INTERFACING PCI with DSP

Group No.: D7
Ankit Kulin (01D07015), Amit Ghorawat (01D07017),
Saurabh Goyal (01D07035)
Supervisor: Prof. Mukul Chandorkar

ABSTRACT

PCI devices form the best way of interfacing a PC with an external device if high data speed with minimum CPU overhead is required. We attempt to build one such interface using TI's PCI 2040 which would be a general purpose device enabling any user to establish communication between PC and DSP through the Host Port Interface of the later. This report provides the procedure we followed to complete the task.

CONTENTS

List of Figures

1. Problem Definition

The primary objective of EDL Project is to design a PCI Device to enable communication through the Host Port Interface (HPI) of the DSP. We aim at data transfer rates upto 1.5 MB/s. The project involves designing proper hardware and writing of the proper device driver to enable the PC to perform the required data transfer. The above is implemented using a TI's PCI 2040.

Our hardware will provide a **general purpose interface** to connect PC to any DSP through the Host port interface. This will be much **faster** than currently used JTAG which is a serial interface and much **cheaper** than the TI JTAG interface which costs about $1500. The debugging and downloading software needs to be developed for our PCI interface. The other option was USB interface to DSP which is also an ongoing project. That is also a serial interface.

Our device would also provide a flexible way to transfer data at high speeds from any other device to PC. The PCI 2040 provides a clean interface with PCI bus on one side and HPI on other side.

The presence of DSP increases the flexibility. The DSP can be used for performing operations of data. These operations will be fully in control of user, free of any operating system interferences.

2. The PCI BUS

The PCI bus is an integral part of today's high-performance personal computer systems. Conceived and designed as a way to give peripheral components high-bandwidth access to the host processor in a PC, the PCI bus is a board-level expansion standard with important benefits to anyone whose work involves PC-based data acquisition. The PCI Local Bus is a high performance bus for interconnecting chips, expansion boards, and processor/memory subsystems. It is synchronous bus architecture with all data transfers being performed relative to a system clock (CLK).

The PCI bus breaks open the bandwidth bottleneck by providing a 132 MB/s (theoretical), 95 MB/s (typical) burst-rate highway. In addition to its high bandwidth, the PCI bus features master/slave operation to reduce latency and offload the host CPU. Other advantages include plug-and-play auto configuration to simplify installation; processor and platform independence, which lets designers easily develop PCI peripherals to run on other platforms than the Pentium PC; and a specified migration path to 3.3 V, which simplifies the design of portable systems.

2.1 Salient features of PCI for Data Acquisition Applications

- For data acquisition, the high-bandwidth of the PCI bus allows simultaneous, real-time gathering analog and digital input data, along with outputs for analog stimuli and digital control. Data acquisition boards built for the PCI bus can feed acquired data directly to the PC's memory, minimizing the need for onboard memory.

- PCI data acquisition boards do not leave gaps in the acquired data. In other words, PCI boards do not lose data like ISA boards when the latter does not respond fast enough to the board's request to transfer data. High bandwidth also ensures gap-free data transfer by allowing several subsystems to be active simultaneously rather than sequentially.
- Because boards on the PCI bus can transfer without intervention from the host CPU, data can be simultaneously acquired and processed in real time. For example, as data is sampled, the host CPU can perform a mathematical operation, plugging the results into a spreadsheet for analysis. Moreover, by adding digital I/O functions, you can set up real-time collection and control experiments.
- In its original implementation PCI ran at 33MHz. This was raised to 66MHz by the later PCI 2.1 specification It can be configured both as a 32-bit and a 64-bit bus, and both 32-bit and 64-bit cards can be used in either. Also, PCI bus mastering reduces latency and results in improved system speeds.

Table 2-1 Advantages of PCI I/O over USB I/O

Attributes	PCI	USB
Type of bus	Parallel	Serial I/O
System overhead	Less	More
Bus width (data bits)	32/64	1
Peak bandwidth (MB/s)	133-512	0.2-15

2.2 PCI Bus Protocol

PCI implements a 32-bit multiplexed Address and Data bus (AD[31:0]). It architects a means of supporting a 64-bit data bus through a longer connector slot, but most of today's personal computers support only 32-bit data transfers through the base 32-bit PCI connector.

The multiplexed Address and Data bus allows a reduced pin count on the PCI connector that enables lower cost and smaller package size for PCI components. Typical 32-bit PCI add-in boards use only about 50 signals pins on the PCI connector of which 32 are the multiplexed Address and Data bus. PCI bus cycles are initiated by driving an address onto the AD[31:0] signals during the first clock edge called the address phase. The address phase is signaled by the activation of the FRAME# signal. The next clock edge begins the first of one or more data phases in which data is transferred over the AD[31:0] signals.

