Technical notes on using Analog Devices DSPs, processors and development tools
Contact our technical support at dsp.support@analog.com and at dsptools.support@analog.com
Or visit our on-line resources http://www.analog.com/ee-notes and http://www.analog.com/processors

ADSP-2191 DSP Host Port Booting

Contributed by Michael Kuegler

Rev 1 – January 23, 2004

1. Introduction

The Analog Devices ADSP-2191 DSP provides a Host Port Interface (HPI) that can be used by a Host to access the internal and external memory and I/O space of the DSP via an 8- or 16-bit bus. It is also possible to boot the DSP via the HPI from the Host. The ADSP-2191 DSP is always passive during Host Port accesses; therefore it is called "Target" in this context. This application note:

- Demonstrates how to access the Host Port from an FPGA or PLD, using VHDL as the design language.
- Provides an example of a software boot loader, written in C, for booting the DSP via the Host Port from a microcontroller or any other processor.

To achieve this, a test system has been designed using another ADSP-2191 DSP as Host processor and an Altera MAX7000 PLD between the two DSPs.

The designed hardware is described in chapter 3.

Chapter 4 covers one of the two main objectives: the VHDL design.

Chapter 5 describes the boot process and the boot loader file, and Chapter 6 provides details about the software boot loader.

Chapters 4 and 6 are independent of each other; it is not necessary to know the internal function of the PLD to use/understand the software boot loader. The boot loader is written in generic C and accesses the PLD using a set of low-level interface functions (which have to be adapted to the underlying hardware).

Screenshots from a logic analyzer, showing Host Port transfers during the boot process, are found in Chapter 7.

Chapter 2 outlines the HPI features and configuration used in this design.

This document does not provide an introduction to the Host Port, and it does not explain all possible configuration parameters.

A complete description of the HPI and the Host Port I/O configuration registers can be found in [1]. For detailed information about pin-out and timing requirements, refer to [2].

The source code for software boot loader and VHDL design are not included in this document; they may be downloaded from the Analog Devices Web site.

2. Overview of Host Port

2.1. Pins

Table 1 lists the pins that make up the Host Port.

16 multiplexed address and data lines are available for the HPI: HAD[15..0].

The default setting for the bus width after reset is 8 bits for data cycles, but the design described in this note uses all 16 lines.

Address cycles always use the full bus width (and HA16), even if the bus is configured for 8 bits.

Two memory select lines provide the Host access to internal and external memory (~HCMS) and I/O space (~HCIOMS).

The HALE pin can be configured to function in Address Latch Enable (ALE) mode or Address Cycle Control (ACC) mode. For ACC mode (as used in this application), it must be driven high during reset.

The polarity of the strobe signals ~HWR and ~HRD must also be configured. They are driven high during reset, setting their active state low.

HACK and HACK_P determine the functionality and polarity of the acknowledge signal during reset. HACK can be configured to act either in Acknowledge mode (ACK) or Ready mode (READY). HACK_P is driven high, and HACK is driven low during reset, setting up HACK in ACK mode, active high.

HALE, ~HWR, ~HRD, HACK, and HACK_P have to be driven during, and for 10 peripheral clock cycles (HCLK) after reset.

Pin Name	Input/Output	Pin Function
HAD[15..0]	I/O/T	Host Port Multiplexed Address and Data Bus
HA16	I	Host Port Most Significant Address Line
~HCIOMS	I	Host Port I/O Space Select
~HCMS	I	Host Port Memory Select
HALE	I	Host Port Address Latch Strobe
~HWR	I	Host Port Write Strobe
~HRD	I	Host Port Read Strobe
HACK	I/O	Host Port Acknowledge
HACK_P	I	Host Port Acknowledge Polarity

Table 1. Host Port pins

2.2. Access Modes

Two modes provide the Host access to memory and I/O space of the ADSP-2191 DSP: Direct mode and DMA mode.

The Host boot loader (described in Chapter 6) uses DMA transfers to write the content of the loader file into the Target memory. Host Port DMA behaves like any other DMA channel on the ADSP-2191 DSP: the transfer direction can be configured, Autobuffer and Descriptor mode are available, and an interrupt can be generated after completion of the data transfer. The software boot loader uses Autobuffer DMA for the boot process because it is suitable for transferring large blocks of data without having to transmit the address for every word. The disadvantage of DMA mode is that it has to be configured, which requires several cycles. For example, it takes five write accesses in Direct mode from the Host to set up one Autobuffer DMA transfer. If only one or two memory locations need to be written to or read from, Direct mode is more suitable. The Host boot loader includes examples of read and write accesses to internal memory and I/O space in Direct mode, for which logic analyzer screenshots are shown in Chapter 7.

Host Port DMA mode is available only for accesses to memory, not I/O space.

2.3. Host Port Registers

Several I/O configuration registers, as listed in Table 2, are dedicated to the Host Port.

The Host Port Configuration Register, *HPCR*, must be set up before any other access. It contains configuration bits for various access parameters, and some status bits, reflecting the functionality of HACK and the polarity of control signals sensed during reset.

For the chosen configuration (modes ACK and ACC, low active strobe signals), the reset value of *HPCR* is 0x0F00. To enable 16-bit bus width, the value 0x0F01 is written into this register by the Host after the Target has come out of reset.

Before a read or write is done in Direct mode, the data type (16- or 24-bit) and the memory page should be set to the correct values by accessing the Host Port Direct Page Register, *HPPR*.

Registers starting with *HOSTD_* are used to configure Host Port DMA transfers. Every DMA transfer starts with the configuration of the appropriate I/O registers (*HOSTD_XXX*); after that, only data are transmitted or received – no addresses need be written.

ⓘ The access to I/O space registers is not affected by the setting of HPPR.

Register Name	DSP Address	Host Address[1]	Register Function
	IOPG:Address		
HPCR	0x07:0x001	0x1C01	Host Port Configuration
HPPR	0x07:0x002	0x1C02	Host Port Direct Page
HPDER	0x07:0x003	0x1C03	Host Port DMA Error
HPSMPHA	0x07:0x0FC	0x1CFC	Host Port Semaphore A
HPSMPHB	0x07:0x0FD	0x1CFD	Host Port Semaphore B
HPSMPHC	0x07:0x0FE	0x1CFE	Host Port Semaphore C
HPSMPHD	0x07:0x0FF	0x1CFF	Host Port Semaphore D
HOSTD_PTR	0x07:0x100	0x1D00	Host Port DMA Pointer
HOSTD_CFG	0x07:0x101	0x1D01	Host Port DMA Configuration
HOSTD_SRP	0x07:0x102	0x1D02	Host Port DMA Start Page
HOSTD_SRA	0x07:0x103	0x1D03	Host Port DMA Start Address
HOSTD_CNT	0x07:0x104	0x1D04	Host Port DMA Word Count
HOSTD_CP	0x07:0x105	0x1D05	Host Port DMA Chain Pointer
HOSTD_CPR	0x07:0x106	0x1D06	Host Port DMA Chain Pointer
HOSTD_IRQ	0x07:0x107	0x1D07	Host Port DMA Interrupt

Table 2. Host Port configuration registers

[1] These addresses are **not** identical to the values listed in Appendix B, Table B-1, in [1], since the required address generation is done in the PLD (left shift of actual address by 1 bit for I/O access).

3. Hardware

The hardware used as the test platform consists of a Host (ADSP-2191 DSP), a Target (ADSP-2191 DSP), and an Altera PLD (see Figure 1). The Host can be any kind of processor, microcontroller, or other programmable device. An ADSP-2191 DSP was chosen because it is assumed that the reader is familiar with the architecture of this DSP, so that there should be no difficulty understanding the system on the Host side.

Two ADSP-2191 EZ-KIT Lite™ boards serve as platforms for the Host and Target. The PLD resides on a customized PCB. Figure 2 shows the connection of the three boards; the schematic diagram can be found in Appendix A.

The External Port on the Host is connected to the Host Port on the Target through a PLD. Besides the two interfaces, two additional signals are required: an error signal from the PLD to the Host, and a reset signal controlled by the Host. This reset signal is connected to the voltage supervisor circuit on the Target board and the PLD. It allows the simultaneous reset of Target DSP and PLD.

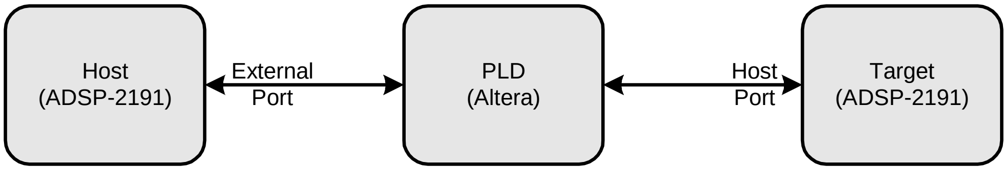


Figure 1. System overview

The loader file, which is transferred to the Target by the software boot loader, is stored in the on-board Flash on the Host EZ-KIT Lite.

Two sets of jumpers, which are not shown in Figure 2, configure the system:

On the Target EZ-KIT Lite board, Pins OPMODE, BMODE0 and BMODE1 have to be set to Host booting, as described in [4].

Jumper	Position	Function
J1	1-2*	Mode for HACK; default is ACK
	2-3	(see ADSP-2191 HRM for details)
J2	1-2*	~HRD is low active
	2-3	~HRD is high active
J3	1-2*	~HWR is low active
	2-3	~HWR is high active
J4	1-2	HACK is low active
	2-3*	HACK is high active
J5	1-2*	HALE acts in ACC mode
	2-3	HALE acts in HALE mode

* default setting

Table 3. Jumper settings for Host Port configuration

Five jumpers on the customized PCB configure the Host Port, overriding the settings on the Target board (smaller resistor values). Do not change the default values as shown in Table 3, since the PLD is designed to interact with the Host Port in modes ACK and ACC, with low active strobe signals.

