22–130
USB_EPI[N]_RXCNT (EPn Number of Bytes Received Register, USB)..... 22–105

USB_EPI[N]_RXCSR_H (EPn Receive Configuration and Status (Host) Register, USB).....22–106
USB_EPI[N]_RXCSR_P (EPn Receive Configuration and Status (Peripheral) Register, USB).....22–111
USB_EPI[N]_RXINTERVAL (EPn Receive Polling Interval Register, USB)..... 22–115
USB_EPI[N]_RXMAXP (EPn Receive Maximum Packet Length Register, USB)..... 22–116
USB_EPI[N]_RXTYPE (EPn Receive Type Register, USB)... 22–117
USB_EPI[N]_TXCSR_H (EPn Transmit Configuration and Status (Host) Register, USB).....22–119
USB_EPI[N]_TXCSR_P (EPn Transmit Configuration and Status (Peripheral) Register, USB).....22–123
USB_EPI[N]_TXINTERVAL (EPn Transmit Polling Interval Register, USB).....22–126
USB_EPI[N]_TXMAXP (EPn Transmit Maximum Packet Length Register, USB)..... 22–127
USB_EPI[N]_TXTYPE (EPn Transmit Type Register, USB) 22–128
USB_EPINFO (Endpoint Information Register, USB)..... 22–104
USB_FADDR (Function Address Register, USB)..... 22–157
USB_FIFO[n] (FIFO Word (32-Bit) Register, USB). 22–160
USB_FIFOB[n] (FIFO Byte (8-Bit) Register, USB)...22–158
USB_FIFOH[n] (FIFO Half-Word (16-Bit) Register, USB).. 22–159
USB_FRAME (Frame Number Register, USB).....22–161
USB_FS_EOF1 (Full-Speed EOF 1 Register, USB)...22–162
USB_IDCTL (ID Control, USB).....22–163
USB_IEN (Common Interrupts Enable Register, USB)..... 22–164
USB_INDEX (Index Register, USB)..... 22–166
USB_INTRRX (Receive Interrupt Register, USB)..... 22–167
USB_INTRRXE (Receive Interrupt Enable Register, USB)... 22–168
USB_INTRTX (Transmit Interrupt Register, USB)...22–169
USB_INTRTXE (Transmit Interrupt Enable Register, USB). 22–170
USB_IRQ (Common Interrupts Register, USB)..... 22–171
USB_LINKINFO (Link Information Register, USB) 22–173
USB_LPM_ATTR (LPM Attribute Register, USB)... 22–174
USB_LPM_CTL (LPM Control Register, USB).....22–175
USB_LPM_FADDR (LPM Function Address Register, USB)..... 22–177
USB_LPM_IEN (LPM Interrupt Enable Register, USB)..... 22–178
USB_LPM_IRQ (LPM Interrupt Status Register, USB)..... 22–180
USB_LS_EOF1 (Low-Speed EOF 1 Register, USB).. 22–183
USB_MP[n]_RXFUNCADDR (MPn Receive Function Address Register, USB).....22–184
USB_MP[n]_RXHUBADDR (MPn Receive Hub Address Register, USB)..... 22–185
USB_MP[n]_RXHUBPORT (MPn Receive Hub Port Register, USB).....22–186
USB_MP[n]_TXFUNCADDR (MPn Transmit Function Address Register, USB)..... 22–187
USB_MP[n]_TXHUBADDR (MPn Transmit Hub Address Register, USB)..... 22–188
USB_MP[n]_TXHUBPORT (MPn Transmit Hub Port Register, USB)..... 22–189
USB_PHY_CTL (FS PHY Control, USB)..... 22–190
USB_PHY_STAT (FS PHY Status, USB).....22–191
USB_POWER (Power and Device Control Register, USB).... 22–192
USB_RAMINFO (RAM Information Register, USB)22–195
USB_RQPKTCNT[n] (EPn Request Packet Count Register, USB)..... 22–196
USB_RXDPKTBUFDIS (RX Double Packet Buffer Disable for Endpoints 1 to 3, USB).....22–197
USB_SOFT_RST (Software Reset Register, USB)..... 22–198
USB_TESTMODE (Testmode Register, USB).....22–199
USB_TXDPKTBUFDIS (TX Double Packet Buffer Disable for Endpoints 1 to 3, USB).....22–200
USB_VBUS_CTL (VBUS Control Register, USB)....22–201
USB_VPLEN (VBUS Pulse Length Register, USB)... 22–202
USB0_DATA interrupt 7–7,22–8
USB0_DATA trigger 8–4,22–8
USB0_STAT interrupt 7–6,22–8
USB DMA
  bus cycles.....22–66
  master channels..... 22–65
  programming.....22–46
  transfering packets..... 22–67
USB endpoint
  allocating devices..... 22–35
  configuration.....22–12,22–67
  service routine..... 22–18

