Host Port Interface

Introduction

This module discusses the Host Port Interface (HPI). First, a brief overview of the HPI will discuss the reasons for including it on these devices and some of the benefits that it provides. Next, we present examples to help you understand the terminology, capabilities, and basic flow of the HPI. The module also includes a discussion of the HPI’s other features. The module ends with a basic comparison of the HPI to the ‘C6202/03/04 Expansion Bus. By the end of this module you will have a good understanding of the HPI and the Expansion Bus and how they provide a capable interface to industry standard hosts processors.

Learning Objectives

<table>
<thead>
<tr>
<th>Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>◆ HPI Overview</td>
</tr>
<tr>
<td>◆ HPI on the DSK</td>
</tr>
<tr>
<td>◆ Host Software Example</td>
</tr>
<tr>
<td>◆ HPI Hardware Description</td>
</tr>
<tr>
<td>◆ Optional Discussions</td>
</tr>
</tbody>
</table>
Chapter Topics

Host Port Interface.........................................................................................................................................16-1

HPI Overview ..................................................................................................................................................16-3

HPI and the DSK ..............................................................................................................................................16-5

HPI – Host Software Example .......................................................................................................................16-6

HPI Hardware Description...............................................................................................................................16-7

Setting Up the Control Register (HPIC)........................................................................................................16-7
Setting Up the Address Register....................................................................................................................16-10
Writing a 32-bit Value ....................................................................................................................................16-12
Reading a 32-bit Value .....................................................................................................................................16-15
Reading Multiple Values ...............................................................................................................................16-18
HPI Pins..........................................................................................................................................................16-20

HSTRB .........................................................................................................................................................16-21

HAS...............................................................................................................................................................16-21

An Example Interface.....................................................................................................................................16-22

HPI Related Registers (Optional Topic)........................................................................................................16-23

HPIC...............................................................................................................................................................16-23

CSL API for the Host Port Interface................................................................................................................16-24

Expansion Bus (Optional Topic).....................................................................................................................16-25

XB Summary..................................................................................................................................................16-30
HPI Overview

The HPI provides an economical 16-bit parallel port for interfacing a ‘C6x to host processors, other ‘C6xs, and PCI bridge chips. This bus is in addition to the ‘C6x external bus (EMIF) and multi-channel serial ports, which may be dedicated to memory and A/Ds or codecs.

A dedicated bus is used to transfer data to or from an address in the ‘C6x memory map. The HPI has a 32-bit registers for each control, address, and data. The HPIC is used to control HPI transfers. The HPIA is the address for the read or write operation. The HPID is the data register.

The HPI is connected to the ‘C6x memory via the DMA Auxiliary Channel, which gives the host access to the entire ‘C6x memory map. The Auxiliary Channel is the fifth channel of the DMA, and it is dedicated to the HPI.
Since the HPI bus is only 16-bits wide, each data transfer to an HPI register requires two read or write operations. Although this is slower, it lowers the pin count of the device.

The HPI provides a simple slave interface to a host, which serves as the master. It gives the host processor access to entire memory map of the ‘C6x, including the internal memories, the EMIF, and the peripheral control registers.

**Why HPI for Communication?**

- Give host control of the transfer
- Allow host to access the entire C6000 memory map
- Additional parallel bus for data exchange between a host and the C6000
- Provide glueless interface to many different types of hosts
HPI and the DSK

Host → DSK Communications

- The C6713 DSK has a HPI connector which brings out the pins of the Host Port Interface
- On the C6416 DSK, this connector contains the muxed HPI/PCI pins
- Also shown, the JTAG emulation connections
HPI – Host Software Example

Some Ideas for Host Interface API

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C6X_open()</td>
<td>Open a connection to the C6000</td>
</tr>
<tr>
<td>C6X_close()</td>
<td>Close a connection to the C6000</td>
</tr>
<tr>
<td>C6X_resetBoard()</td>
<td>Reset the entire board</td>
</tr>
<tr>
<td>C6X_resetDsp()</td>
<td>Reset only the DSP on the board</td>
</tr>
<tr>
<td>C6X_dspImageLoad()</td>
<td>Load a DSP image (COFF) to DSP memory</td>
</tr>
<tr>
<td>C6X_memRead()</td>
<td>Read DSP memory via the HPI</td>
</tr>
<tr>
<td>C6X_memWrite()</td>
<td>Write to DSP memory via the HPI</td>
</tr>
<tr>
<td>C6X_ctrlRead()</td>
<td>Read HPI control register</td>
</tr>
<tr>
<td>C6X_ctrlWrite()</td>
<td>Write to HPI control register</td>
</tr>
<tr>
<td>C6X_generateInt()</td>
<td>Generate a DSP interrupt</td>
</tr>
<tr>
<td>C6X_isr()</td>
<td>Respond to host interrupt (HINT) from DSP</td>
</tr>
</tbody>
</table>