In PCI terminology, data is transferred between an initiator which is the bus master, and a target which is the bus slave. The initiator drives the C/BE[3:0]# signals during the address phase to signal the type of transfer (memory read, memory write, I/O read, I/O write, etc. During data phases the C/BE[3:0]# signals serve as byte enable to indicate which data bytes are valid. Both the initiator and target may insert wait states into the data transfer by de-asserting the IRDY# and TRDY# signals. Valid data transfers occur on each clock edge in which both IRDY# and TRDY# are asserted.

A PCI bus transfer consists of one address phase and any number of data phases. I/O operations that access registers within PCI targets typically have only a single data phase. Memory transfers that move blocks of data consist of multiple data phases that read or

write multiple consecutive memory locations. Both the initiator and target may terminate a bus transfer sequence at any time. The initiator signals completion of the bus transfer by de-asserting the FRAME# signal during the last data phase. A target may terminate a bus transfer by asserting the STOP# signal. When the initiator detects an active STOP# signal, it must terminate the current bus transfer and re-arbitrate for the bus before continuing. If STOP# is asserted without any data phases completing, the target has issued a retry. If STOP# is asserted after one or more data phases have successfully completed, the target has issued a disconnect.

Initiators arbitrate for ownership of the bus by asserting a REQ# signal to a central arbiter. The arbiter grants ownership of the bus by asserting the GNT# signal. REQ# and GNT# are unique on a per slot basis allowing the arbiter to implement a bus fairness algorithm. Arbitration in PCI is "hidden" in the sense that it does not consume clock cycles. The current initiator's bus transfers are overlapped with the arbitration process that determines the next owner of the bus.

PCI supports a rigorous auto configuration mechanism. Each PCI device includes a set of configuration registers that allow identification of the type of device (SCSI, video, Ethernet, etc.) and the company that produced it. Other registers allow configuration of the device's I/O addresses, memory addresses, interrupt levels, etc.

PCI defines support for both 5 Volt and 3.3 Volt signaling levels. The PCI connector defines pin locations for both the 5 Volt and 3.3 Volt levels. However, most early PCI systems were 5 Volt only, and did not provide active power on the 3.3 Volt connector pins. Over time more use of the 3.3 Volt interface is expected, but add-in boards which must work in older legacy systems are restricted to using only the 5 Volt supply. We will be using 5 V supply as the socket for 5V is easily available on all motherboards currently in use.

Table 2-2 Byte enables for various commands in PCI bus.

C/BE[3:0]#	Command Types
0000	Interrupt Acknowledge
0001	Special Cycle
0010	I/O Read
0011	I/O Write
0100	Reserved
0101	Reserved
0110	Memory Read
0111	Memory Write
1000	Reserved
1001	Reserved
1010	Configuration Read
1011	Configuration Write
1100	Memory Read Multiple
1101	Dual Address Cycle
1110	Memory Read Line
1111	Memory Write and Invalidate

3. The Host Port Interface of DSP

The HPI is an 8-bit parallel port used to interface a host device or host processor to a C54x DSP. Information is exchanged between the DSP and the host device through on-chip C54x memory that is accessible by both the host and the DSP.

The HPI is designed to interface to the host device as a peripheral, with the host device as the master of the interface, and so facilitating the ease of access by the host. The host device communicates with the HPI through dedicated address and data registers, to which the DSP does not have direct access, and the HPI control register using the external data and interface control signals. Both host devices and the HPI have access to the HPI control register.

In C54x, the HPI provides 16-bit data to the DSP while maintaining an external interface of 8-bit by automatically combining the successive bytes into 16-bit words. When the host performs a data transfer with the HPI registers, the HPI control logic automatically performs an access to DSP's memory to complete the transaction. The DSP can then access the data within its memory space.

The DSP can work independently of the transactions on HPI interface. So Accessing HPI doesn't affect DSP performance. Also Host can have access to DSP memory during its RESET state also.

3.1 Modes of Operation

In C54x, the HPI has two modes of operation as follows:

Shared access mode (SAM): This is the normal mode of operation and in this mode both the DSP and host can access the HPI memory. In this case, the asynchronous host accesses are resynchronized internally. In the case of a conflict between the DSP and host, host has access

Host only mode (HOM): In this mode, only the host can access the HPI memory while the DSP is in reset state or IDLE2 with all internal or external clocks stopped. This mode allows host to access the HPI memory while the DSP is in minimum power consumption configuration. In HOM, the HPI supports higher speed back-to-back accesses on the order of 1 byte/50 ns (160 MB/s) independent of the DSP's clock rate.