states....22–18
USB host mode
endpoint configuration....22–30
multi-point support....22–35
operating....22–2,22–11,22–29
OTG supplement....22–1
programming....22–46
suspend/resume....22–39
transaction control....22–29,22–30
USB peripheral mode
endpoint 0 service routine....22–18
error handling....22–28
operating....22–2,22–11
programming....22–46
suspend....22–27,22–40
transfer work flows....22–14,22–23,22–28
USB transfer flows
bulk....22–14
control....22–14,22–15,22–29
interrupts....22–14
isochronous....22–14
USERCODE Register, TAPC (TAPC_USERCODE). 33–15
User Key Compare0 Register, TAPC (TAPC_USERKEYCMP0)....33–16
User Key Compare1 Register, TAPC (TAPC_USERKEYCMP1)....33–17
User Key Compare2 Register, TAPC (TAPC_USERKEYCMP2)....33–18
User Key Compare3 Register, TAPC (TAPC_USERKEYCMP3)....33–19
UTMI interface, USB....22–5

V

V (Voltage) Sample Register, HAE (HAE_VSAMPLE)28–33
V (Voltage) Waveform Register, HAE (HAE_VWAVEFORM)....28–34
VBUS Control Register, USB (USB_VBUS_CTL)....22–201
VBUS event, USB....22–38
VBUS events, USB....22–37
VBUS Pulse Length Register, USB (USB_VPLEN)... 22–202
VLAN Tag Register, EMAC (EMAC_VLANTAG)... 23–267
Voltage Level Register, HAE (HAE_VLEVEL)....28–32

W

Wakeup Enable Register, DPM (DPM_WAKE_EN).....6–17
Wakeup Polarity Register, DPM (DPM_WAKE_POL). 6–18
wake-up signals/sources....6–4
Wakeup Status Register, DPM (DPM_WAKE_STAT).. 6–19
watchdog, oscillator, CGU....4–7
watchdog mode, CAN....21–12
WATCHDOG mode, GP timer....14–5,14–14,14–25
watchdog timer *, see* WDOG
Watchdog Timer Status Register, WDOG (WDOG_STAT)....16–6
watchpoint mode, SWU....32–3,32–5,32–8
waveforms, output, PWM....18–13,18–14,18–31
WDOG
features....16–1
interface....16–2
overview....16–1
WDOG_CNT (Count Register, WDOG)....16–4
WDOG_CTL (Control Register, WDOG)....16–5
WDOG_STAT (Watchdog Timer Status Register, WDOG)....16–6
WDOG0_EXP interrupt 7–3,16–2
WIDCAP mode, GP timer....14–5,14–9,14–24,19–7
width capture mode *, see* WIDCAP mode
windowed watchdog mode *, see* WATCHDOG mode
window offset (WOFFSET), SPORT....25–33
write protection register, SPU....5–2,5–5
Write Protect Register n, SPU (SPU_WP[n])....5–9

Z

zero-marker-error mode, CNT....17–11,17–17
zero-marker-zeros-counter mode, CNT....17–10,17–17
zero-once mode, CNT....17–11