



Technical notes on using Analog Devices products and development tools
Visit our Web resources <http://www.analog.com/ee-notes> and <http://www.analog.com/processors> or
e-mail processor.support@analog.com or processor.tools.support@analog.com for technical support.

ADSP-CM40x Serial Flash Execute-in-Place Technology

Contributed by Prasanth Rajagopal, Prashant Gawade, and Kritika Shahu

Rev 2 – December 7, 2015

Introduction

The ADSP-CM40x series of mixed-signal control processors is based on the ARM Cortex™-M4 core, and it includes features to support fast memory accesses for motor control and industrial applications, among others. These devices also offer large on-chip SRAM memory (up to 384 KB) that can be accessed at core clock speeds of up to 240 MHz, as well as a system in package (SiP) serial flash memory device (up to 2 MB).

Furthermore, ADSP-CM40x processors support direct code execution from Serial Peripheral Interface (SPI) flash devices. Most flash memory devices support features such as quad-I/O fast read, execute in place (XiP), and wrap, all of which are supported directly by the embedded on-chip SPI Controller. These features, when combined with instruction cache support, cache pre-fetch support, and merged accesses, enable code execution from serial flash devices at fast execution rates.

This EE-Note discusses each block involved in the transfer path of SPI-flash-based direct code execution and how the various sub-blocks interact with each other. The main blocks consist of flash memory, the SPI controller, and the cache controller. Also, comparison of SPI flash throughput benchmarks with different memories is provided in the [Performance Analysis](#) section. Several oscilloscope plots explaining the SPI memory-mapped mode features, such as wrap and XiP, are provided in [Appendix A](#). The associated .ZIP file^[1] contains code examples that demonstrate performance analysis and improvement techniques.



This EE-Note consists of specific information pertaining to the above context (i.e., flash reads involving the cache controller and the SPI controller). For more details on the ADSP-CM40x mixed-signal control processors, please refer to the device *Datasheet*^[2] and the processor *Hardware Reference Manual*^[3].

ADSP-CM40x Memory Map

ADSP-CM40x processors include sufficient internal SRAM to support fast execution of real-time code. Non-real-time code can be executed from SPI flash memory, as well as from external SRAM via the Static Memory Controller (SMC). [Figure 1](#) and [Figure 2](#) provide an overview of the ADSP-CM40x processor memory map.

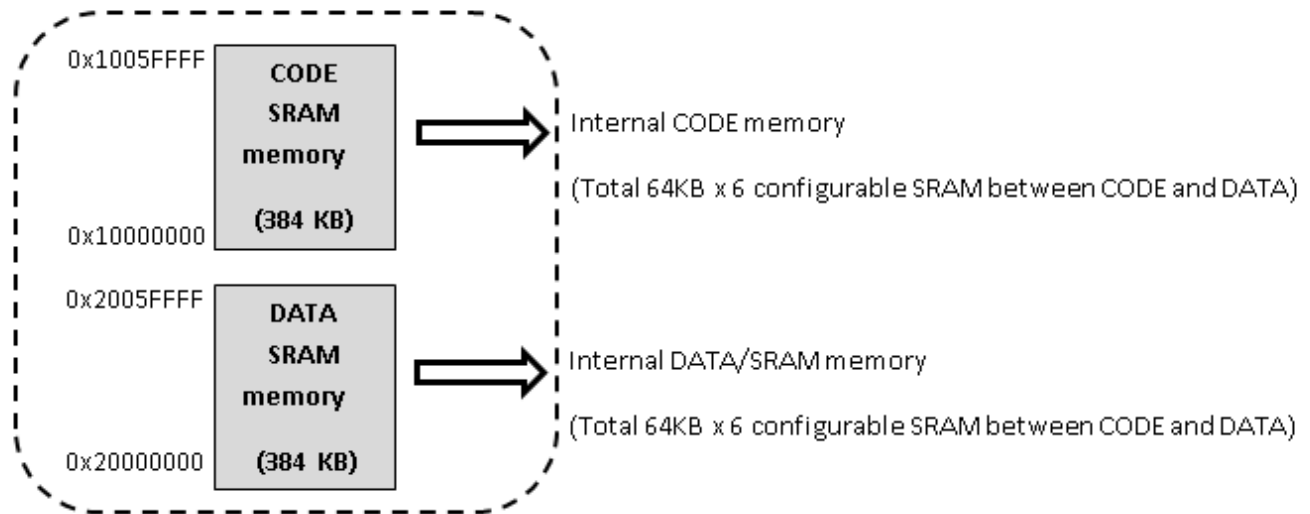


Figure 1. ADSP-CM40x Internal SRAM Space

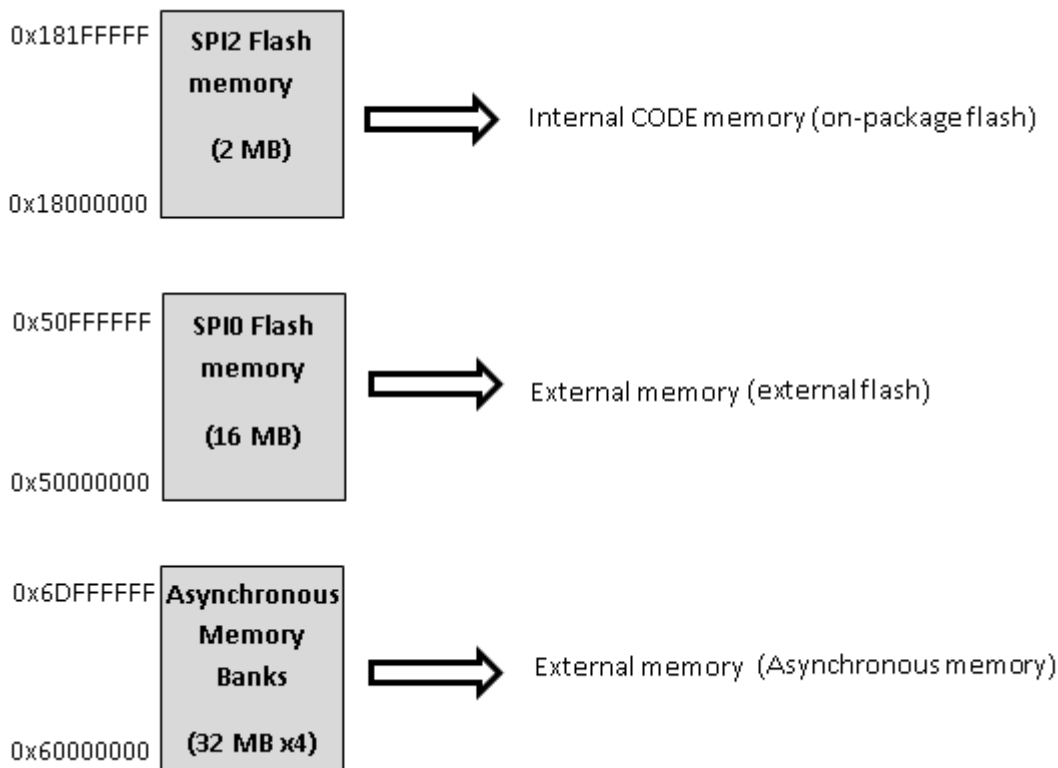


Figure 2. Cacheable External Memory Blocks

Recommended Memory Management Strategy

The following guidelines for typical memory management strategy can be helpful during ADSP-CM40x application development:

1. Place all code and data in internal SRAM (up to 64 KB x 6 banks of on-chip SRAM available).
2. If it doesn't fit in internal SRAM, profile the code using the development tools (typically using a map file, which can be generated through the IAR embedded workbench). Map the real-time code (including all ISRs) to on-chip SRAM, and map the setup code and initialization routine to flash, as these routines do not execute frequently and are typically less time-critical.
3. Cache all other non-critical code executing from the in-package SPI flash space (enabled by default). For fast execution rates, configure the SPI controller for quad-I/O operation with Command Skip / Continuous Read mode, merge, and cache pre-fetch enabled, and set the SPI baud rate to the maximum supported speed of 100 MHz.
4. Consider mapping constant data to flash memory and again utilizing the cache. Shared/split memory banks are available for data and code accesses to cache memory.
5. If more memory space than that available in the on-chip SPI flash is required, use an external SPI flash device. The code can be cached when accessing an external SPI flash device.
6. Lastly, if cost and the selected pin multiplexing scheme allows, consider interfacing with external SRAM memory via the SMC.

Direct Code Execution from Serial Flash Devices

For applications that have code and/or constant data that do not fit into the on-chip SRAM space, additional features have been added to allow the core to efficiently access SPI flash memory space. The following serial memory blocks are available as directly addressable by the core. Because these memories are addressable by the core without additional driver software, direct code execution is available as a built-in feature.