3.2 HPI Functional Description

In C54x, information is exchanged between the host and the DSP via 8-bit external data bus. Due to the 16-bit word structure of the C54, all transfers consist of two consecutive bytes. The dedicated HWIL pin indicates whether the first or second byte is being transferred. Bits 0 and 8 (BOB) in the HPI control register determines whether the first byte is MSB or LSB. The host must not break the first/second byte sequence, otherwise the data may be lost or some unpredictable results may happen.

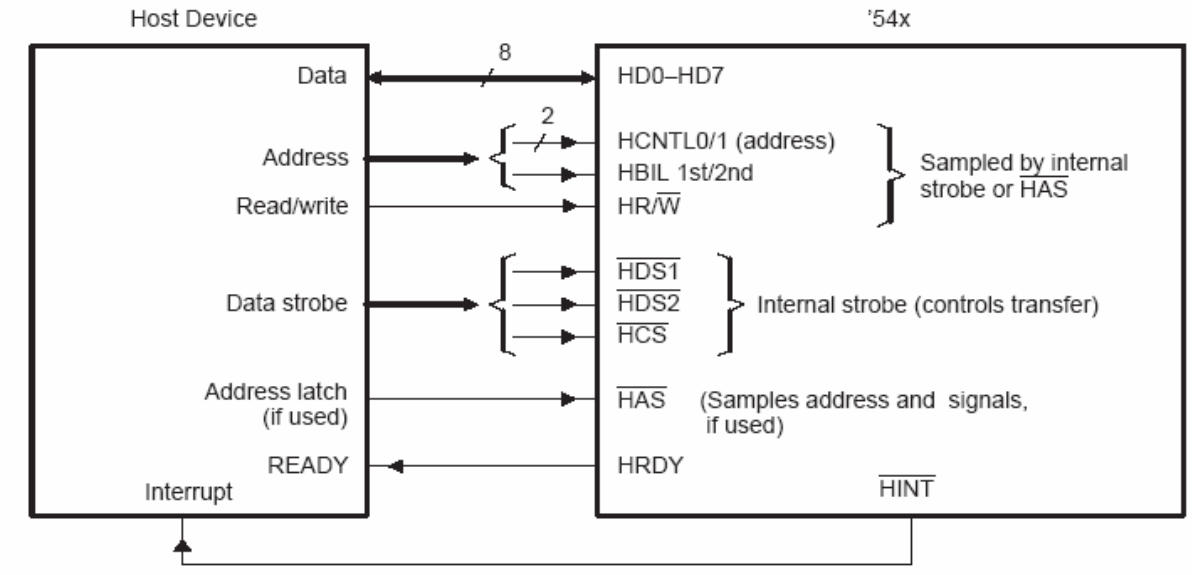


Figure 1: Block Diagram of Generic HPI Interface

3.3 HPI Registers

The HPI utilizes three registers for communication between the host device and the CPU. These registers are:

HPI address register (HPIA). It is directly accessible only by the host and contains the address in HPI memory at which the current address access occurs.

HPI control register (HPIC). It is directly accessed by the host or by C54x and contains the control and status bits for HPI operation.

HPI data register (HPID). This register is directly accessible by the host and contains the data that was read from the HPI memory if the current access is a read, or the data that will be written to the HPI memory if the current access is a write.

The two control inputs, **HCNTL1 and HCNTL0**, indicate which internal register is being accessed as shown below.

Table 3–1. C54X HPI Registers Access Control

HCNTL1	HCNTL0	DESCRIPTION
0	0	PCI2040 read/write to HPI control register.
0	1	PCI2040 read/write to HPI data register. Address auto-increment is selected.
1	0	0 PCI2040 read/write to HPI address register
1	1	PCI2040 read/write to HPI data register. Address auto-increment is not selected

3.3.1 C54X HPI Control Register

Upper word of control register is same as lower nibbles. The higher nibble of each word is reserved for future applications. With the DSPs having extended memory space the LSB of higher nibble of each word is XHPIA. This is to enable extended HPI Memory Space. If this bit is set to 1 the Address Register will point to extended HPI Memory Space. The functions of lower nibble bits are as in following table.