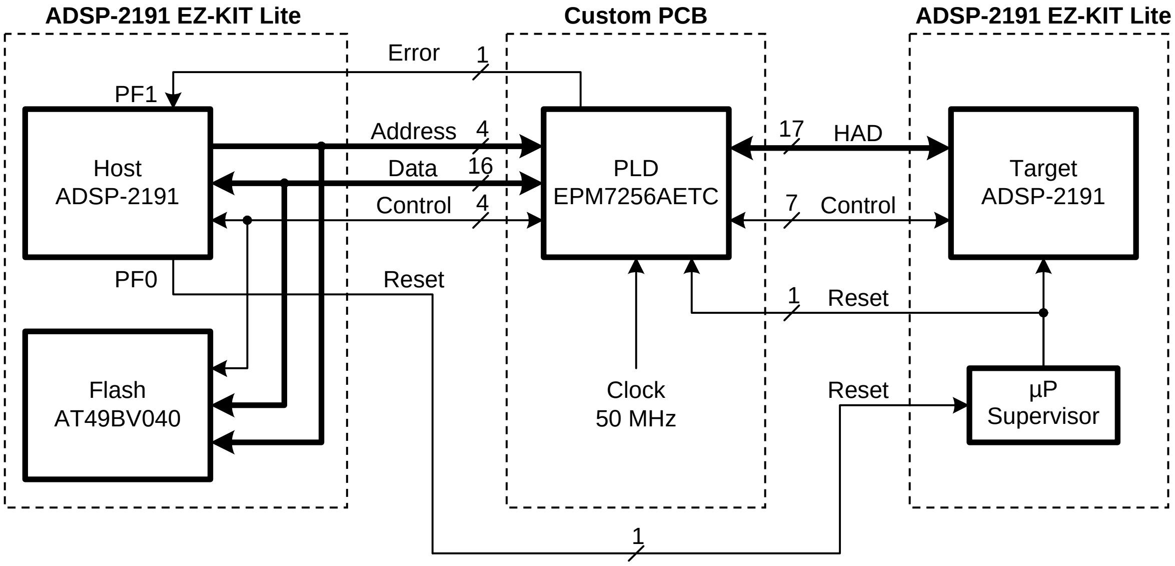


Figure 2. Hardware block diagram

4. VHDL Design

4.1. Overview

Since the External Port on the Host and Host Port on the Target are not completely compatible with each other, additional logic is required. This logic is implemented in an Altera MAX7000 PLD, which resides between Host and Target on a customized PCB.

The design generates timing for Host Port modes ACC and ACK.

The main building blocks of the VHDL design is shown in Figure 3. A graphical representation of the top-level design together with the pin assignment is found in Appendix B.

All control signals of External Port and Host Port are completely separated in the PLD. An access from the Host is detected in block “External Port interface logic”. This block will then signal to the “Host Port interface logic” block that an access to the Host Port has been requested.

No information is stored inside the PLD, so the Host is held off until the access to the Target is finished (flow-through system). This requires the Host to wait for the PLD to acknowledge that the access can be finished.

The data are transferred through block “Data Path”, in which drivers are enabled/disabled according to the direction of the data flow (read or write).

The “Watchdog” block contains logic for error detection.

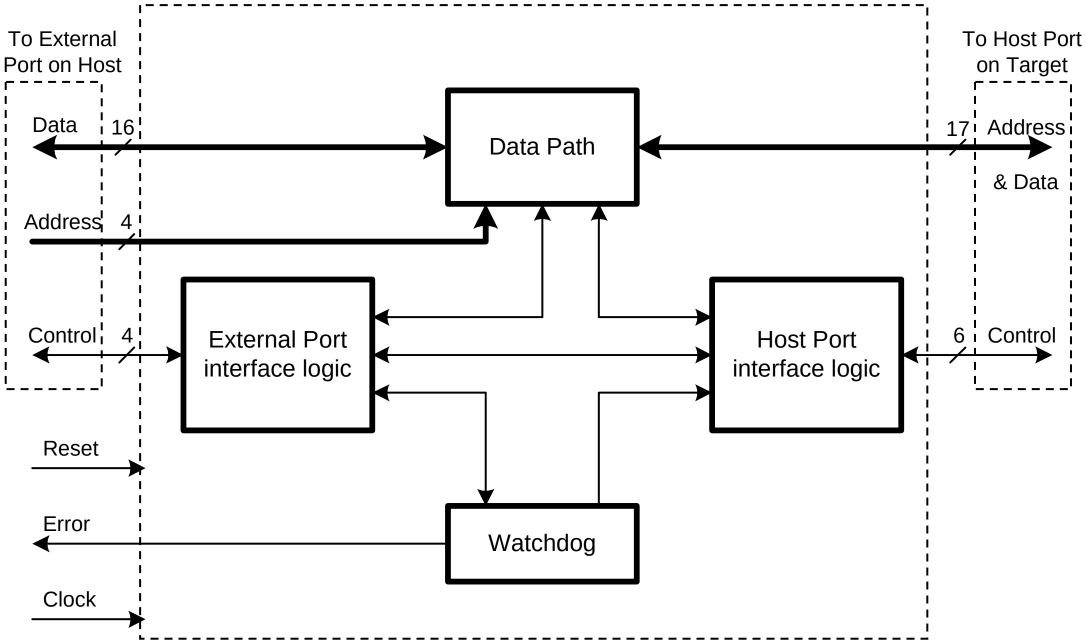


Figure 3. PLD block diagram

4.2. Interface between Host Software and PLD

Data between the Host and the Target are transmitted via a bi-directional, multiplexed address and data bus. On the Host side, the data bus of the External Port is used for this purpose.

All accesses to the Host Port can be considered to be a (sequential) combination of address cycles and/or data cycles. Information that specifies whether an access is an address cycle or data cycle, plus additional parameters are encoded in four External Port address lines. This method creates a "virtual address space", which means that there are no physical memory locations implemented in the PLD. Rather, these four address lines select the appropriate type of access.

In summary, the virtual address space on the Host side transfers information to help the PLD:

- Determine whether an address cycle or data cycle is to be performed (addresses need to be shifted on multiplexed address and data bus due to byte addressing on Host Port)
- Select correct memory type for address shifting (DM addresses need to be shifted left by 1 bit; PM addresses need to be shifted left by 2 bits)
- Assert appropriate chip select (~HCMS or ~HCIOMS)

Table 4 shows the types of accesses that can be performed through the PLD, and the addresses that have to be used to trigger the access.

For example, a write to location 4 in the PLD triggers a Host Port Address cycle, with the data on the External Port data bus transferred to bus HAD, shifted left by 1 bit, and with ~HCIOMS asserted.

A read from location 1, on the other hand, starts a Host Port Data read cycle, with ~HCMS asserted, and no shifting of data on the multiplexed bus.

One address location inside the PLD is dedicated to a special purpose: A read from address 8 returns the revision number of the VHDL design.

Some addresses in the PLD are reserved, and some combinations of access type and data direction are invalid. A read from address 0, for example, is not a valid Host Port access, since an address read cycle is not specified for the Host Port. When the Host performs an invalid access, the PLD does not signal an error, but simply acknowledges the access. The data on the External Port are not valid in such a case (PLD bus drivers are tri-stated).

PLD Address	Access type	Write / Read
0	DM Address	W / -
1	DM Data	W / R
2	PM Address	W / -
3	PM Data	W / R
4	IO Address	W / -
5	IO Data	W / R
6 - 7	Reserved	- / -
8	PLD Version	- / R
9 - 15	Reserved	- / -

Table 4. Virtual address space for performing different HPI accesses through the PLD

The External Port address lines are not used to transmit addresses to the PLD, but carry additional information required for the Host Port access.

4.3. External Port Interface Logic

When the Host starts a read or write access to the PLD, the External Port interface logic detects this, and signals to the Host Port interface logic to start a Host Port access. Some logic gates are used to assert a signal when the Host begins an access. This signal, in turn, starts a state machine (Figure 4) that simply waits until one of the following conditions is met:

- Host Port state machine finishes cycle
- Error signalled from Watchdog
- Invalid access signalled from Data Path

After that, ACK is asserted to the Host, marking the end of the access.

4.4. Host Port Interface Logic

This section describes the “core” of the VHDL design: The logic required for driving the Host Port control signals. It consists mainly of a state machine used to generate the timing, as given in [2]. The state diagram is given in Figure 5 on the following page.

Access to the Host Port in Direct mode consists of an address cycle and one or more data cycles. In DMA mode, only data cycles are performed. One complete execution of the state machine in Figure 5 triggers one address cycle or one data cycle, depending on the input values.

A complete Host Port access can be “assembled” by executing the Host Port state machine several times with the appropriate parameters.

For example, access to 24-bit Program Memory with a 16-bit bus width in Direct Mode requires one address cycle followed by two data cycles. The state machine would be started three times in this case: the first time configured for address cycle, then two times with the input parameters set to data cycle.