- The SoC SPI flash memory supporting the quad-I/O SPI protocol, with speeds of up to 100 MHz, is available through the SPI2 port.
- The interface to external SPI flash memory supporting the quad-I/O SPI protocol, with speeds of up to 50 MHz, is available through the SPI0 port.

The SPI controller also includes features such as wrap, merge, and XiP mode for efficient accesses to serial flash memory. These features are explained in detail in the [Performance Enhancement Features in the SPI and Cache Controllers](#) section of this EE-Note. An instruction cache is also provided to improve throughput of recently used code. Up to 16 KB of recently executed instructions can be retained inside the cache memory for faster reuse by the application.

Theoretically, internal SPI flash reads can occur at throughputs as high as 50 MB/s. When instructions are brought into internal cache upon a cache miss, any further execution (cache hits) occurs at the core clock rate (up to 240 MHz). Please refer to the [Performance Analysis](#) section for more details on cache performance.

Flash Data Flow Architecture

When the core requests instruction op-codes from memory-mapped regions belonging to SPI flash memories via the MEM-X and MEM-Y blocks (Figure 3), the requests are first directed to the cache controller. If the controller finds a tag match for the requested address, the code is extracted from cache memory and returned to the core at the core clock (CCLK) speed. If there is no tag match, the cache controller will pass the request to the SPI controller. The controller then sends out the translated serial address for SPI flash along with the required commands (options for critical word first and command skips are available).

The SPI controller must be configured appropriately by programming its memory-mapped header register in order to initiate memory-mapped accesses. The cache controller is always enabled, and the default configurations typically ensure that code is cached with optimal performance. It is also possible to bypass cache, though this can result in significant performance loss.

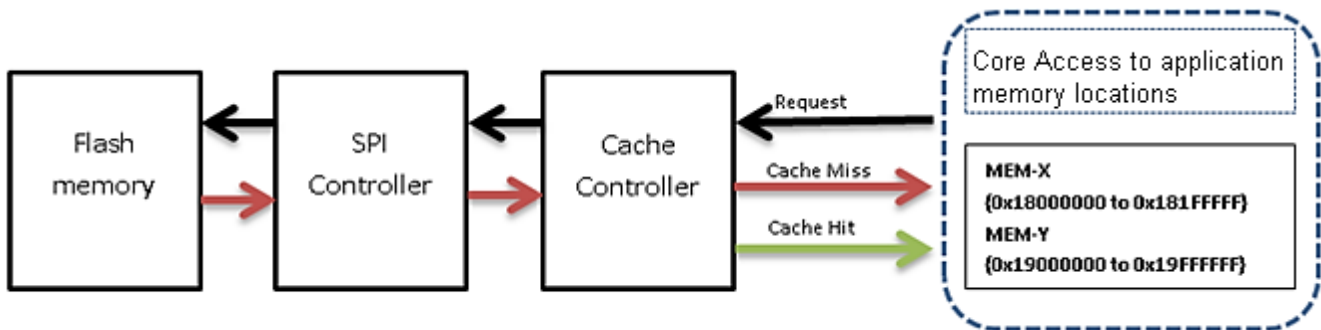


Figure 3. Flash-Based Data Transfer Diagram

The ARM Cortex-M4 core does not have a strict memory space definition for code and data access within each other's memory space. In other words, it allows data to be placed in code regions and code to be placed in data regions. This is inefficient, as it causes bus contention. Since the ADSP-CM40x processor's internal SRAM supports user-configurable partitions between code and data banks, there is usually no need to mix code and data placement. Cache memory, though it is inside the code region, allows for raw data (read-only) to be cached through the D-Code interface, as shown in Figure 4.

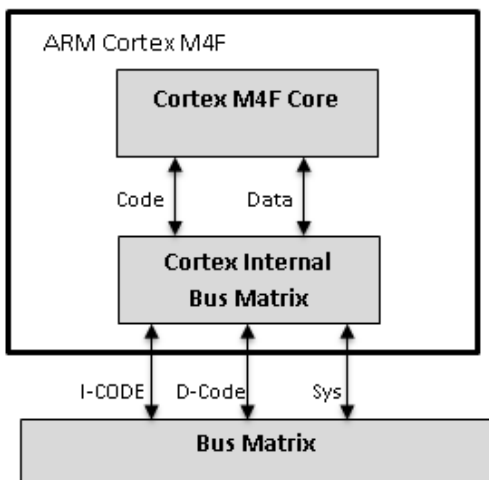



Figure 4. Cortex-M4 Internal Bus Matrix

Memory X and Memory Y Blocks

Both cache memory and the on-chip SPI flash (via the SPI2 port) reside in the code region of the Cortex-M4 core. External SPI flash (via the SPI0 port) and external memory accessed via the SMC can also be cached for better performance, but these blocks reside outside the code region. Therefore, it is preferable to have direct address translation inside the chip. This translation makes sure that all accesses are initiated via the D-Code/I-Code interface (Figure 4), leading to efficient throughput. Otherwise, instruction fetches would contend with data accesses on the SYS (data) bus. These translation memory blocks are referred to as “Memory X” (MEM-X) and “Memory Y” (MEM-Y). The cache controller works only with MEM-X and MEM-Y when performing memory-mapped accesses from SPI0, SPI2, or SMC-interfaced memory.

The translator is assigned to a specific memory space (SPI0, SPI2, or SMC memory) by programming the MEM-X and MEM-Y registers. Unlike sophisticated virtual memory management units, it consists of direct one-to-one mapping. Note that MEM-X and MEM-Y are basically translator regions; these are not populated with any real physical memories, and hence must be thought of as virtual memory regions. Figure 5 to Figure 7 depict the transfer paths involving cache when the core initiates an access.

As illustrated in Figure 5, the most common use case involves the on-chip flash as a device always in use. Therefore, the MEM-X region is typically always assigned to SPI2 internal flash (which is the default configuration). This design allows the application to assign SPI2 flash memory to the MEM-X interface and either of SPI0 flash or SMC memory to the MEM-Y interface.

 Since there are only two memory blocks for translation purposes, the application has to manually manage configuring the MEM-X and MEM-Y interfaces for SPI0 flash, SPI2 flash, and the SMC when all three blocks are in use.