- Here are some ideas for the host software (and hardware) functionality you might want to build into your system.
- These routines could be combined to create more advanced host functions (like routines for setting up the EDMA and such).
- Unfortunately, we cannot provide these functions for you, as they must be written specific to the hardware of your host.
HPI Hardware Description

Setting Up the Control Register (HPIC)

The first step in using the HPI is to setup the HPIC. This register contains the halfword ordering bit, or HWOB. HWOB sets the endianness for HPI transfers. If HWOB=0, then the first halfword transferred will be put in the MSBs. If HWOB=1, then the first halfword transferred will be put in the LSBs.

<table>
<thead>
<tr>
<th>Setup HPI Control Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
</tr>
<tr>
<td>reserved</td>
</tr>
<tr>
<td>15</td>
</tr>
</tbody>
</table>

- Setup the HPI Control register (HWOB-bit) to specify which 16-bits (upper or lower) are transferred first.
- Similar to little/big endian.
- Order doesn’t matter when writing to HPIC as the fields are aliased to both halves.

Writing to this register is selected by the HCNTL(1:0) pins. These pins select the register that the host wants to read or write. They are usually connected to address pins on the host side.

<table>
<thead>
<tr>
<th>HCNTL Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCNTL1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

1. Use HCNTL[1:0] = 00, to enable access to HPIC
The HR/W pin determines the direction of the transfer.

HHWIL identifies which halfword is being transferred. For the first halfword of a transfer, HHWIL will be low. For the second halfword, HHWIL will be high. Remember that the HWOB bit in the HPIC determines if the first halfword is put in the LSBs (little endian) or the MSBs (big endian). What happens to HPIC when it is written for the first time? Is the value written to the LSBs or the MSBs? It turns out that HPIC is really only 16 bits, and the LSBs and MSBs are the same.
The HSTRB signal initiates the transfer. At the falling edge of HSTRB, the other control signals are sampled and the write operation becomes active. The value on the HD pins is latched into the HPIC register at the rising edge of HSTRB. The first half of the 32-bit transfer is complete.

For the second half of the transfer, some of the control pins (HCNTL, HR/W) do not need to change. In the case of HPIC, HD does not change. HHWIL will transition high to indicate the second half of a transfer.
The falling edge of HSTRB indicates an active transfer. At the second rising edge of HSTRB, the transfer is complete and HPIC is setup.

### Setting Up the Address Register

The next step is for the host to setup the address in the HPIA register. This transfer is very similar to the HPIC setup. HCNTL selects the HPIA register. HHWIL is low for the first half of a transfer. HR/W is low to indicate a write operation. Finally, HD has the lower 16-bits of the address.
The falling edge of HSTRB indicates an active transfer. Since HWOB=1 indicating little endian, the value of the HD pins is copied into the LSBs of HPIA.

For the second half of the transfer, HCNTL and HR/W do not change. HHWIL transitions high to indicate that this is the second part of a transfer, and the host has changed the HD pins to the upper 16-bits of the address.

The falling edge of HSTRB indicates an active transfer and the address is written to the HPIA.
**Writing a 32-bit Value**

When the HPIC and the HPIA are setup, the HPI is ready to exchange data with the host. In order for the host to write to the address indicated in the HPIA, it initiates a write operation while HCNTL selects the HPID register.

**Example 1: Writing a 32-bit Value - 1**

1. HCNTL[1:0] = 11b (HPID)
   HR/W = 0, HD = 5678
   HHWIL = 0
The falling edge of HSTRB initiates the transfer, and the rising edge latches the data into the lower 16-bits of the HPID register.

Example 1: Writing a 32-bit Value - 2

1. HCNTL[1:0] = 11 (HPID)
   HR/W = 0, HD = 5678
   HHWIL = 0
2. HSTRB

For the second half of the transfer, HHWIL transitions high, and the value of the HD pins changes to reflect the upper 16-bits of data.