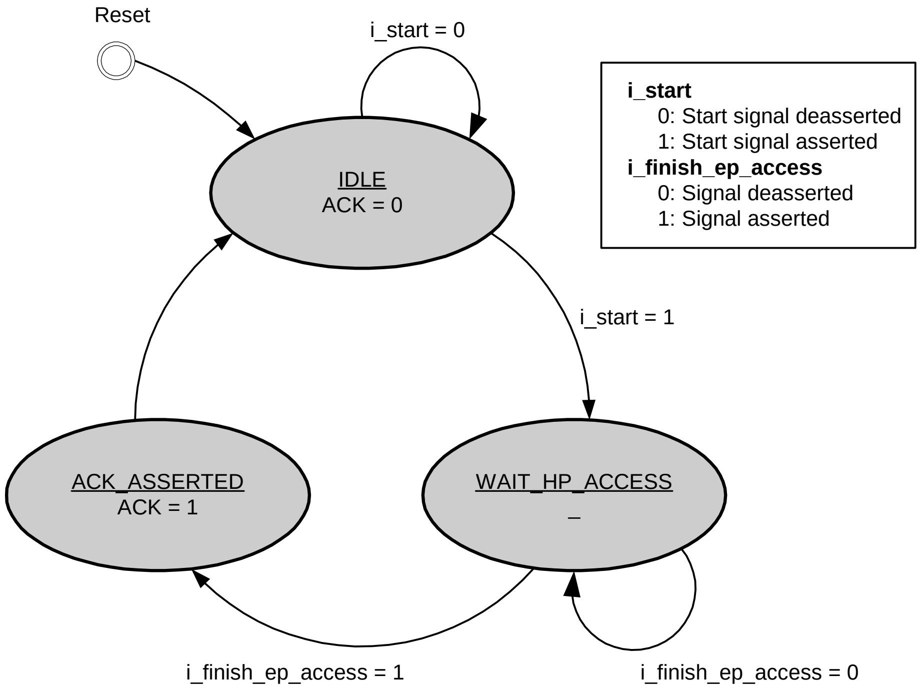


Figure 4. External Port state machine

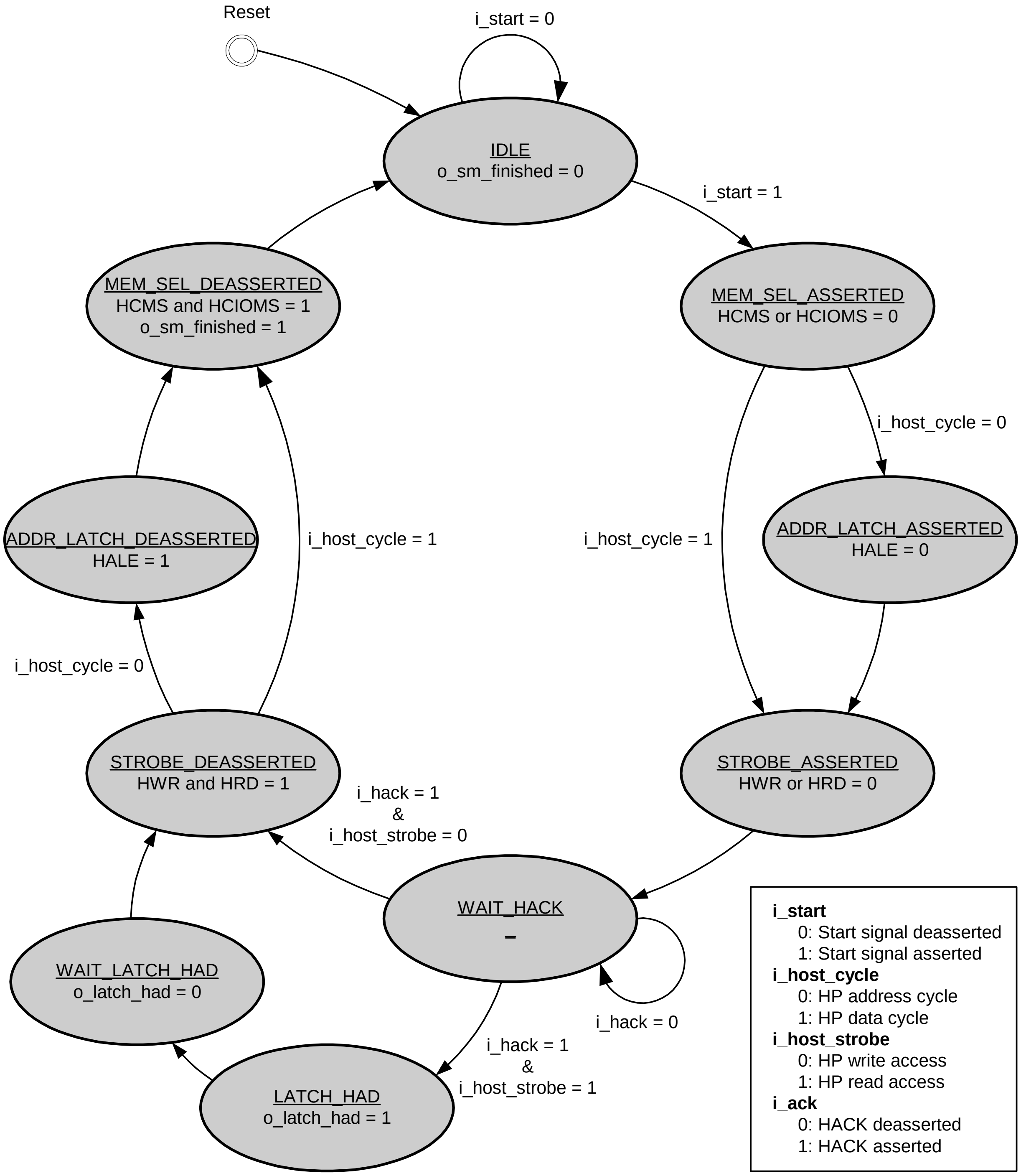


Figure 5. Host port state machine

4.5. Data Path

The Data Path block accomplishes the bi-directional data transfer between Host and Target via the multiplexed address and data bus.

The Host Port uses byte addressing, which means that addresses for accesses to DM and I/O space are shifted left by 1 bit, whereas addresses for PM need to be shifted left by 2 bits. Address generation is done in the Data Path block. The details about the type of access (address cycle or data cycle; access to memory or I/O space; access to DM or PM) are decoded from information contained in the External Port address lines.

This block also detects whether the current operation is a valid access. Due to the address mapping described earlier, invalid combinations of access type and data direction can occur. For example, the virtual address space allows an address read cycle, which is not defined for the Host Port. In such a case, the PLD acknowledges the access to the Host without executing the access to the Host Port. Accesses to reserved addresses, or incorrect combinations of access type and data direction, do not cause ERROR to be asserted. The Host is responsible for performing valid operations only.

Figure 6 shows a simplified block diagram of the Data Path. The shifters required to shift addresses on the multiplexed address and data bus are not shown.

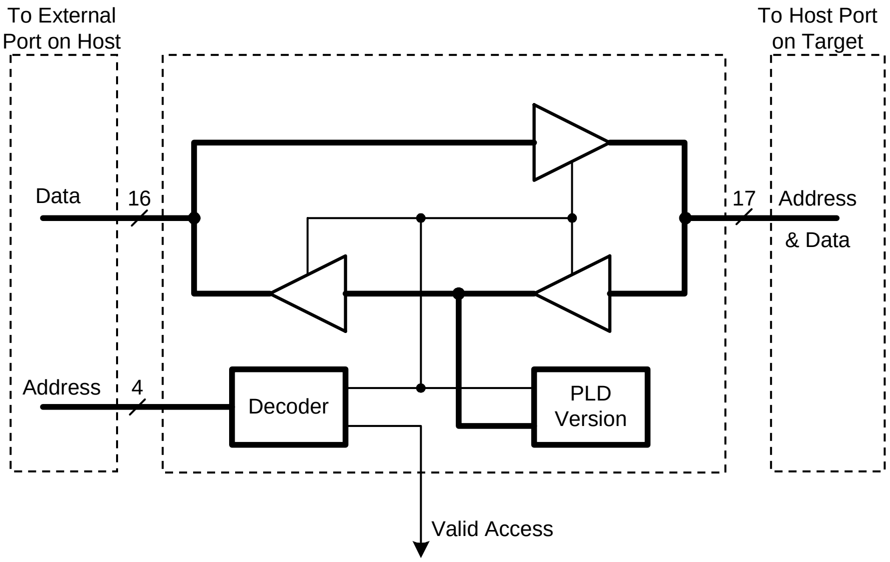


Figure 6. Data path block diagram

4.6. Watchdog

The Watchdog has two functions:

Should the Host not wait for an acknowledge from the PLD, and start an access while the previous access is still occurring, the Watchdog detects this and signals an error to the Host. This error is caused by an incorrect External Port setup, which should be configured for "external acknowledge only".

The second, more important function of the Watchdog, is to detect when the Target DSP does not respond to a Host Port access with the assertion of HACK. After a certain length of time, the ERROR signal is asserted to the Host.

In both cases, the Watchdog resets the Host Port state machine and forces the External Port state machine to finish the access by asserting the External Port acknowledge signal.

5. Booting via Host Port

5.1. Boot Mode Configuration

The ADSP-2191 DSP provides different means of loading a program into memory, selected by the state of three pins during hardware reset: OPMODE, BMODE1, and BMODE0.

Driving OPMODE and BMODE0 low, and BMODE1 high during reset, causes the DSP to start in Host boot mode. In this mode, the processor simply waits in an endless-loop until a '1' is written into the Host Port Semaphore A register. Following this write, code execution starts in internal memory at address 0x00 0000. It is the Host's responsibility to transfer all program data into the Target memory before writing to the semaphore register.

The last 16 locations in Program Memory in Page 0 and the last 272 locations in Data Memory in Page 0 are reserved and should not be used for the placement of static data or code.

However, they can be used when the boot process is finished.

5.2. Boot Loader File

For booting, VisualDSP++® provides a special output file – the boot loader file. The default file extension is 'ldr'. Details about the creation of the loader file and available options are explained in [3].

Loader files can be created in 8- or 16-bit format. The hardware described earlier uses 16-bit-wide data buses, so the obvious choice is to create a loader file that contains 16-bit data words. The structure of a 16-bit Host boot loader file in ASCII format is shown in Figure 7.