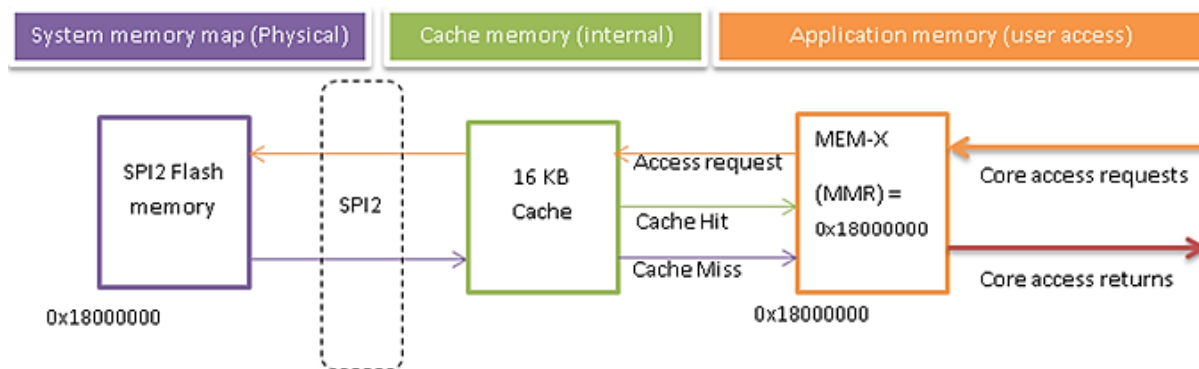


Figure 5. Memory-Mapped and Cache Access Transfer Path for SPI2 Flash

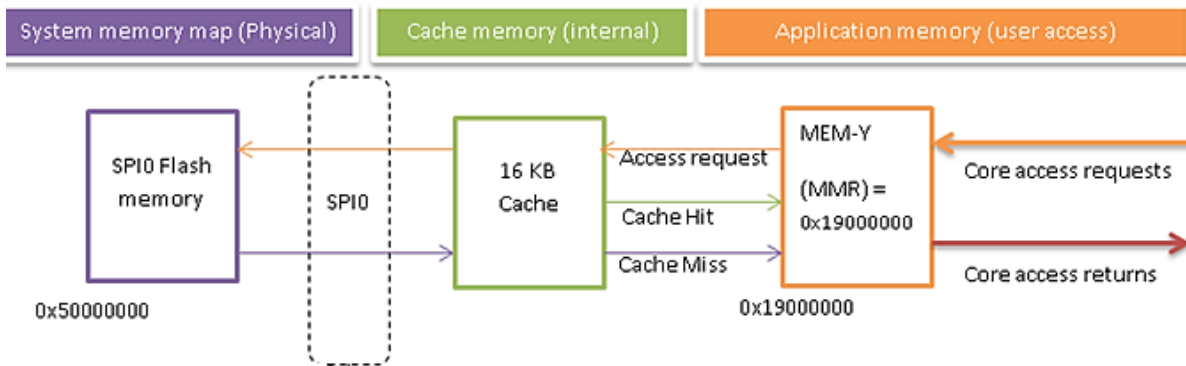


Figure 6. Memory-Mapped and Cache Access Transfer Path for SPI0 Flash

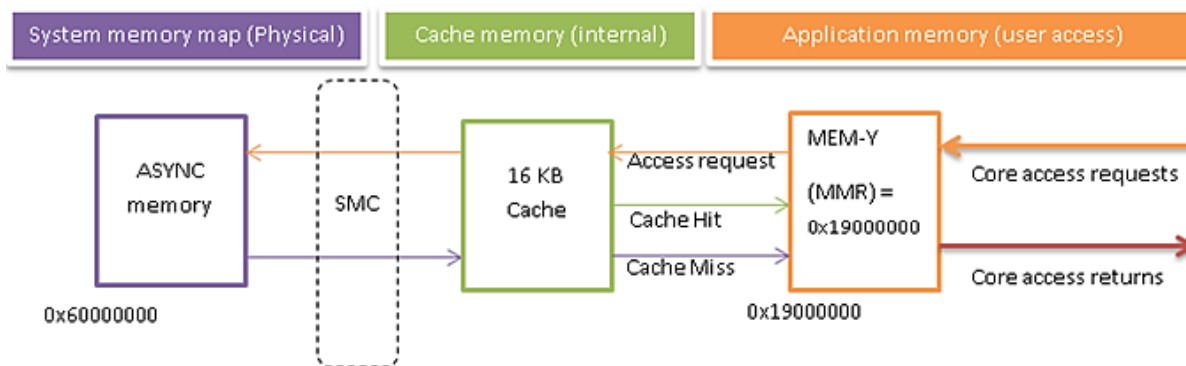


Figure 7. Memory-Mapped and Cache Access Transfer Path for Asynchronous Memory (SMC)

SPI Memory-Mapped Mode

The ADSP-CM40x SPI controller supports direct memory-mapped read accesses to the flash memory interfaced to it. Only an initial configuration is required for setting up the SPI controller and its memory-mapped header register. Once this is done, all the read accesses are automatically carried out by the controller when the core accesses memory-mapped locations.



Only read accesses are supported in memory-mapped mode. For programming the flash with instructions or for writing raw data, the non-memory mapped mode must be used to explicitly configure SPI to send flash instructions, flash addresses, etc.

Memory-Mapped Mode Operation

Memory-Mapped mode allows direct execution of instructions from SPI memory without the need of a low-level software driver. All overhead tasks such as transmission of the read instruction, signal timing, and receive data sizing are handled in hardware. Memory-Mapped mode is enabled by setting the `SPI_CTL.MMSE` bit. Additional programming flexibility is provided in the Memory-Mapped Read Header register (`MMRDH`) for compatibility with a wide range of SPI memory devices. Typical Memory-Mapped mode operation is as follows (Figure 8):

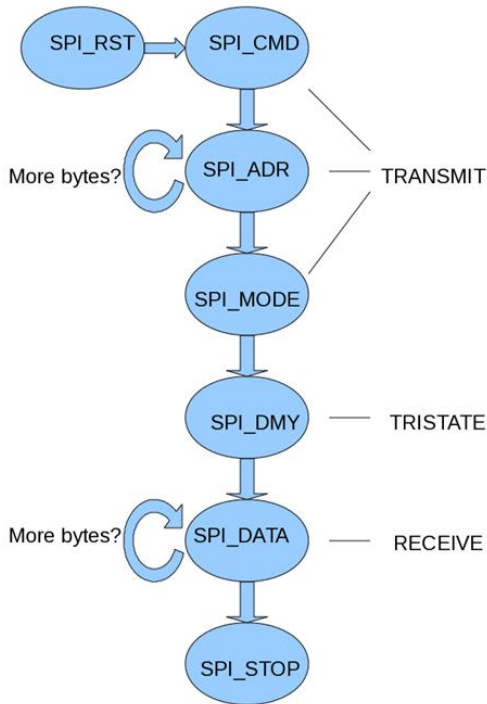


Figure 8. SPI Memory-Mapped Data Transfer Process

1. Once memory-mapped mode is enabled, the SPI device will accept read requests from the core through a dedicated on-chip system bus slave interface.
2. The SPI peripheral, if ready, accepts these requests and begins the process of assembling the read header. The read header is assembled in the SPI transmit FIFO based upon access attributes described in both the `MMRDH` register and the request itself.
3. The read header is then transmitted across the SPI signals `SPIx_MOSI`, `SPIx_MISO`, `SPIx_SEL`, and `SPIx_SCK` (standard SPI mode) or `D[3:0]`, `SPIx_SEL`, and `SPIx_SCK` (multiple I/O mode).
4. Once the read header transmission is complete, a pin turn-around tri-state period is timed (required for multiple I/O mode), and the receiver is enabled.
5. Once all receive data pins are tri-stated, receive data can be driven by the SPI memory device and accepted by the SPI peripheral.
6. The SPI peripheral continues clocking the SPI memory device until all bytes have been received.
7. Upon reception of the last byte, the SPI peripheral typically de-asserts `SPIx_SEL` to prepare for the next read header.

Programmable Features of SPI Memory-Mapped Mode

The available SPI memory-mapped features programmable through the `MMRDH` register provide great flexibility in the choice of SPI memory device type and manufacturer. The supported features are listed below. For more details, refer to the hardware reference manual^[3].

- OPCODE[8] : Read command op-code value, typically 0x03, 0x0B, 0x3B, 0x6B, 0xBB, or 0xEB
- CMDPINS[1]: Number of SPI pins used for sending read command
- CMDSKIP[1]: Command skip enable (for XIP operation)
- ADRSIZE[3]: Number of address bytes for read address
- ADRPINS[1]: Number of SPI pins used for sending address
- DMYSIZE[3]: Number of dummy bytes
- MODE[8]: Mode field used in command skip mode (to be driven during dummy period)
- TRIDMY[2]: Three-state timing during dummy period
- WRAP[1]: Enable burst wrap
- MERGE[1]: Enable merging of two successive cache line transfers

Protocol Example Cases

Figure 9 to Figure 14 show a number of examples to help visualize the important features of the memory-mapped protocol like merge, wrap, and command skip mode. Note that lag, lead, and stop are part of the SPI controller features and are not specific to Memory-Mapped mode.

Example Case 1



Figure 9. 32-Byte Cache Line (Wrap Mode On, Command Skip Off, Word 4 Requested by Processor)

Example Case 2



Figure 10. 32-Byte Cache Line (Wrap Mode Off, Command Skip Off, Word 4 Requested by the Processor)

Example Case 3



Figure 11. 32-Byte Cache Line (Wrap Mode On, Command Skip On, Word 4 Requested by the Processor)

Example Case 4



Figure 12. 32-Byte Cache Line (Wrap Mode Off, Command Skip On, Word 4 Requested by the Processor)

Example Case 5

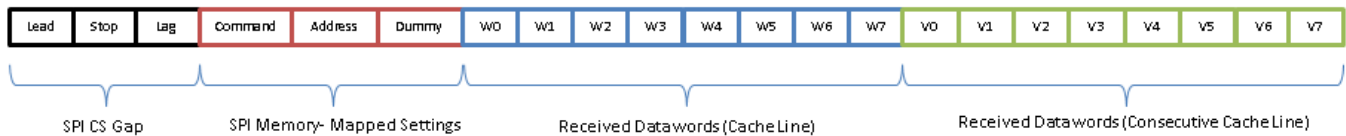


Figure 13. 32-Byte Cache Line (Merge On, Prefetch On, Command Skip Off, Word 0 of Consecutive Cache Lines Requested by the Processor)

Example Case 6

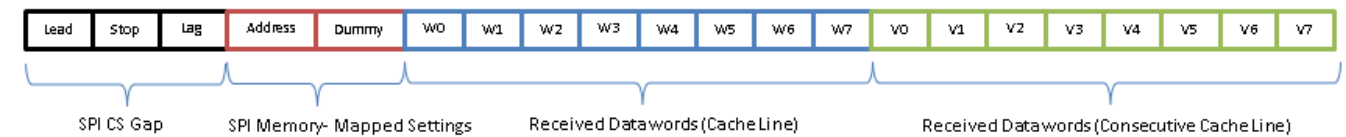


Figure 14. 32-Byte Cache Line (Merge On, Prefetch On, Command Skip On, Word 0 of Consecutive Cache Lines Requested by the Processor)