Example 1: Writing a 32-bit Value - 3

3. HCNTL[1:0] = 11 (HPID)
   HR/W = 0
   Write value: HHWIL = 1, HD = 1234

HSTRB falls low to indicate an active transfer. At the rising edge of HSTRB, the data is latched into the HPID. The 32-bit transfer to the HPI is now complete, but has the data actually been written to the address?
When HPID has been written, the HPI will signal the DMA Auxiliary Channel to transfer the data from the HPI to the address in the HPIA. Several factors affect the length of time that it will take for the DMA to complete this transfer. These include:

- Speed of the destination memory
- Bus contention
- DMA Auxiliary Channel Priority

If the time needed to transfer from the HPI to memory can vary, how does the host know when it can write a new value to the HPI? The HPI uses the HRDY pin to signal the host that it is busy with a current transfer. This prevents the host from overwriting information in the HPI. When HRDY is low, the HPI is ready. So, at the second rising edge of HSTRB, when all of the data is latched into the HPID, HRDY is asserted high (not ready) until the DMA has completed the transfer.

HRDY is used more as a not-ready pin to state either data is not yet available on a read or the DMA hasn’t yet completed the write (thus freeing-up the HPID).
Reading a 32-bit Value

The process for a host read operation with the HPI is similar to a write. If the HPIC and HPIA are setup, the host sets up the control pins for the first half of a read operation using appropriate values on HCNTL, HHWIL, and HR/W.

Example 2: Reading a 32-bit Value - 1

1. HCNTL[1:0] = 11b (HPID)
   HR/W = 1
   Read value: HHWIL = 0

The falling edge of HSTRB initiates a read from the address in the HPIA register. This address is copied to the DMA Auxiliary Channel.

Example 2: Reading a 32-bit Value - 2

1. HCNTL[1:0] = 11b (HPID)
   HR/W = 1
   Read value: HHWIL = 0
2. HSTRB, HPIA is copied to DMA address
At this point, the HPI has to wait for the DMA to complete the transfer from memory to the HPID register. HRDY is asserted high to hold off the host until the data is written into the HPID.

Example 2: Reading a 32-bit Value - 3

1. HCNTL[1:0] = 10b (HPID)
2. HR/W = 1
3. HSTRB, HPIA is copied to DMA address
4. HRDY is asserted until HD = 5678

The second half of the read is setup with the appropriate control signals.

Example 2: Reading a 32-bit Value - 4

1. HCNTL[1:0] = 11b (HPID)
2. HR/W = 1
3. Read value: HHWIL = 0
4. HCNTL[1:0] = 11b (HPID)
5. HR/W = 1
6. Read value: HHWIL = 1
The second half of the read begins with the second falling edge of HSTRB.

Example 2: Reading a 32-bit Value - 5

4. HCNTL[1:0] = 11 (HPID)
HR/W = 1
Read value: HHWIL = 1
5. HSTRB

What, no Not-Ready before the second 16-bit read? Since the data is already present in the HPID, HRDY is not required and will not be asserted. This is similar to a transfer to the HPIC or the HPIA. Since the value is being transferred directly to (or from) the HPI, no delay time is needed for the DMA to complete a memory transfer.

Example 2: Reading a 32-bit Value - 6

4. HCNTL[1:0] = 11 (HPID)
HR/W = 1
Read value: HHWIL = 0
5. HSTRB

K: HD = 1234
Reading Multiple Values

A nice feature of the HPI is the ability to read or write sequential word addresses without stopping to setup the HPIA every time. This is accomplished by using the HCNTL pins to select the HPID register with an autoincrement of the HPIA register.

Example 3: Sequential Accesses - 1

The read is setup exactly like a read without increment, except for the value of the HCNTL pins. The first falling edge of HSTRB initiates the first transfer. After the initial address is sent to the DMA, the address in the HPIA will automatically be incremented by four bytes.

Example 3: Sequential Accesses - 2
HRDY is asserted high while the DMA completes the memory transfer to the HPID.