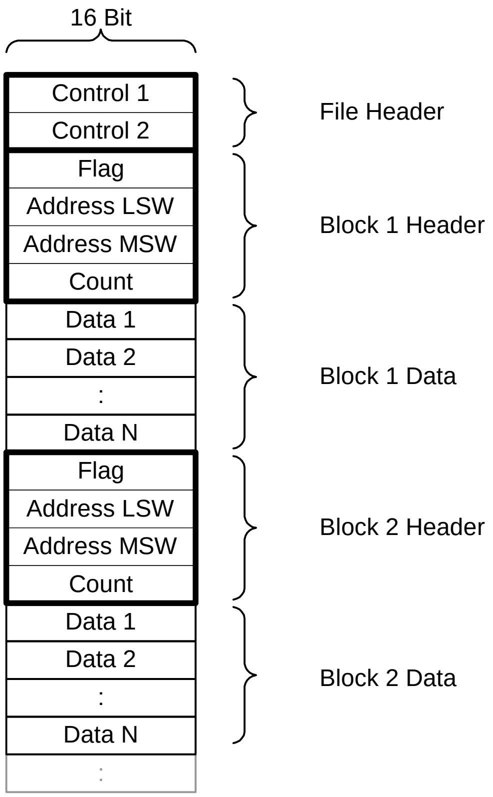


Figure 7. Format of 16-bit Host boot loader file

Control

The file begins with a file header, consisting of two 8-bit control words (the upper 8 bits in both words are not used and are padded with zeros). The first word, *Control 1*, contains information about the configuration of the External Port and whether the Host should enable the SPI after the boot process. A bit description for *Control 1* is given in Figure 8. Bit field *Wait States* holds the number of wait states to be inserted for every access to the External Port. The *Base Clock Divider* determines the clock frequency for the External Port. These two fields have meaning only when code or data are to be placed in external memory.

On the ADSP-2191 DSP, the SPIs are multiplexed with SPORT2. Bit 6, *OPMODE*, determines whether the SPIs or SPORT2 are enabled. This bit corresponds to Bit 0 in the System Configuration Register (*SYSCR*).

Only two bits are used in *Control 2*, the second configuration word (see Figure 9). Bit 0, *Host Port Bus Width*, determines whether the loader file was created for 8- or 16-bit bus width (the word width of the values in the loader file). Bit 1, *CRC Checksum*, indicates that a checksum is used for the header file.

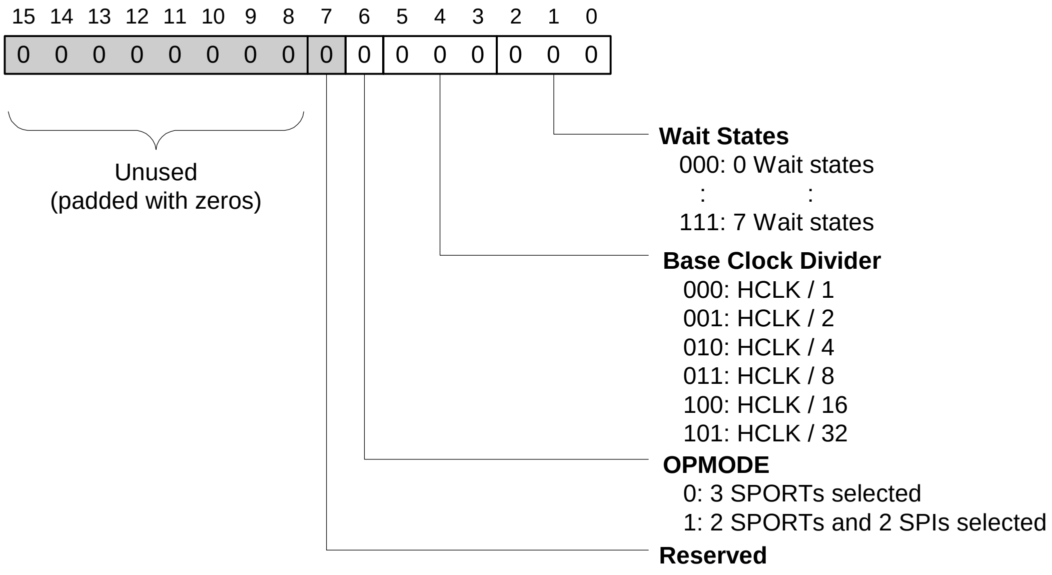


Figure 8. Bit description of Control 1

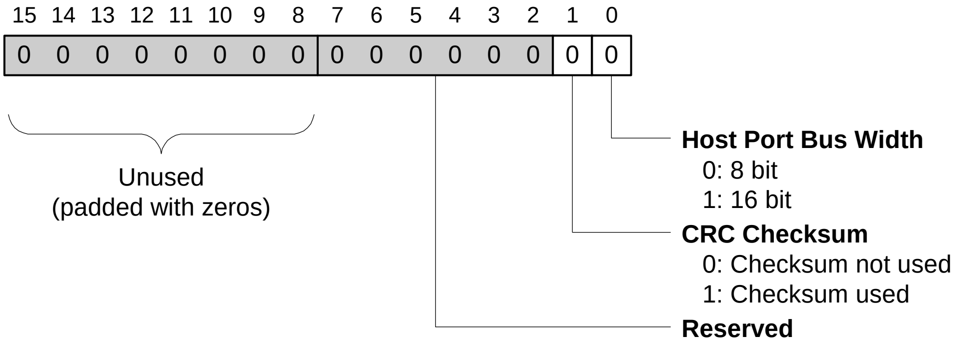


Figure 9. Bit description of Control 2

Flag

One or more block headers follow the file header.

The first entry in every block header, called *Flag*, is a 16-bit configuration word that contains information about the payload following the header. Figure 10 provides bit descriptions for this configuration word.

Bit 0, *Target Memory Type*, specifies the width of the values following the header or the values the Host has to send to the Target to initialize a part of memory to zero. A ‘0’ indicates access to 24-bit memory, and a ‘1’ indicates access to 16-bit memory.

For the last header in the loader file, Bit 1, *Final Block*, is set to ‘1’, indicating to the boot loader that the end of the file has been reached.

If Bit 2, *Zero-Init Memory*, is cleared, the header is followed by payload data. This block of data is to be transferred to the Target DSP. If Bit 2 is set, no data follow the header. Instead, the Host initializes the specified memory block to zero. This mechanism allows it to reduce the size of the loader file, since there are often chunks of memory containing zeros (for example external and static variables in C programs are initialised to zero by default if not otherwise initialised).

Bit 4 determines the *External Memory Bus Width*. It is used only when data are to be placed in external memory. If this bit is set to ‘1’, the bus width for the external port is 16 bits, otherwise the bus width is 8 bits. The Host can use this information to configure the external port on the ADSP-2191 DSP. When a block contains data for external memory, the *Address MSW* field contains a non-zero value.

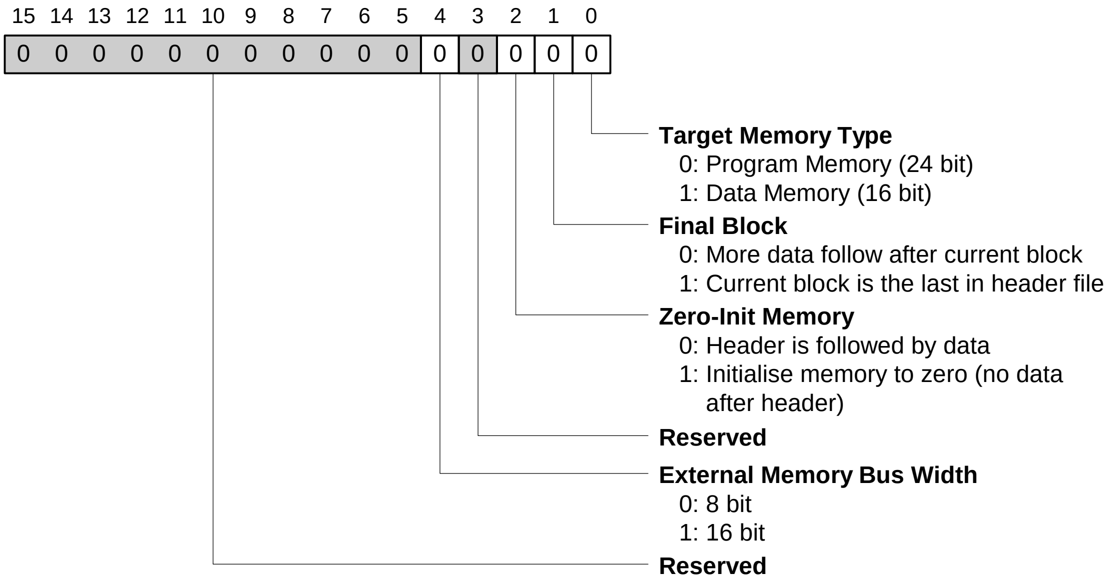

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Target Memory Type
0: Program Memory (24 bit)
1: Data Memory (16 bit)
Final Block
0: More data follow after current block
1: Current block is the last in header file
Zero-Init Memory
0: Header is followed by data
1: Initialise memory to zero (no data after header)
Reserved
External Memory Bus Width
0: 8 bit
1: 16 bit
Reserved

Figure 10. Bit description of Flag

Address

The loader file described in this chapter contains 16-bit values. The address range of the ADSP-2191 DSP, however, is 24 bits. Two 16-bit values are necessary to provide a complete address for the boot loader. The entries *Address MSW* and *Address LSW* in the block header provide that address. *Address LSW* contains the 16 least significant bits, and the lower 8 bits of *Address MSW* contain the 8 most significant bits of the address in Target memory. The upper 8 bits of *Address MSW* are unused and always contain zeros. Figure 11 shows the representation of addresses in the 16-bit loader file.