SPI Memory-Mapped Configuration

The code example provided in Listing 1 illustrates how to configure SPI2 flash in Quad-I/O Fast Read mode under memory-mapped operation.

```

/* Setup SPI -
   - Quad - 0xEB
   - SPI Clock Frequency = SCLK frequency
   - LAG = LEAD = 1, STOP = 3
   - Fast mode enabled
   - Memory-Mapped Mode enabled
   - Merge enabled
   - Wrap disabled
   - No of ADDR pins = 4
   - No of CMD pins = 1
   - Dummy = 5 bytes
*/
// Configure the system clock to 100MHz for high performance

void Set_MemMap_XIP(void)
{

/* Disable the SPI */
*pREG_SPI2_CTL = 0;

/* Putting the SPI into single bit mode and enabling SPI again */
*pREG_SPI2_CTL = 0x3;

/* Send the Continuous mode reset command 0xFF to terminate XiP mode */
spi_exit_xip();

/* Disable SPI and make sure we are out of Memory-Mapped mode */
*pREG_SPI2_CTL = 0; __DSB();
*pREG_SPI2_MMRDH = 0; __DSB();

/* Program flash with Dummy cycles: Dummy cycles can be programmed using bit[3:0]
of Status Register 3 of flash device. Additionally, if using Quad mode, user must
program bit 1 of Status Register 2.*/
ProgramFlash_Dummy_Quad();

/* Program baud rate */
*pREG_SPI2_CLK = 0x00;

/* Program delay register with lead and lag times */
*pREG_SPI2_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 ) );

/* Program SPI control register */
*pREG_SPI2_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 |
        ((0 << BITP_SPI_CTL_CPHA) & BITM_SPI_CTL_CPHA) |
        ((0 << BITP_SPI_CTL_CPOL) & BITM_SPI_CTL_CPOL) );

/* Program transmit control register */
*pREG_SPI2_TXCTL = (BITM_SPI_TXCTL_TTI | BITM_SPI_TXCTL_TEN);

/* Program receive control register */
*pREG_SPI2_RXCTL = (BITM_SPI_RXCTL_REN);

/* Program Memory Map Read Header register */
*pREG_SPI2_MMRDH = (
    ((0xEB << 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) |
    ((5 << BITP_SPI_MMRDH_DMYSIZE) & BITM_SPI_MMRDH_DMYSIZE) |
    ((0xFF << BITP_SPI_MMRDH_MODE) & BITM_SPI_MMRDH_MODE) |
    ((1 << BITP_SPI_MMRDH_TRIDMY) & BITM_SPI_MMRDH_TRIDMY) |
    ((1 << 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) );

/* Program Memory-mapped Top register */
*pREG_SPI2_MMTOP = 0x18000000 + (FLASH_BLOCK_SIZE * FLASH_BLOCK_COUNT);

/* Program SPI with slave select */
*pREG_SPI2_SLVSEL = ENUM_SPI_SLVSEL_SSEL1_HI | ENUM_SPI_SLVSEL_SSEL1_EN;
__DSB();

/* Now enable SPI */
*pREG_SPI2_CTL |= BITM_SPI_CTL_EN;
__DSB(); }

```

Listing 1. SPI Memory-Mapped Mode Programming Example

Internal Flash Memory

Device Description

The ADSP-CM40x on-chip flash is a high-performance multiple input/output serial flash memory device. It features execute-in-place (XiP) functionality, advanced write protection mechanisms, and a high-speed SPI-compatible bus interface. High-performance dual- and quad-I/O fast read instructions allow for doubling or quadrupling (respectively) the transfer bandwidth for read operations.

Memory Organization

The memory is organized as 32 64-KB blocks that are further divided into 16 sectors each (512 sectors in total). The memory can be erased one 4-KB sector at a time, one 64-KB block at a time, or as a whole.



Refer to the hardware reference manual^[3] and datasheet^[2] for more information on the internal flash device.

Operating Protocols and Commands

The standard SPI protocol is enhanced by dual and quad operations. In addition, the dual-SPI and quad-SPI protocols improve the data access time and throughput of a single I/O device by transmitting addresses, mode bytes, dummy cycles, and data across two or four data lines, respectively. The SPI operating protocols and commands are shown in [Table 1](#).

SPI Protocols	Read Command					
	Read	Fast Read	Dual-Output Fast Read	Dual-I/O Fast Read	Quad-Output Fast Read	Quad-I/O Fast Read
	0x03	0x0B	0x3B	0xBB	0x6B	0xEB
Command Input Lines	D0	D0	D0	D0	D0	D0
Address Input Lines	D0	D0	D0	D[1:0]	D0	D[3:0]
Data Output Lines	D1	D1	D[1:0]	D[1:0]	D[3:0]	D[3:0]

Table 1. SPI Operating Protocols and Commands

XiP Mode

The XiP mode/Command Skip mode does not require the read command to be sent to the flash, improving random access time and eliminating the need to shadow code in the RAM for fast execution. Quad-I/O Fast Read and Dual-I/O Fast Read modes support XiP operation, which requires only the address to be sent.

Activate or Terminate XiP

Applications may switch to XiP because XiP provides faster memory read operations by requiring only an address to execute, rather than a command code and an address. The following is required to configure this mode:

1. Enable XiP by setting the mode bits in the `SPI_MMRDH` register to 0x20.
2. Enable the SPI module and execute a dummy read.
3. Set the `SPI_MMRDH.CMDSKIP` bit.
4. Once in XiP mode, instructions require only the address bits to execute. A command code is not necessary, and device operations use the SPI protocol that is enabled.
5. XiP is terminated by sending the Continuous Read Mode Reset command (0xFF).

XiP requires to send the XiP mode byte to the memory during the dummy clock cycles after the address has been sent, as shown in [Figure 15](#).

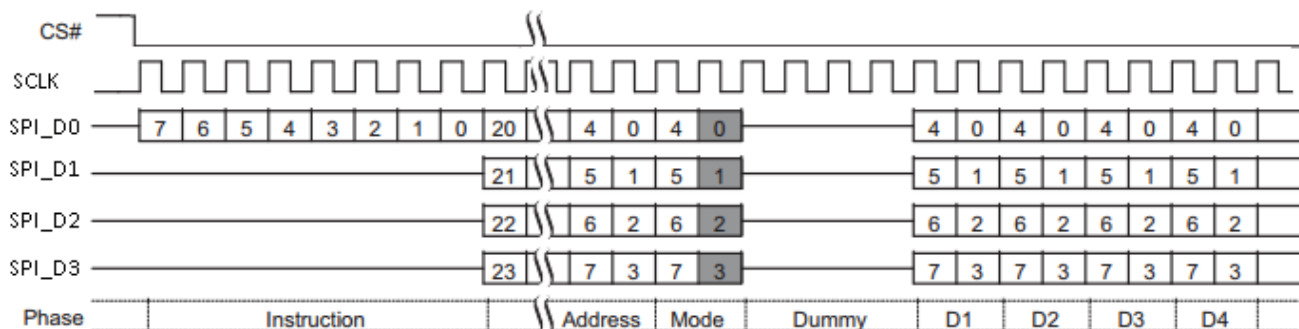


Figure 15. Sending XiP Confirmation to SPI Flash in Quad-I/O Fast Read Mode

Setting XiP Mode

XiP confirmation is sent as shown in Listing 2:

```

/* Configure the MODE bits of SPI_MMRDH register to 0x20 */

*pREG_SPI2_MMRDH = (*pREG_SPI2_MMRDH & (~BITM_SPI_MMRDH_MODE)) | ((0x20 <<
BITP_SPI_MMRDH_MODE) & BITM_SPI_MMRDH_MODE);

*pREG_SPI2_CTL |= BITM_SPI_CTL_EN;

/* Perform one dummy access to SPI flash for setting MODE bits to 0x20 */
unsigned int* pSPI_MEM = (unsigned int*) 0x18004000;

/* Set the Command Skip bit of SPI_MMRDH register */
*pREG_SPI2_MMRDH |= BITM_SPI_MMRDH_CMDSKIP;
DSB();

/* All further SPI flash accesses will be in XiP mode (command skip) */

```

Listing 2. Sending XiP Confirmation

In utilizing the above code, the programmer must ensure that the following guidelines are respected:

1. The cache line size is set to 32 bytes.
2. The code routine is executed from internal SRAM.
3. A cache clear is performed before this routine is called.



The code in Listing 2 also assumes that the mode byte was previously configured to a value other than 0x20.