The second halfword of the transfer is completed without HRDY since the data is already in the HPID.
At the second rising edge of HSTRB, when the 32-bit transfer is complete, the new address in the HPIA is copied to the DMA. The DMA uses this address to pre-fetch the data for the next transfer. This helps reduce the latency between HPI transfers. Since the DMA is busy with the pre-fetch, HRDY is asserted high. Thus, when the host tries to initiate the next transfer, it may encounter a not-ready condition until the DMA completes the memory transfer.

Example 3: Sequential Accesses - 5

HPI Pins

The HPI uses several pins to provide a glueless interface to many industry standard hosts. Several of these pins may or may not be used in any given application. Below is a summary of the typical connections.
Sidebar

**HSTRB**

HSTRB is an internal signal that is decoded from up to three host strobe signals. HSTRB is active low when both HCS is active and either HDS1 or HDS2 is active.

**HAS**

HAS is an input signal to the HPI that can be used with hosts that have multiplexed address and data lines. HAS allows the HPI to sample the control signals earlier in the access cycle so that the bus can stabilize before the data is placed on it. HAS is usually connected to the host’s Address Latch Enable (ALE) pin.
An Example Interface

The MC68360 Quad Integrated Communication Controller is a 32-bit controller that is a member of the Motorola M68300 family. It is a versatile microprocessor that can be used in a variety of control applications.

Here we can see how the address lines are connected to the HPI’s HCNTRL and HHWIL pins.
HPI Related Registers (Optional Topic)

HPIC

Earlier in the module, we briefly mentioned the HPIC, or the HPI Control Register. This register contains the Half-Word Ordering Bit, HWOB, which sets the endianness of HPI transfers. Remember that this register is mirrored across the upper and lower 16 bits.

Some of the other capabilities controlled by the HPIC are Interrupts and Software Handshaking. HPI interrupt capability is controlled by the DSPINT and HINT bits. DSPINT is one of the C6000’s interrupt sources. It allows the host to interrupt the ‘C6x via an external interrupt pin. HINT allows the ‘C6x to interrupt the host by controlling the state of the HINT output.

Software Handshaking is useful for hosts that do not have an external RDY signal. If this is the case, the host can poll the HRDY bit in the HPIC to determine the state of the HPI. Notice that this bit is active high, unlike the hardware pin HRDY. The FETCH bit initiates a read operation from the address in HPIA when it is set to 1. This capability allows the host to initiate a read operation through software.
CSL API for the Host Port Interface

**CSL HPI Support**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPI_getDspint</td>
<td>F</td>
<td>Reads the DSPINT bit from the HPIC register</td>
</tr>
<tr>
<td>HPI_getEventId</td>
<td>F</td>
<td>Obtain the IRQ event associated with the HPI device</td>
</tr>
<tr>
<td>HPI_getFetch</td>
<td>F</td>
<td>Reads the FETCH flag from the HPIC register and returns its value.</td>
</tr>
<tr>
<td>HPI_getHint</td>
<td>F</td>
<td>Returns the value of the HINT bit of the HPIC</td>
</tr>
<tr>
<td>HPI_getHrdy</td>
<td>F</td>
<td>Returns the value of the HRDY bit of the HPIC</td>
</tr>
<tr>
<td>HPI_getHwob</td>
<td>F</td>
<td>Returns the value of the HWOB bit of the HPIC</td>
</tr>
<tr>
<td>HPI_setDspint</td>
<td>F</td>
<td>Writes the value to the DSPINT field of the HPIC</td>
</tr>
<tr>
<td>HPI_setHint</td>
<td>F</td>
<td>Writes the value to the HINT field of the HPIC</td>
</tr>
<tr>
<td>HPI_SUPPORT</td>
<td>C</td>
<td>A compile time constant whose value is 1 if the device supports the HPI module</td>
</tr>
</tbody>
</table>

Note:  F = Function; C = Constant; S = Structure; T = Typedef
Expansion Bus (Optional Topic)

Most DSP systems would like to use the 32-bit parallel memory interface for several different types of devices. However, as devices are added to the bus, system performance can be affected. So, how can a system access more data without sacrificing performance?

The Expansion Bus (XB) on the ‘C6202 provides a solution to this problem. It is 32-bits wide and it provides access to off-chip peripherals, FIFOs, host processors, and PCI interface chips.
The XB includes an HPI which is very similar to the ‘C6201’s. The primary difference is that the XB is 32-bits wide.