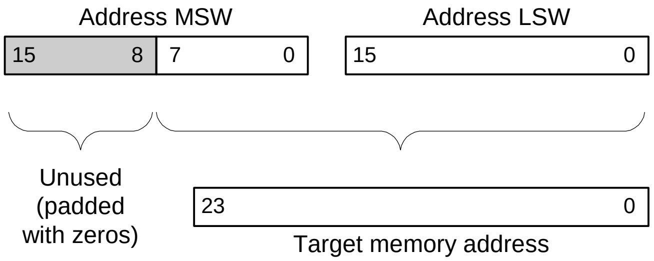


Figure 11. Representation of Target memory address in loader file

Count

The last entry in every block header is *Count*, which determines the number of data words for the Target memory that follow the header (*N*), or the number of words in Target memory that have to be initialised to zero. If the *Target Memory Type* in *Flag* was set to 16 bits, the number of words following the header is *N = Count*. If *Target Memory Type* was set to 24 bits, two data words in the loader file are required to hold one 24-bit value. The number of 16-bit words following the header is then *N = 2*Count*. In this case, the 16 most significant bits of a 24-bit value are stored in an even data word (*Data 2n*), whereas the lower 8 bits of the 24-bit value are stored in the upper 8 bits of an odd data word (*Data 2n-1*). Figure 12 shows the packing of a 24-bit word into two 16-bit words in the loader file.

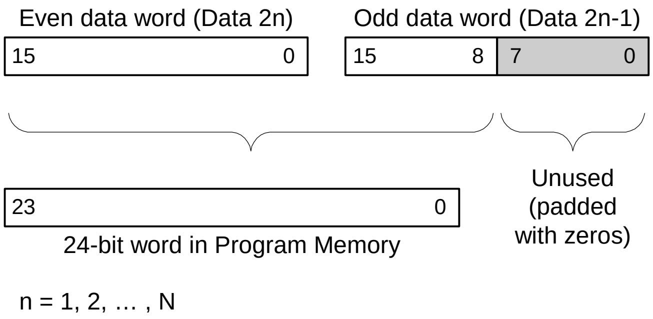


Figure 12. Representation of 24-bit words in loader file

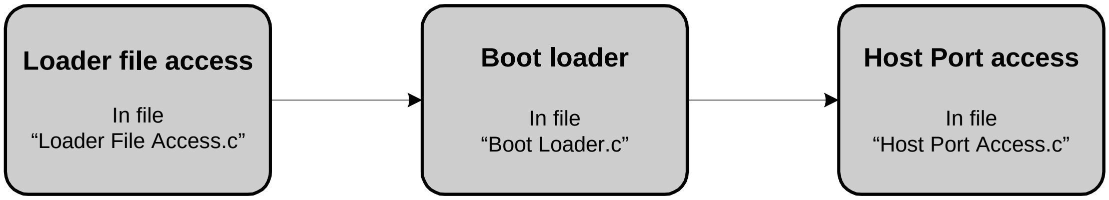


Figure 13. Software boot loader overview

6. Software Boot Loader

General Description

A software boot loader, running on the Host DSP, transfers the loader file to the Target DSP. It reads the complete content of the loader file, analyses the information in the headers, and transfers the data contained in the file accordingly via the Host Port to the Target memory. Figure 13 shows a simplified block diagram of the program.

Loader File Access

Access to the loader file is provided by the following functions, which are stored in the file "Loader File Access.c":

- Open_Loader_File
- Close_Loader_File
- Read_Next_Loader_Value

A complete description of these functions can be found in the source code.

For the system described earlier, where the loader file is stored in Flash memory connected to the Host, no "real" file handling is necessary.

Host Port Access

Data transfer to and from the Target Host Port is handled by the following set of functions, located in the file "Host Port Access.c":

- Write_Target_IO_Direct
- Write_Target_DM_Direct
- Write_Target_PM_Direct
- Read_Target_IO_Direct
- Read_Target_DM_Direct
- Read_Target_PM_Direct
- Write_Target_DM_DMA
- Write_Target_PM_DMA
- Init_Target

A complete description of these functions can be found in the source code.

For the current implementation, these functions communicate with a PLD, which, in turn, accesses the Host Port.

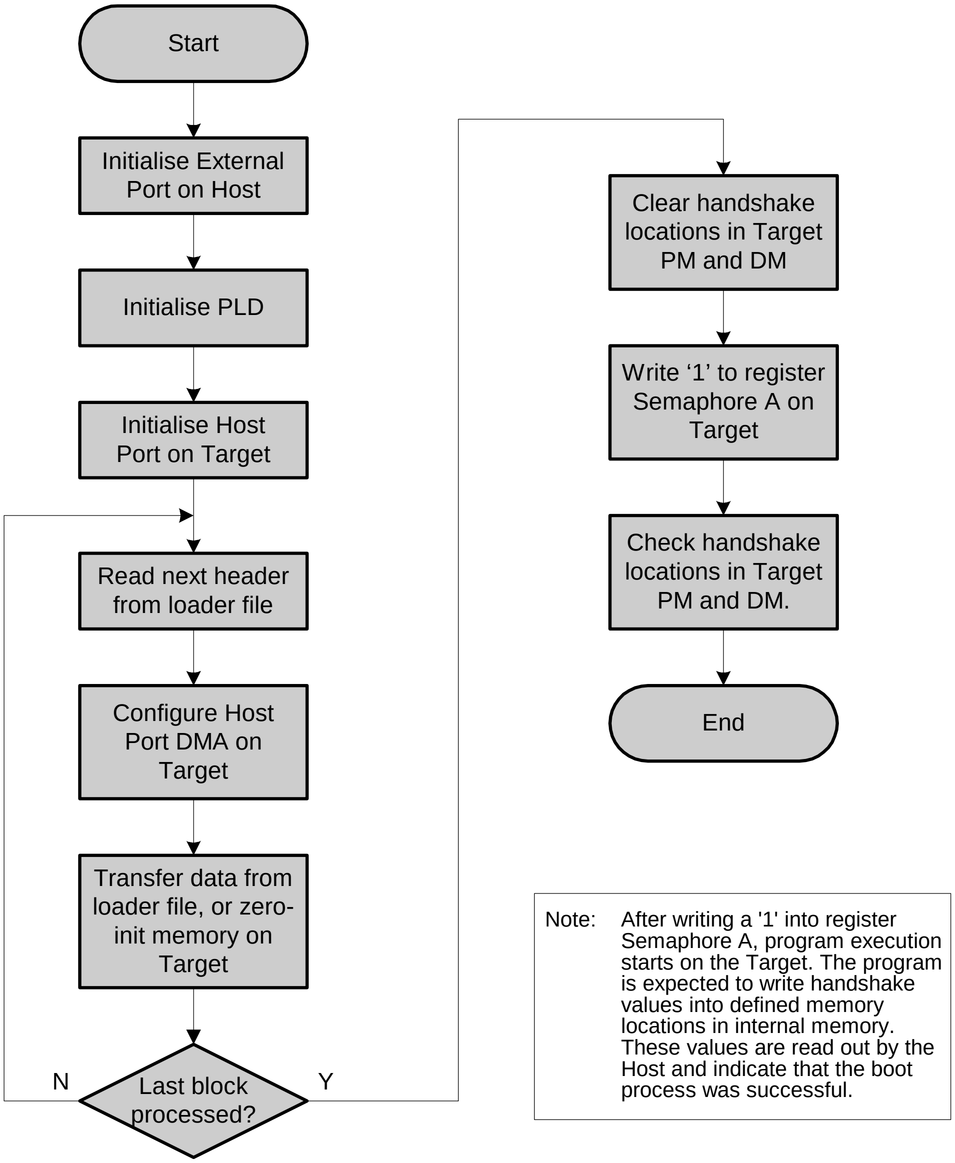
Boot Loader

The actual core of the boot loader, which is located in the file “Boot Loader.c”, is written in generic C, and can be executed on any platform that supports this language. For this case, the functions that access the loader file and Host Port must be adapted to the underlying hardware. To make porting the code to another platform easier, the interface between boot loader and loader file (and between boot loader and Host Port) are kept in a very general form, so only the implementation needs to be changed, not the interface.

A flow chart of the software boot loader is shown in Figure 14. After initializing the hardware, the program sequentially reads the values from the loader file. It uses the information contained in the block headers to initialize the Host Port DMA on the Target, and then transmits the content of the data blocks into the Target’s memory.

When the last block has been processed, writing a ‘1’ into the Host Port Semaphore A register starts program execution on the Target. The transferred program is expected to write certain values into pre-defined memory locations in Program Memory and Data Memory. When the Host finds the correct values in those handshake locations, it is an indication that the boot process was successful.

To ensure that the *complete* program has been transferred correctly, the Host would have to verify that every single word is correct in Target memory. But the simple handshake just described is considered to be sufficient for this example project.



Start
Initialise External Port on Host
Initialise PLD
Initialise Host Port on Target
Read next header from loader file
Configure Host Port DMA on Target
Transfer data from loader file, or zero-init memory on Target
Last block processed?
N
Y
Clear handshake locations in Target PM and DM
Write ‘1’ to register Semaphore A on Target
Check handshake locations in Target PM and DM.
End

Note: After writing a '1' into register Semaphore A, program execution starts on the Target. The program is expected to write handshake values into defined memory locations in internal memory. These values are read out by the Host and indicate that the boot process was successful.

Figure 14. Boot loader flow chart

7. Screenshots

This final chapter contains screenshots from a logic analyzer showing the Host Port signals for accesses to I/O space and memory in Direct mode and DMA mode. The screenshots were taken during different stages of the boot process.

Since the default Host Port bus width after reset is 8 bits, the first action for the Host is to configure the HPI for a 16-bit bus width. Bit 0 in the Host Port Configuration Register must be set to ‘1’ for that. Figure 15 shows the first access to the Target after reset, consisting of two 8-bit writes to *HPCR* (I/O address 0x3802). The reset value of this register is 0x0F00, so the value 0x0F01 is written into it (first the LSB, 0x01; then the MSB, 0x0F). After that, the Host Port is configured for 16-bit bus width.