Dummy Cycles

For read operations, the internal SPI flash supports a variable number of dummy clock cycles based on the read mode selected and the frequency of operation, as shown in Table 2. Note that `SPI_MMRDH.DMYSIZE` must be programmed in terms of bytes relative to the flash requirement. Therefore, there may be cases where the SPI port provides more dummy cycles than required. If the number of dummy cycles is insufficient for

the operating frequency, invalid data will be read. For the purpose of setting XiP mode, a mode byte is required to be sent after the address is sent, and the dummy cycles follow the mode bytes. Note that some values of dummy cycles are not applicable, as the SPI controller is configured in terms of bytes (dummy cycles plus mode bytes). For example, the Quad-I/O Fast Read command sends the dummy and mode bytes over four data lines, so it takes two cycles to send a byte. Hence, for the Quad-I/O Fast Read command, only dummy cycle configurations which are multiples of two cycles are allowed. The number of dummy cycles can be set by configuring the Status Register-3 of the flash in volatile mode.

Number of Dummy Clock Cycles	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)
1	NA	NA	NA	NA	NA
2	NA	NA	NA	NA	59 (2 mode and 2 dummy cycles)
3	NA	NA	NA	NA	NA
4	NA	NA	100 (4 mode and 4 dummy cycles)	NA	78 (2 mode and 4 dummy cycles)
5	NA	NA	NA	NA	NA
6	NA	NA	NA	NA	95 (2 mode and 6 dummy cycles)
7	NA	NA	NA	NA	NA
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)

Table 2. Internal Flash Dummy Cycles

Interfacing External Flash Memory Devices

External flash memory may be interfaced with the controller through the SPI0 interface. To improve performance, the memory accesses through SPI0 must be cached. Since the cache controller works only with MEM-X and MEM-Y, when performing memory-mapped accesses from SPI0, SPI2 or SMC-interfaced memory, the application has to manually manage configuring the MEM-X and MEM-Y interfaces between SPI0 flash, SPI2 flash, and the SMC. This is done by setting the base address field of the M4P_CACHE_MEMX and M4P_CACHE_MEMY appropriately, as shown in [Listing 3](#).

```
#define SPI0_MEM 0x50000000
#define ASYNC_MEM 0x60000000
#define SPI2_MEM 0x18000000

void SetupMemXMemY(unsigned int MEMX_TRANS, unsigned int MEMY_TRANS)
{
    *pREG_M4P0_CACHE_MEMX = MEMX_TRANS | 0x3;
    __DSB();

    *pREG_M4P0_CACHE_MEMY = MEMY_TRANS | 0x3;
    __DSB();
}

//setup for SPI0 and SPI2
SetupMemXMemY(SPI2_MEM, SPI0_MEM);
```

Listing 3. Programming MEM-X and MEM-Y Registers

Performance Enhancement Features in the SPI and Cache Controllers

Features

Merge

With Merge set in the SPI controller, linear accesses will be merged, resulting in high throughput. The SPI controller will merge cache line bursts if the addresses are contiguous. The address and read command will not be sent, and consecutive cache line data will be merged.

Speculative Operations

The Speculative bit in the cache controller enables speculative fetching, and this bit must be enabled for proper merge operations to take place. This enables the cache controller to start work on the next access from the core (even when stalled during the current access). This is enabled by default in the cache controller.

Pre-emptive

Pre-emptive fetching is a technique which only helps when the cache is shared, so it is useful when ICODE and DCODE could conflict with one another. If enabled (disabled by default), non-stalled banks fetch ahead from cache arrays and buffer the data.

Wrap

Wrap mode is a specific feature supported by most flash devices. It enables the host processor to ask for the critical word first in a cache line. If enabled, the core is delivered with the critical word it requested as soon as possible. When using this mode, the cache controller, SPI controller, and flash device all must be pre-programmed to support this mode.

Command Skip

Command Skip (True XiP or Continuous Read mode) is a method where higher throughput is possible by sending only the address (without the overhead of sending any commands). To support this, appropriate mode bytes should be sent, and the `SPI_MMRDH` register must be configured appropriately.

Pre-fetch

There is a pre-fetch unit present in the cache that aims to predict and generate sequential cache line fills for the ICODE interface during the fill read-return of the current cache line. The hardware automatically generate a pre-fetch request after a particular count of words are read. The count is configurable in the `REG_M4P0_CACHE_CFG` register.

Choosing the Correct Mode of Operation

It is recommended that application performance is first evaluated using the default modes of operation in the `M4P_CACHE_CFG` register, which is optimal for most cases. Further evaluation of the software can be made with other features, such as utilizing the Pre-emptive, Pre-fetch and Wrap modes. Based on this analysis, the optimum settings can be selected. Some ideas are explained below to help choose the right mode.

Line-Wrap vs. Line-Base-First Modes

In Line-Base-First mode, the line is always fetched starting at offset 0 (line base). In Line-Wrap mode, the critical word is returned first while the rest of the line is fetched with wrap-around addressing. Since Line-Wrap mode implies non-linear sequencing, merging is not possible. Usually, Line-Base-First mode will be faster because merging is possible.

Merge vs. Non-Merge Modes

Since merging is performed automatically, it should be avoided when predictable access time is required, when debugging, and when using Wrap mode.

Pre-emptive Mode

Pre-emptive fetching may provide benefits when there is heavy contention for the same cache ways, banks, and stripes. However, it may result in greater power consumption if the processor never requires data that is pre-emptively acquired.

Cache Line Size

A larger cache size setting is always desirable, provided the flash memory supports the selected size. At the same time, a large cache line can cause Interrupt Service Routine (ISR) execution to be delayed because the cache has to read the complete burst of the cache line prior to executing the cached instructions.

Pre-fetch Mode

The pre-fetch mode is enabled using the `REG_M4P0_CACHE_CFG.PF` bits. A setting of b#000 disables the pre-fetch unit, while a setting of b#111 configures it for the most aggressive prediction, where the next line is always pre-fetched regardless of the number of received words. An intermediate value tells the pre-fetch unit to begin pre-fetching after a set value of received words.

Enabling and Disabling of Modules

Enabling

Cache Controller

The cache controller is enabled by default with the following settings: 32-byte cache line, line-base first mode, speculative mode, and shared cache. The pre-fetch feature of the cache controller is enabled by setting the `REG_M4P0_CACHE_CFG.PF` bits. By default, the pre-fetch feature is disabled.

SPI Controller

For SPI Master Boot mode, the code that resides in the boot ROM space will enable the SPI controller in Memory-Mapped mode with quad-I/O data enabled. Additionally, 10 dummy cycles are configured, which is required for the highest frequency of operation. The boot kernel, however, will not enable XiP mode, which the user must change as needed. To configure XiP mode, refer to the associated example code^[1].

Once executing user code, the SPI clock frequency must be programmed to make it the same as the System Clock (`SCLK`) frequency (via the `SPI_CLK` register). Since the boot ROM space operates in Active mode, it is the user's responsibility to program the Clock Generation Unit (CGU) for Full-On mode. With these two procedures in place, the access speed from flash memory can be as high as 100 MHz, provided `SCLK` is also programmed to the same frequency.

Flash Device

For special modes such as Wrap, the flash device must be programmed manually by the user setting bit 4 of Status Register 3 in the volatile mode. By default, flash memories operate in sequential address format, so the internal flash device has to be programmed for 8-, 16-, or 32-byte Wrap modes using bits 5 and 6 of Status Register 3 in the volatile mode. Users must ensure that the SPI and cache controllers are also configured with the same settings. Additionally, the wrap boundary alignment must be the same as the cache line size and can be programmed for 8-, 16-, or 32-byte. Refer to the hardware reference manual^[3] for more details regarding the flash device.

Disabling

Cache Controller

Typically, there is no need to disable the cache controller, as that will result in performance degradation.

SPI Memory-Mapped Mode

SPI Memory-Mapped mode needs to be disabled if a write operation is to be performed (such as for flash programming). This can only be done using the SPI controller memory-mapped registers.

Terminating XiP Mode

In certain cases, it may be desired to terminate XiP mode. For instance, all flash programming has to occur through non-memory-mapped mode only (via MMRs). If the flash device has been configured for XiP mode, programming of the flash in non-memory-mapped mode will cause a bus conflict, as the flash device will not expect any commands from the processor. Thus, terminating XiP mode is required, which is achieved by executing the Continuous Mode Reset command (0xFF).

Performance Analysis

Analyzing Code Performance Using Cache

The overall performance of the application is a factor of the speed/efficiency of SPI-based reads, as well as the cache miss-to-hit ratio. The modes of operation described above will help achieve the right performance required for the application. The choice may involve weighing algorithm performance between cache-based accesses and internal SRAM accesses (typically used for executing critical code).

Cache counters may be employed to understand cache utilization in detail. This is an effective tool when there are large numbers of misses and hits (the reused functions always compete within the 16 KB space). As a basic scenario, the I-Code Reference Counter (M4P_CACHE_IREF), I-Code Miss Counter (M4P_CACHE_IMISS) and I-Code Fill Counter (M4P_CACHE_IFILL) will first be explored.