Other important differences are that the XB can be either synchronous or asynchronous, and that it can serve as the slave or the master of the bus. These differences give the XB the ability to interface with a minimum amount of glue logic to a PCI interface. The XB also includes an internal arbiter for bus arbitration.
The XB uses the DMA Auxiliary Channel to transfer data to and from the host.

**C6000 DMA Aux. Channel**

The XBUS as the master writes to the host. The DMA Aux Ch is used to service the request of the XBUS to the ‘C6x mem map.

The XB HPI Control Register(XBHC) has a field which is used to store the frame count, XFRCT. It also includes fields to start transfers and to control interrupts.

**XBUS HPI Control Register (XBHC)**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XFRCT</td>
<td>Transfer counter when XBUS is master</td>
</tr>
</tbody>
</table>
| INTSRC | 10 - interrupt is caused when XFRCT=0  
01 - DSPINT is the interrupt source |
| START | 01 - starts a write burst  
*XBIMA to *XBEA  
10 - starts a read burst  
*XBEA to *XBIMA |
| DSPINT | External master to DSP interrupt |

Technical Training Organization
In addition to an HPI, the XB includes another sub-block, the I/O Ports. The HPI and the I/O Ports can co-exist in a system. The I/O Ports is broken up into four distinct spaces, XCE0 – XCE3. Each of these spaces has access to 16 word locations. The ‘C6202 memory map shows a 64M word block, which is really the same 16 locations aliased over and over.

Each XCEx space can access either 32-bit wide async memory, or 32-bit wide clocked FIFOs. The memory type of each space is configured in it’s XCE Control Register, in the MTYPE field.
The I/O Ports asynchronous interface uses other fields in the XCE Control Registers. These fields should look familiar, they are identical to the EMIF’s CE Control Registers. In fact, the signals used by the two interfaces are alike.

Asynchronous Interface

<table>
<thead>
<tr>
<th></th>
<th>Write Setup</th>
<th>Write Strobe</th>
<th>Write Hold</th>
<th>Read Setup</th>
</tr>
</thead>
<tbody>
<tr>
<td>RW, +1111</td>
<td>RW, +1111</td>
<td>RW, +11</td>
<td>RW, +1111</td>
<td></td>
</tr>
<tr>
<td>15 14 13 8 7 6 4 3 2 1 0</td>
<td>rsv</td>
<td>MTYPE</td>
<td>rsv</td>
<td>Read Hold</td>
</tr>
<tr>
<td></td>
<td>RW, +11111</td>
<td>R, +x</td>
<td>RW, +11</td>
<td></td>
</tr>
</tbody>
</table>

- What does this remind you of?
- An async XCE space is identical to the async EMIF
- If FIFO interface is selected, only MTYPE is used

The I/O Ports synchronous interface is designed to interface gluelessly to 32-bit clocked FIFOs. The I/O Ports can interface up to 3 write FIFOs and one read FIFO (located in XCE3) without any glue. A minimum amount of glue can be used to expand the capabilities of this interface to include other sizes of FIFOs (8 and 16 bit) and up to 16 read and write FIFOs per XCE space.

Synchronous Interface

Note: XOE is only enabled in XCE3 for a glueless read interface.
XB Summary

The XB, composed of the HPI and the I/O Ports, adds five new “ports” for accessing hosts and peripherals. Each of these ports can operate in an asynchronous mode or a synchronous mode. Each mode provides different capabilities, which can make your system easier to design and implement.

<table>
<thead>
<tr>
<th>Port</th>
<th>Async</th>
<th>Sync</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPI</td>
<td>Slave only</td>
<td>Master/Slave</td>
</tr>
<tr>
<td>XCE0</td>
<td>16 word addresses 16 read/16 write</td>
<td>No Glue  Glue</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>16 R/W</td>
</tr>
<tr>
<td>XCE1</td>
<td>16 word addresses 16 read/16 write</td>
<td>No Glue  Glue</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>16 R/W</td>
</tr>
<tr>
<td>XCE2</td>
<td>16 word addresses 16 read/16 write</td>
<td>No Glue  Glue</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>16 R/W</td>
</tr>
<tr>
<td>XCE3</td>
<td>16 word addresses 16 read/16 write</td>
<td>No Glue  Glue</td>
</tr>
<tr>
<td></td>
<td>Read</td>
<td>16 R/W</td>
</tr>
</tbody>
</table>