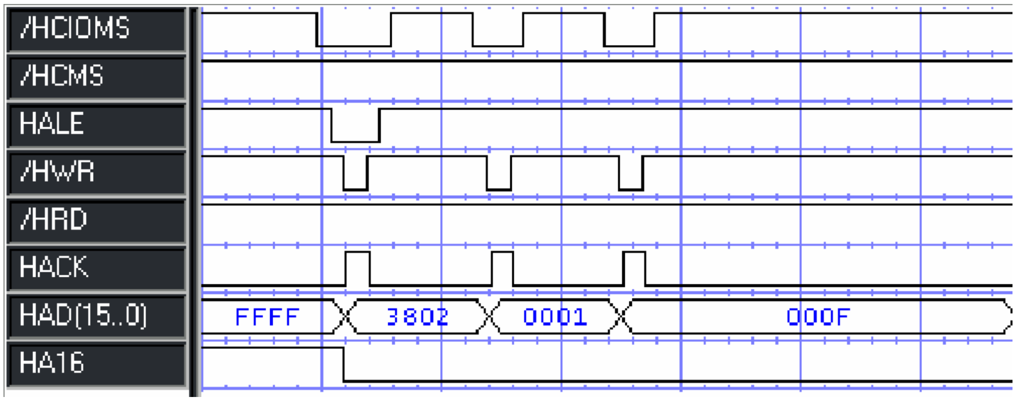


Figure 15. Direct write access to register HPCR in 8-bit mode

The software boot loader described earlier uses Host Port Autobuffer DMA to transfer the data segments stored in the loader file to the DSP. A typical configuration for such a DMA transfer is shown in Table 5, which contains the register name to be configured, the address and value for that register, and a short explanation of the purpose.

The screenshot in Figure 16 shows the corresponding transfers via the Host Port.

The DMA data transfer following the DMA configuration is shown in Figure 17. The five values being transmitted are 0x0061, 0x0064, 0x0069, 0x0000 and 0x8000; they will be stored in five consecutive locations in DM, starting at address 0x8000.

As can be seen, no addresses need to be sent during DMA transfers. However, from the amount of data necessary for a complete DMA setup, it is obvious that using DMA is efficient only for transferring large amounts of data, or when the DMA configuration does not change.

	Transfer to register	Address	Value	Comment
1	HOSTD_CFG	0x3A02	0x0010	Enable Autobuffer mode
2	HOSTD_SRP	0x3A04	0x0000	Set start page 0x00
3	HOSTD_SRA	0x3A06	0x8000	Set start address 0x8000
4	HOSTD_CNT	0x3A08	0x0005	Set word count 5
5	HOSTD_CFG	0x3A02	0x0017	Start DMA write

Table 5. Example of Host Port DMA configuration

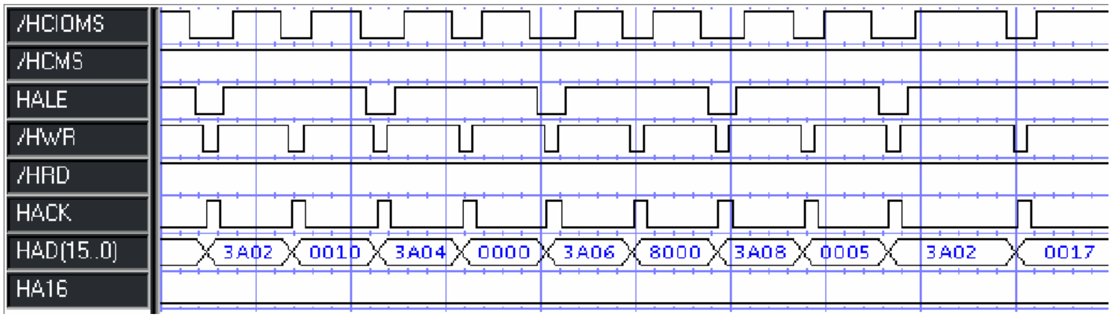


Figure 16. Configuration of Host Port Autobuffer DMA via Host Port

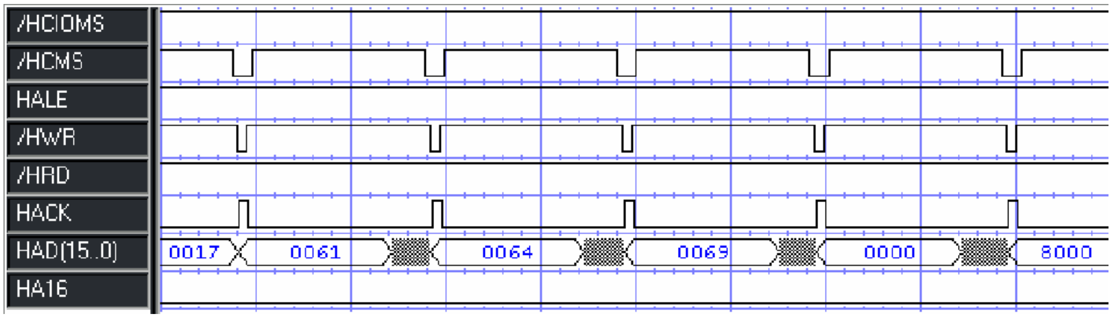


Figure 17. DMA write access of five 16-bit words

When the complete loader file has been transferred to the DSP, writing a 1 into the Semaphore A register starts program execution. The screenshot in Figure 18 shows the value 0x0001 being written to address 0x39F8 (register *HPSMPHA*), which marks the end of the boot process.

Figure 19 shows a read from Data Memory location 0xFFFF. The address is shifted left by 1 bit, and HA16 is set to 1. The 16-bit value read from this location is 0x1234.

The transfer shown in Figure 20 is a 24-bit read from Program Memory location 0x7FFF. The address is shifted left by 2 bits, and HA16 is set to 1. The value read from this location is 0x123456.

The boot loader uses these last two reads as a simple method for verifying that program execution has started on the Target (the values are written by the transferred program). When the expected values can be found in the handshake locations, it can be assumed that the boot process was successful and the program is running properly.

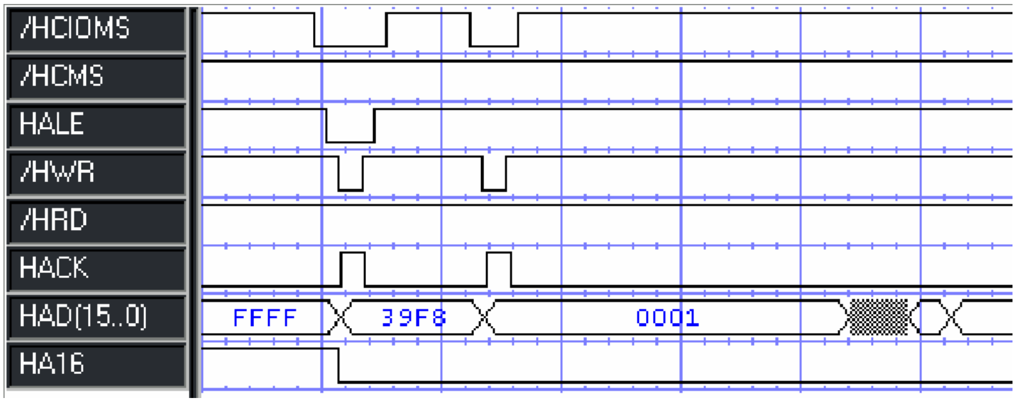


Figure 18. Direct write access to register HPSMPHA (16-bit mode)

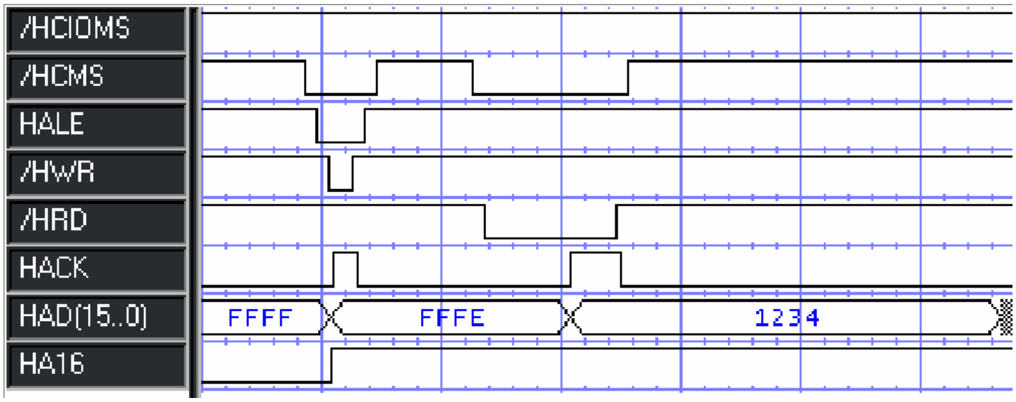


Figure 19. Direct read access from DM location 0xFFFF (16-bit data)