The M4P_CACHE_IFILL fill counter always shows the number of cache line fills read from SPI flash memory. The M4P_CACHE_IREF reference counter shows the number of word accesses requested from the core. The M4P_CACHE_IMISS miss counter reflects the misses that occurred during the above requests, and the number of hits can be extrapolated (M4P_CACHE_IREF value – M4P_CACHE_IMISS value). Note that the Cortex-M4 core can request more words through its pre-fetch unit, so there would be a few additional requests reported by the counters when branches are not taken, which is not significant when looking at the overall performance.

Figure 16 is a screenshot taken while running the CMSIS FFT on the ADSP-CM40x mixed-signal control processor in a cache-testing-based environment, captured using the IAR Embedded Workbench for ARM development tools^[4]. A Cycle/Sys-Tick counter, as well as an output comparison, would add a sanity-check for any tests performed, especially when dealing with an external device connected to the SPI0 or SMC interfaces.

In this example, counters are reset before every subsequent run. Reference counter values are closely related to the actual core cycle count (depending on code layout). It is also evident that Miss/Fill counters show zeroes for all subsequent runs after the first cache miss case because the entire program could fit inside cache with no further cache misses. In a larger application, there may be a mix of misses and hits, depending on cache line replacement.

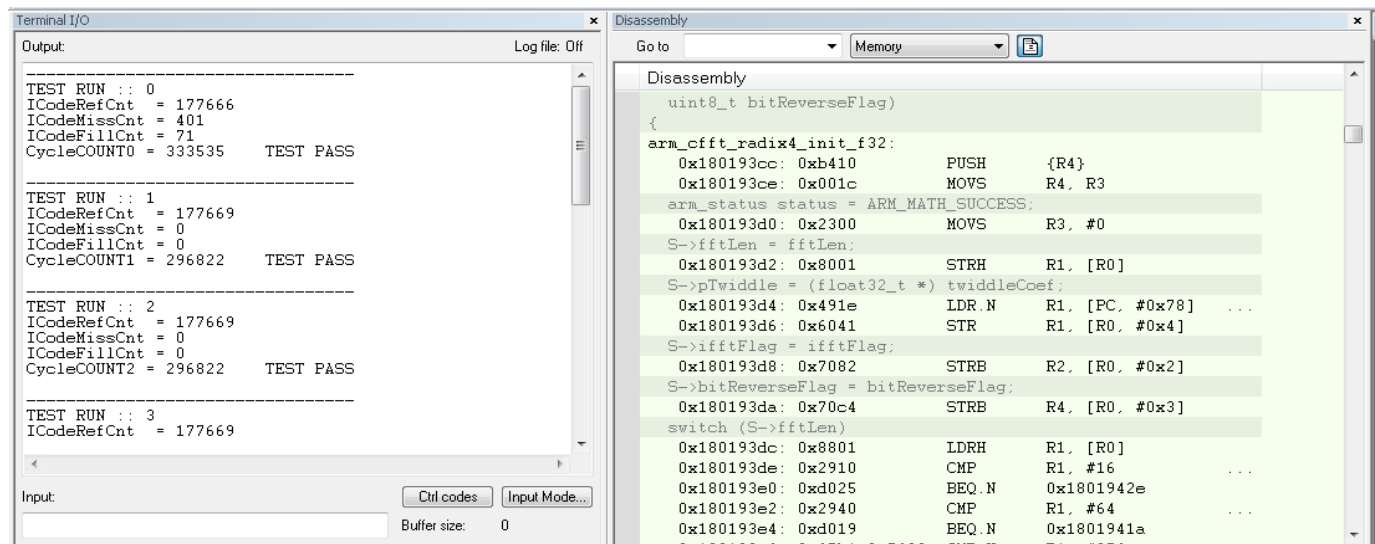


Figure 16. Analyzing Cache Miss and Hit Measurements

Throughput Benchmarks

Throughput Comparison Across Various Memory Blocks

Table describes the sustained performance of various memory pathways in the system. It is important to note that this measures the data transfer bandwidth of the SPI in a benchmark composed of 4000 bytes of linear code (so that cache accesses are not intended to *hit*).

32-Bit Instructions Sequential Read Accesses	Cache Usage	CCLK: 225 MHz SCLK: 100 MHz	CCLK: 240 MHz SCLK: 96 MHz	CCLK: 240 MHz SCLK: 80 MHz	CCLK: 200 MHz SCLK: 100 MHz
		Throughput (MB/s)	Throughput (MB/s)	Throughput (MB/s)	Throughput (MB/s)
Internal SRAM	Unused	897.31	957.13	957.13	797.61
Internal SPI2 Flash	1st run (code not cached)	48.96	47.02	39.19	48.95
Internal SPI2 Flash	2nd run (code cached)	893.74	953.33	953.33	794.44
External Async SRAM	1st run (code not cached)	99.49	95.52	79.63	99.46
External Async SRAM	2nd run (code cached)	893.74	953.33	953.33	794.44

Table 3. Throughput Benchmarks of Various Memory Blocks



This table should not be taken as a benchmark for the performance of typical application code, as the cache miss-to-hit ratio is important to include when estimating the overall application performance. These measurements reflect how fast the interfaces can read data in from the memory upon a cache miss. The cache pre-fetch unit was enabled while performing these throughput measurements.

Comparison Among Modes

Typically, it's desirable to always enable the XiP and pre-fetch features in internal flash memory space. However, it is important to understand how these modes affect efficiency. The data summarized in Table 4 and depicted graphically in Figure 17 through Figure 19 was obtained from an experiment that exercised the pre-fetch and XiP features during execution of a 4000-byte application comprised of 3996 bytes of Cortex-M4 linear instructions and a return instruction mapped to the SPI2 flash memory. In all cases, the SPI clock frequency is the same as the SCLK frequency, and the SPI, cache, and flash conditions used for the analysis were as follows:

- Cache enabled with 32-byte cache lines
- SPI uses Quad-I/O Fast Read command
- SPI configured for one command line, four address lines, and four data lines
- SPI Merge mode enabled
- Cache speculative access enabled
- Optimal dummy cycles configured based on SPI clock (same as SCLK) frequency

For the four possible combinations of XiP on/off and pre-fetch on/off, the following settings were modified between tests:

- XiP mode toggled on or off using the sequence in the associated code example
- Pre-fetch toggled on or off by setting the REG_M4P0_CACHE_CFG.PF bits to either b#000 (off – pre-fetch disabled) or b#111 (on – always pre-fetch the next cache line)

Note that a similar performance analysis may be conducted on external devices.

Command = 0xEB Mode = Fast Quad-I/O	CCLK/SCLK (MHz)	XiP On (MB/s)	XiP Off (MB/s)
Pre-fetch Off	225/100	35.03	32.18
	240/96	34.77	31.87
	240/80	29.66	27.13
	200/100	34.64	31.86
Pre-fetch On	225/100	48.96	48.91
	240/96	47.02	46.97
	240/80	39.19	39.16
	200/100	48.95	48.90

Table 4. SPI Flash Throughput Summary for Quad-I/O Fast Read Mode

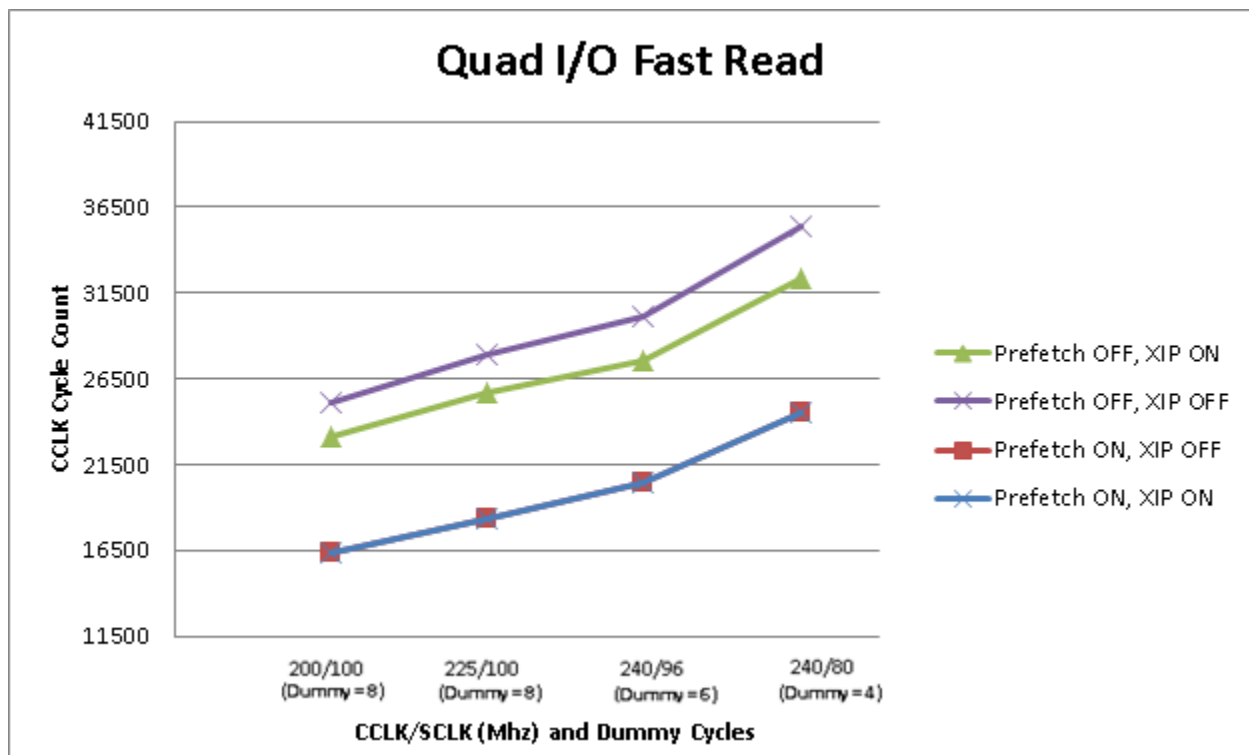


Figure 17. Quad-I/O Performance Vs. Clock Frequencies and Dummy Cycles

Figure 18 shows a zoomed version of the quad-I/O, pre-fetch off case with respect to XiP on and off measurements.

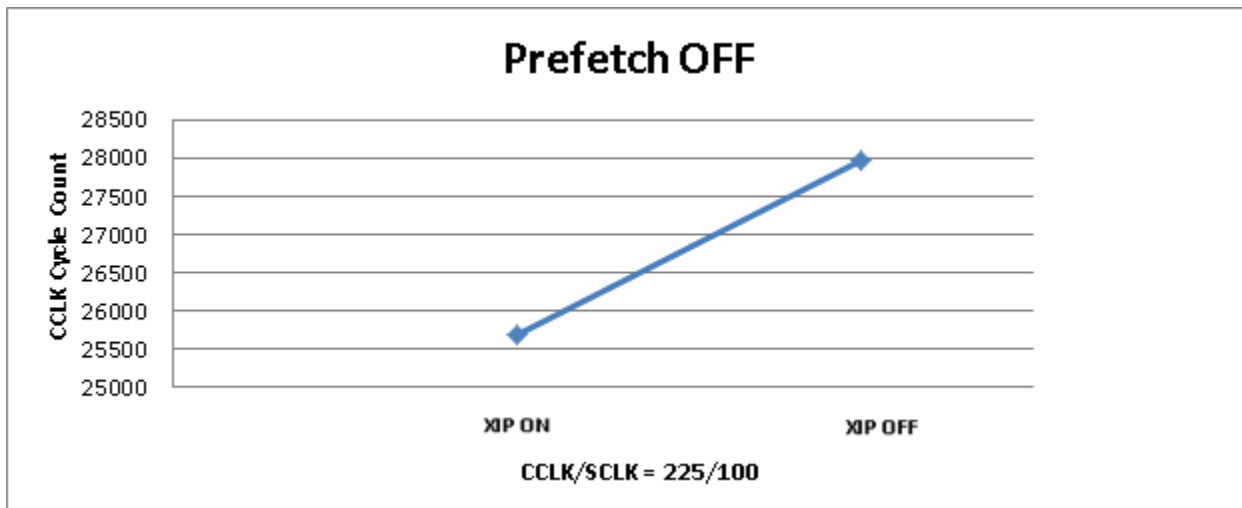


Figure 18. Quad-I/O Performance Difference with Pre-fetch Off (XiP On Vs. XiP Off)

For the quad-I/O with pre-fetch on case, the performance gain between XiP on vs. XiP off is very narrow (Figure 19). With Pre-fetch and Merge enabled, continuous merging of data occurs. The read command, address, and dummy bytes are skipped, and the read data is always merged. Hence, the difference between XiP on and XiP off is only a read command of eight SPI clock cycles. Even with Pre-fetch and Merge enabled causing continuous merging of data, the hardware introduces a delay of 1 SPI clock cycle between two cache lines.

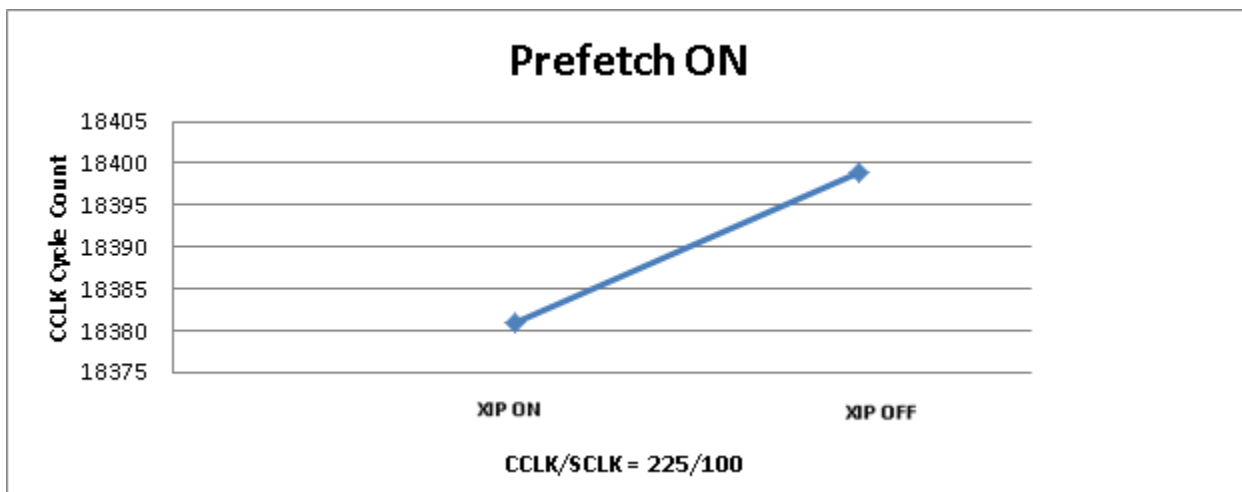


Figure 19. Quad-I/O Performance Difference with Pre-fetch On (XiP On Vs. XiP Off)

Appendix A: SPI Flash Accesses Oscilloscope Plots

XiP Mode Off (CMDSKIP in SPI Controller Off)

In Figure 20, 32-bit cache lines are read by the core. In this case, a command is sent to flash for every access, thus adding additional overhead in throughput.

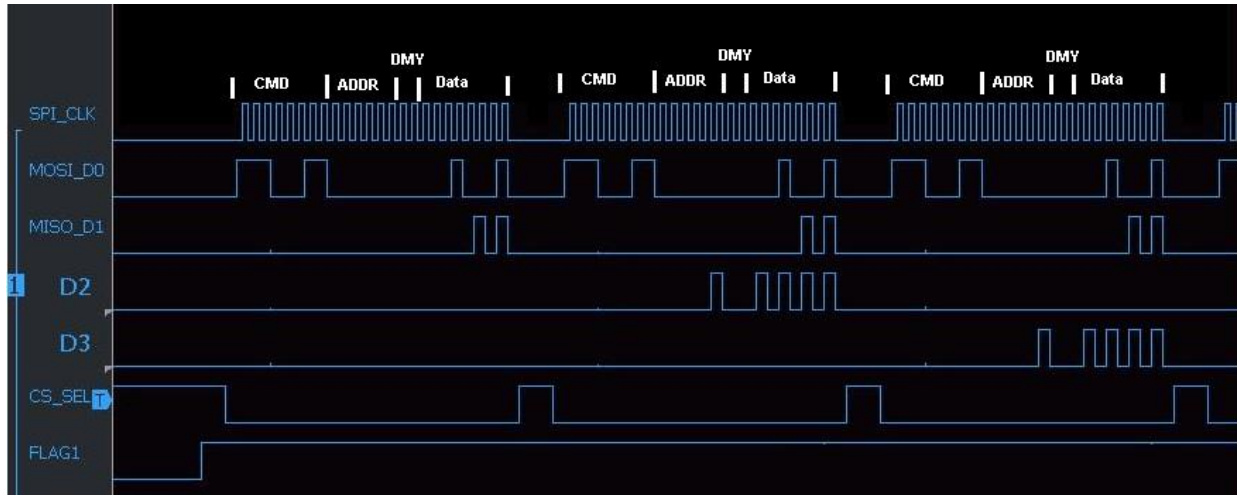


Figure 20. Scope Plot for XiP Mode Off

XiP Mode On (CMDSKIP in SPI Controller On)

In the same 32-bit cache line scenario, both the SPI controller and the flash device are now pre-configured for Command Skip mode, such that only the address needs to be sent for every access, as shown in Figure 21.