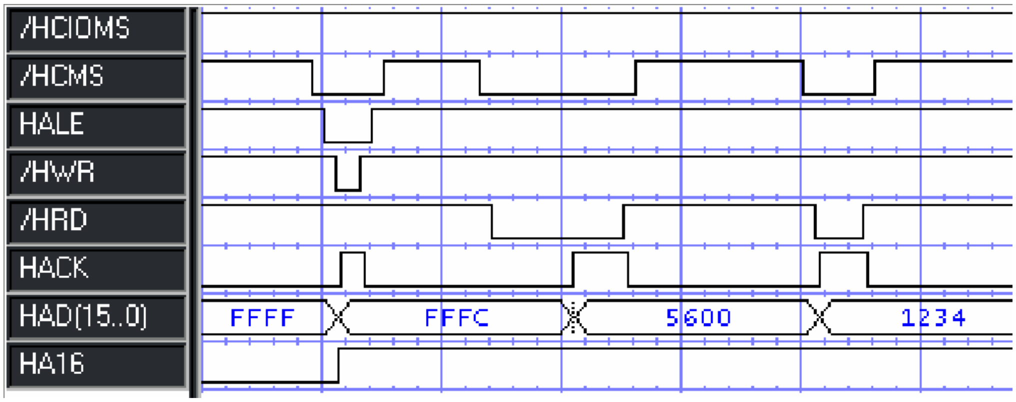
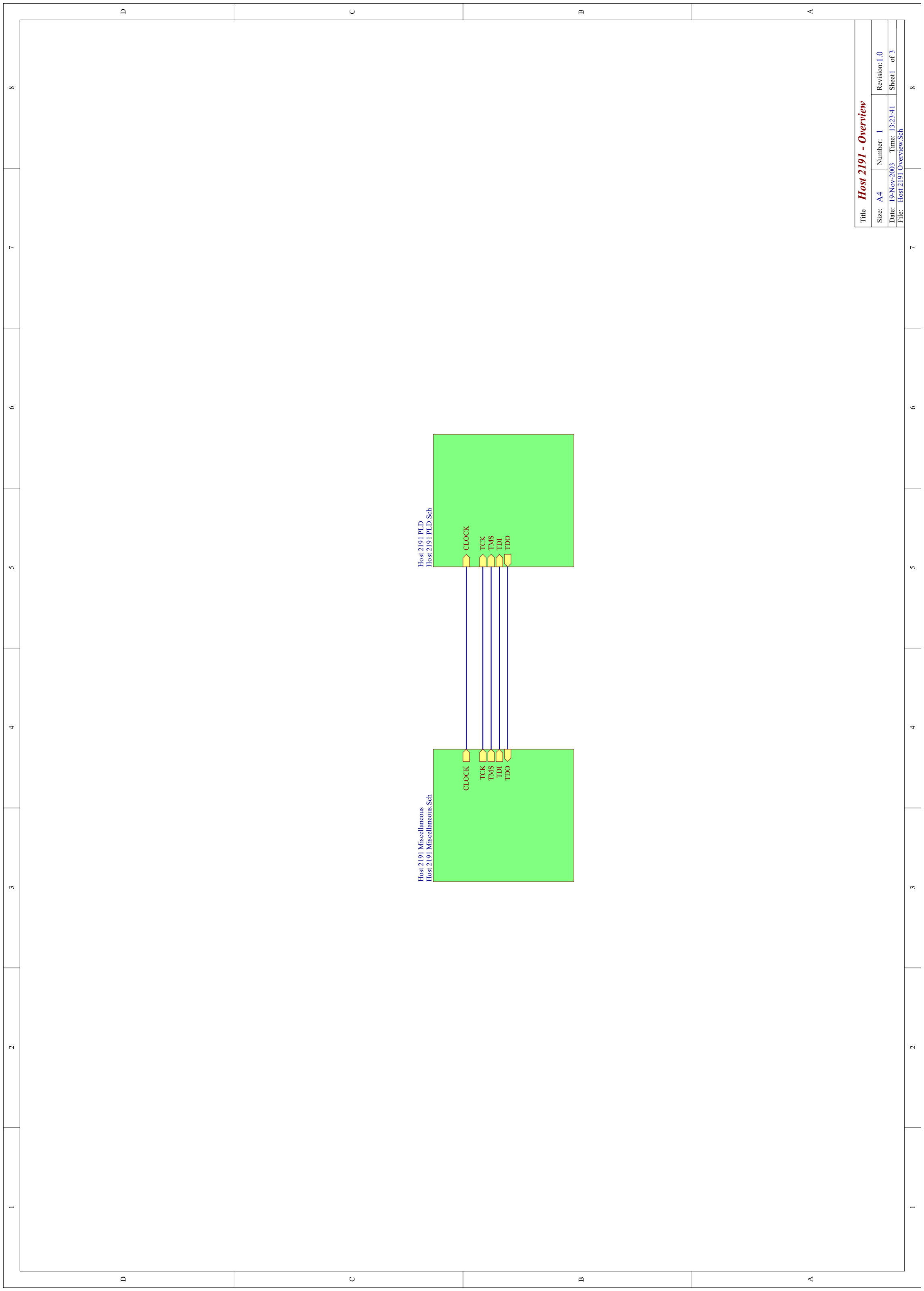
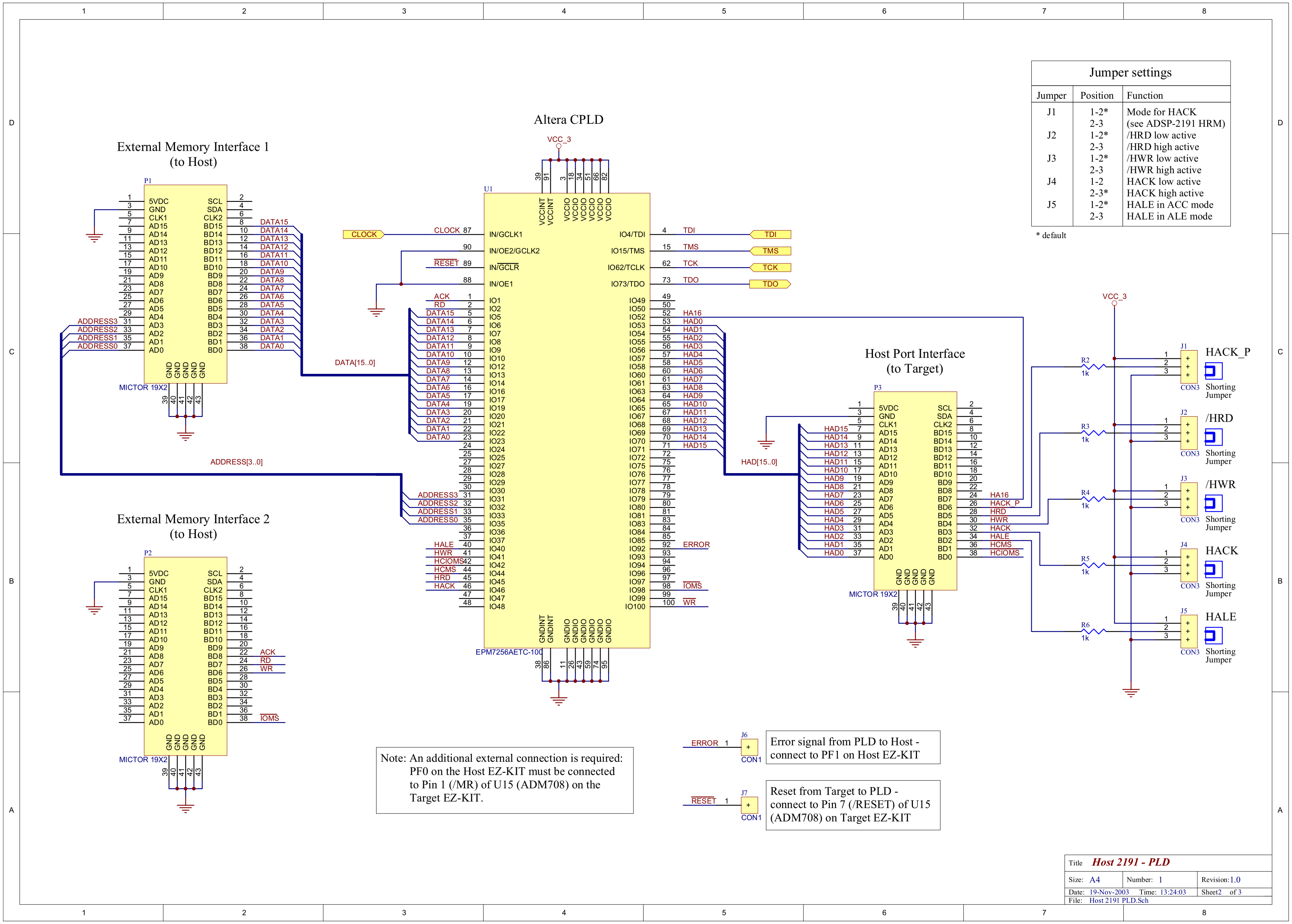


Figure 20. Direct read access from PM location 0x7FFF (24-bit data)

A. Schematic Diagram





Jumper settings

Jumper	Position	Function
J1	1-2*	Mode for HACK
	2-3	(see ADSP-2191 HRM)
J2	1-2*	/HRD low active
	2-3	/HRD high active
J3	1-2*	/HWR low active
	2-3	/HWR high active
J4	1-2	HACK low active
	2-3*	HACK high active
J5	1-2*	HALE in ACC mode
	2-3	HALE in ALE mode