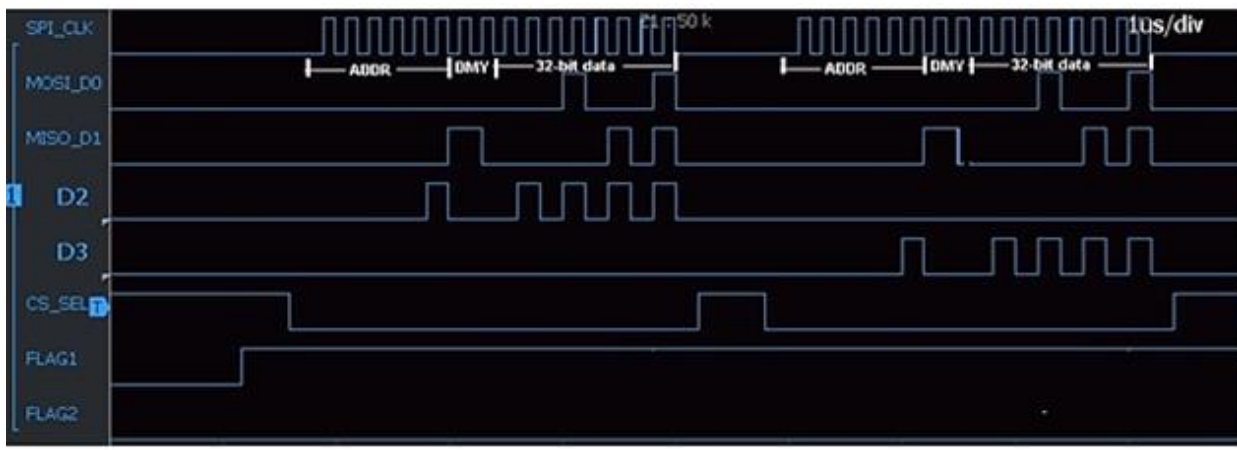


Figure 21. Scope Plot for XiP Mode On

Line Base Access (Wrap Disable)

Let's now assume that a cache line is 64 bits (i.e., each transfer consists of two 32-bit words) and the core requested the second word. In Line Base Access mode, the entire cache line is first read out, with the requested data provided only when it is made available (in this case, only after the first 32-bit data), as shown by the FLAG1 GPIO signal in Figure 22. FLAG1 is toggled after the access request is made by the core. In this example, cache line data read from the flash is {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07}.

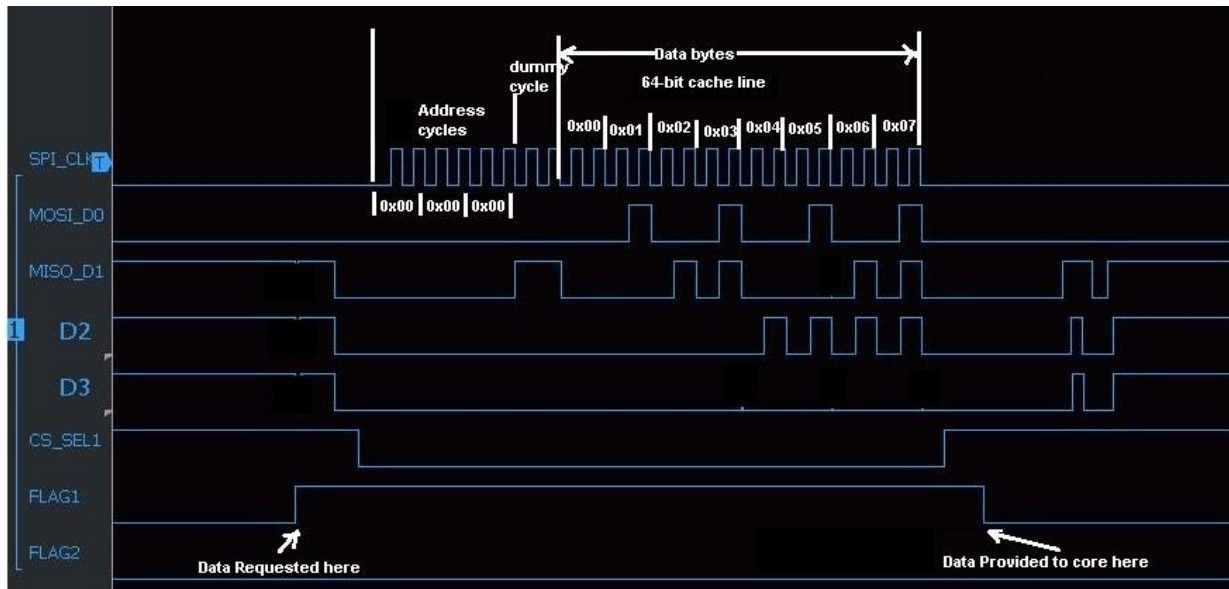


Figure 22. Scope Screenshot for Line Base Access

Critical Word First Access (Wrap Enable)

Working from the previous example, if Wrap mode is now enabled, the SPI controller will fetch the critical word first (in this case, it is the second 32-bit word in the 64-bit cache line). As shown in Figure 23, the FLAG1 pin toggles right after the critical 32-bit data is presented to the core. Also, after the second 32-bit word is received, the access is wrapped around to receive the first 32-bit word.

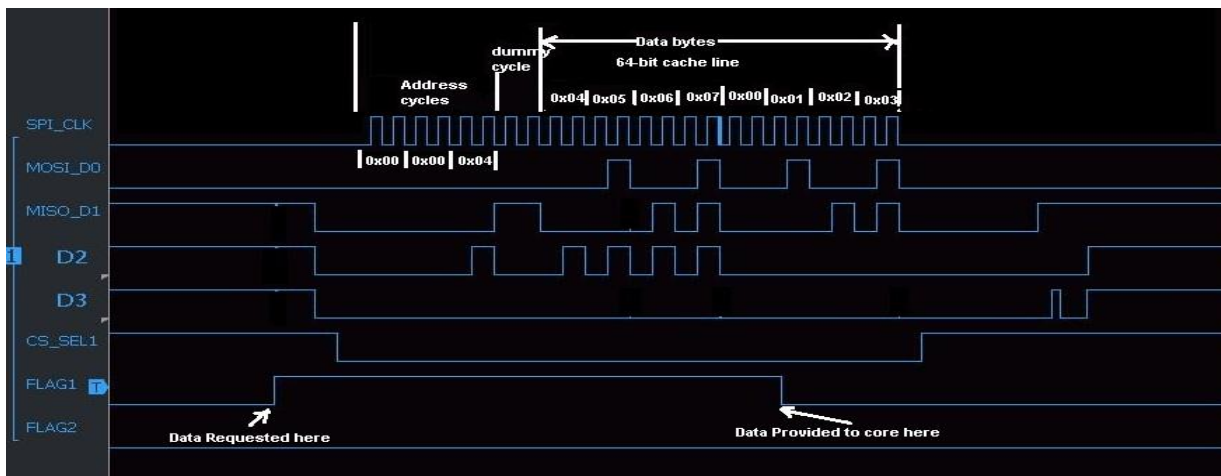


Figure 23. Scope Screenshot for Critical First Access

References

- [1] *Associated ZIP File for EE-363: ADSP-CM40x Serial Flash Throughput Benchmarks*. Rev 2, December 2015. Analog Devices, Inc.
- [2] *ADSP-CM40x ARM Cortex-M4 Mixed-Signal Control Processor Data Sheet*. Rev A, October 2015. Analog Devices, Inc.
- [3] *ADSP-CM40x Mixed-Signal Control Processor with ARM Cortex-M4 Hardware Reference*. Rev 1.0, November 2015. Analog Devices, Inc.
- [4] *ADSP-CM40x ARM Cortex-M4 Mixed-Signal Control Processor Anomaly Sheet*. Rev H, December 2015. Analog Devices, Inc.
- [5] *ADSP-CM40x Enablement Software Package*. Rev 2.1.0. Analog Devices, Inc.
- [6] *IAR Embedded Workbench for ARM* (<http://www.iar.com/>). 6.60. IAR Systems AB.
- [7] *IAR C/C++ Compiler, Compiling and Linking Manual* (<http://www.iar.com/>). 9th Edition May 2012. IAR Systems AB.

Document History

Revision	Description
<i>Rev 2 – December 7, 2015</i> <i>by Prasanth Rajagopal, Prashant Gawade, and Kritika Shahu</i>	Updated the document and example code for Silicon revision H with added security and pre-fetch features.
<i>Rev 1 – September 20, 2013</i> <i>by Prasanth Rajagopal and Prashant Gawade</i>	Initial release.