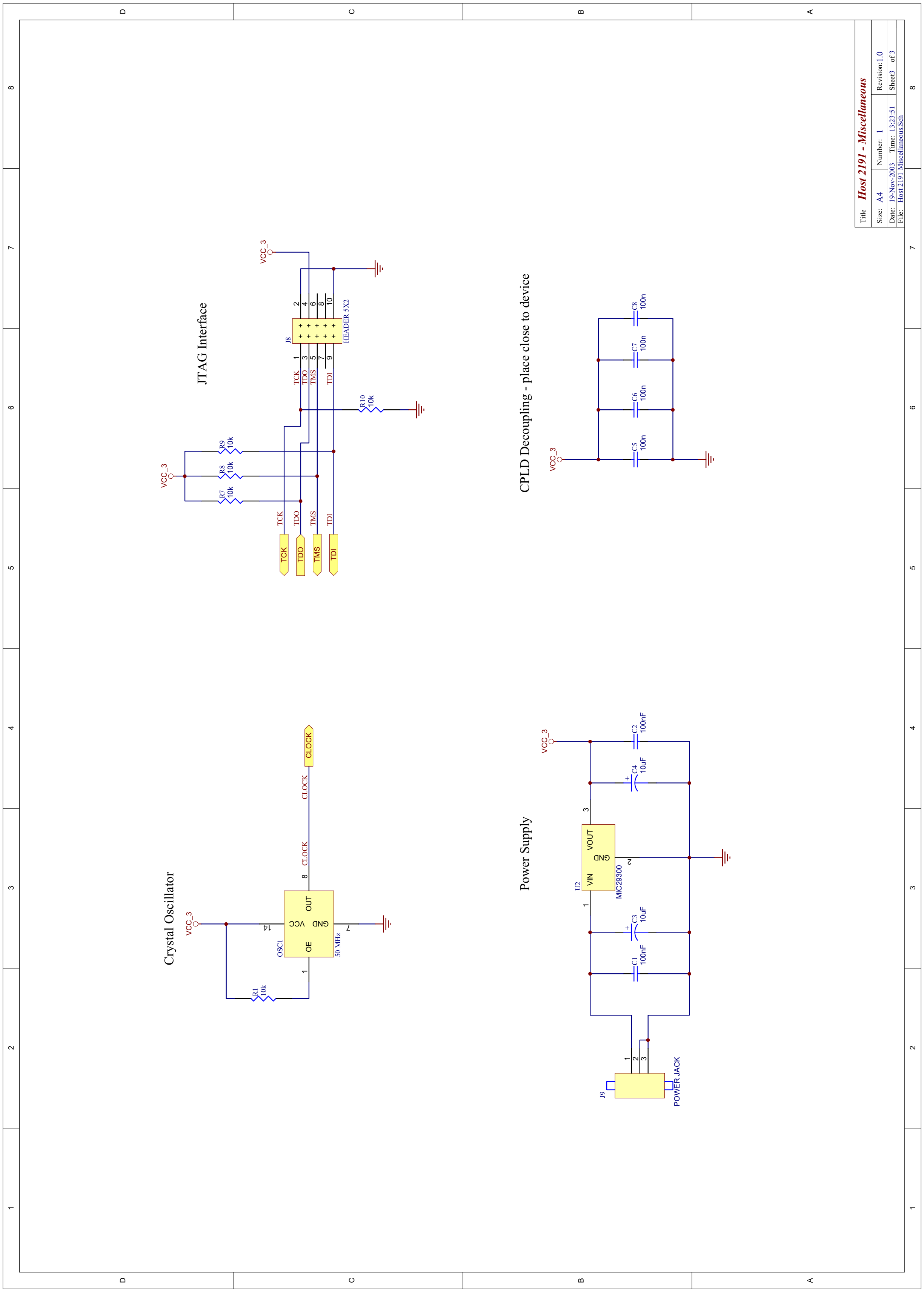
* default

Note: An additional external connection is required: PF0 on the Host EZ-KIT must be connected to Pin 1 (/MR) of U15 (ADM708) on the Target EZ-KIT.

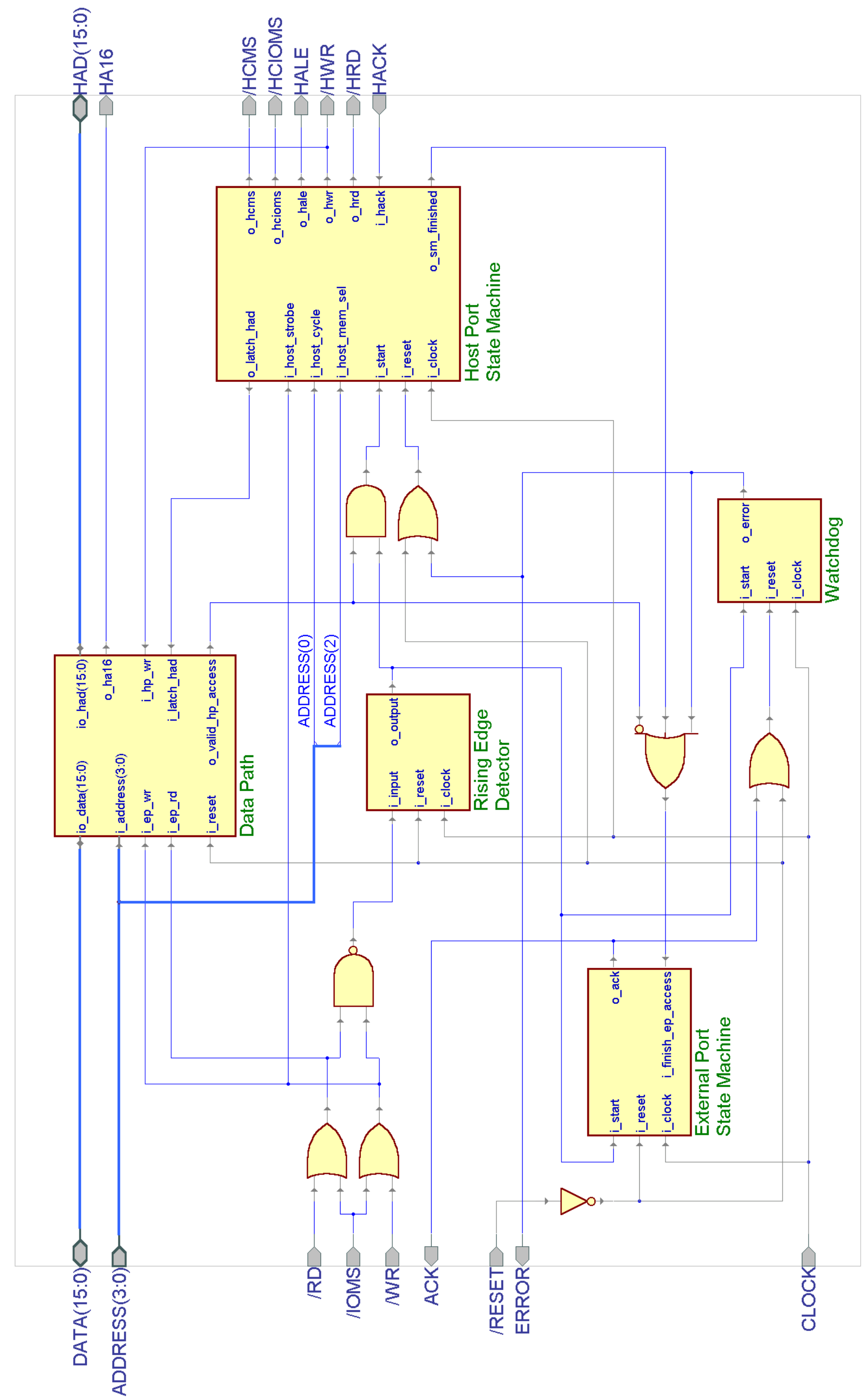
Error signal from PLD to Host - connect to PF1 on Host EZ-KIT

Reset from Target to PLD - connect to Pin 7 (/RESET) of U15 (ADM708) on Target EZ-KIT

Title	*Host 2191 - PLD*		
Size: A4	Number: 1		Revision: 1.0
Date: 19-Nov-2003	Time: 13:24:03		Sheet2 of 3
File: Host 2191 PLD.Sch			


JTAG Interface
VCC_3
J8
HEADER 5X2
1 3 5 7 9
2 4 6 8 10
TCK
TDO
TMS
TDI
R7 10k
R8 10k
R9 10k
R10 10k
CPLD Decoupling - place close to device
C5 100n
C6 100n
C7 100n
C8 100n
Crystal Oscillator
OSC1
50 MHz
OE
VCC
GND
OUT
R1 10k
CLOCK
Power Supply
U2
MIC29300
VIN
GND
VOUT
C1 100nF
C3 10uF
C4 10uF
C2 100nF
J9
POWER JACK
Title Host 2191 - Miscellaneous
Size: A4
Number: 1
Revision:1.0
Date: 19-Nov-2003
Time: 13:23:51
Sheet3 of 3
File: Host 2191 Miscellaneous.Sch
A B C D
1 2 3 4 5 6 7 8

B. VHDL Top-Level Design



Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	ACK	26	GNDIO	51	VCCIO	76	-
2	~RD	27	-	52	HA16	77	-
3	VCCIO	28	-	53	HAD0	78	-
4	TDI	29	-	54	HAD1	79	-
5	DATA15	30	-	55	HAD2	80	-
6	DATA14	31	ADDRESS3	56	HAD3	81	-
7	DATA13	32	ADDRESS2	57	HAD4	82	VCCIO
8	DATA12	33	ADDRESS1	58	HAD5	83	-
9	DATA11	34	VCCIO	59	GNDIO	84	-
10	DATA10	35	ADDRESS0	60	HAD6	85	-
11	GNDIO	36	-	61	HAD7	86	GNDINT
12	DATA9	37	-	62	TCK	87	CLOCK
13	DATA8	38	GNDINT	63	HAD8	88	GND
14	DATA7	39	VCCINT	64	HAD9	89	~RESET
15	TMS	40	HALE	65	HAD10	90	GND
16	DATA6	41	~HWR	66	VCCIO	91	VCCINT
17	DATA5	42	~HCIOMS	67	HAD11	92	ERROR
18	VCCIO	43	GNDIO	68	HAD12	93	-
19	DATA4	44	~HCMS	69	HAD13	94	-
20	DATA3	45	~HRD	70	HAD14	95	GNDIO
21	DATA2	46	HACK	71	HAD15	96	-
22	DATA1	47	-	72	-	97	-
23	DATA0	48	-	73	TDO	98	~IOMS
24	-	49	-	74	GNDIO	99	-
25	-	50	-	75	-	100	~WR

Table 6. PLD Pin Assignment

References

[1] *ADSP-219x/2191 DSP Hardware Reference*. 2001. Analog Devices Inc.

[2] *ADSP-2191M Data Sheet*. 2002. Analog Devices Inc.

[3] *VisualDSP++ Linker and Utilities Manual for ADSP-218x and ADSP-219x DSPs*. 2002. Analog Devices Inc.

[4] *ADSP-2191 EZ-KIT Lite Evaluation System Manual*. 2001. Analog Devices Inc.

[5] *Booting the ADSP-2191/95/96 DSPs (EE-131)*. April 2003. Analog Devices Inc.

[6] *ADSP-2191 Host Port Interface (EE-154)*. October 2002. Analog Devices Inc.

Document History

Version	Description
Rev 1 January 23, 2004 *by M. Kuegler*	Initial Release based on VisualDSP++ version 3.0