Table 3-2 Description of HPI Control Register

Bit No	Name	Description
3	HINT	This bit determines the state of the DSP HINT output which is used to generate an interrupt to the host. HINT= 0 after reset. The HINT can be set only by the DSP by writing a 1 to this bit and can be cleared only by the host writing a 1 to this bit.
2	DSPINT	Host to DSP interrupt. This bit can only be written by the host and is not readable by the host or the DSP. When the host writes a 1 to this bit, an interrupt is generated to the DSP. Writing a 0 has no effect.
1	SMOD	This bit determines the mode of operation. 0 = HOM is selected (always during reset) 1 = SAM is selected BOB affects both data and address transfers. Only the host can modify this Bbit and it is not visible to the DSP. BOB must be initialized before the first data or address register access. 0 = First byte is MS 1 = First byte is the LS
0	BOB	BOB affects both data and address transfers. Only the host can modify this bit and it is not visible to the DSP. BOB must be initialized before the first data or address register access. 0 = First byte is MS 1 = First byte is the LS

3.4 Auto Increment Feature

The HPI data register can be accessed with optional auto-address increment. It provides a convenient way of reading or writing to subsequent word locations. In the auto-increment mode, the data read causes a post increment of the HPI address register and a data write causes a pre increment of the HPI address register. Because the HPI has 2Kx16-bit emory, it uses only the 11 LSBs of the HPI address register but, during the auto-increment operation, all 16 bits will be incremented or decremented.

3.5 Host Read/Write Access to HPI

The host begins accessing the HPI interface first by initializing the HPI control register, then by initializing the HPI address register, and then by reading data from or writing data to the HPI data register. Writing to the HPI address or HPI data register initiates an internal cycle that transfers the desired data between the HPI data register and the internal HPI memory. This process may take several cycles. Each time an access is made, data written to HPI data register is not written to HPI memory until after the host access cycle and the data read from HPI data register is the data from the previous cycle. Therefore, when reading, the data is obtained from the location specified in the previous access and the current access serves as an initiation of the next cycle. A similar operation occurs for the

write operation. The data written to the HPI data register is not written to HPI memory until after the external cycle is completed. If the HPI data register read operation immediately follows an HPI data register write operation, then the same data (the data written) is read

3.6 HPI Memory Access during Reset

The DSP is not operational during reset, but the host can access the HPI hereby allowing the program or data to be downloaded to the HPI memory. However to use this capability, it is convenient for the host to control the DSP's reset. Initially, the host stops accessing the HPI at least six DSP periods before driving the DSP reset line low. The HPI mode is set to HOM during the reset and the host can start accessing the HPI after four DSP periods.

3.7 HPI Memory Map

The host uses the HPIA register as a pointer to '54x on-chip memory, and all on-chip RAM locations are accessible through the HPI-8. Because the internal memory map of each '54x device is unique, the address range that the HPI-8 can access varies from device to device. For example, the 'VC5410 includes much more on-chip RAM than the 'VC5402.

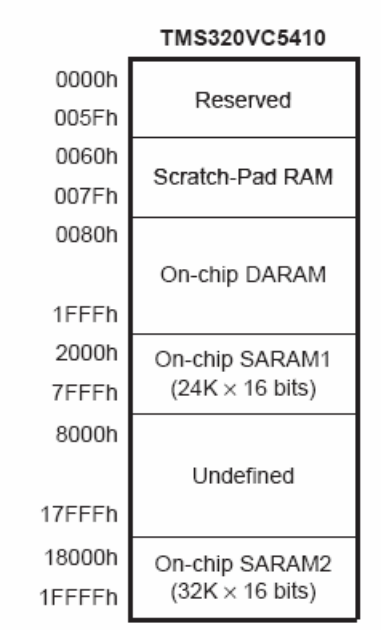


Figure 2: HPI-8 Memory Map

4. PCI 2040

The TI PCI2040 is a PCI-DSP bridge that provides a glue less connection between the 8-bit host port interface (HPI) port on the TMS320C54X or the 16-bit HPI port on

TMS320C6X to the high performance PCI bus. It provides a PCIbus target interface compliant with the PCI Local Bus Specification. The PCI2040 provides several external interfaces: the PCI bus interface with compact PCI support, the HPI port interface with support for up to four DSPs, a serial ROM interface, a general-purpose input/output interface (GPIOs), and a 16-bit general-purpose bus to provide a glue less interface to TI JTAG test bus controller (TBC). The PCI2040 universal target-only PCI interface is compatible with 3.3-V or 5-V signaling environments. The PCI2040 interfaces with DSPs via a data bus (HPI port).

4.1 PCI Interface of PCI 2040

PCI2040 provides an integrated 32-bit PCI bus interface compliant with the PCI Local Bus Specification. The PCI2040 incorporates a PCI target interface for configuration cycles, accesses to internal registers, and access to the HPI interface via memory-mapped space. The PCI2040 does not provide PCI mastering.

As a PCI bus target, PCI2040 incorporates the following features:

1. Supports the memory read, memory write, configuration read, and configuration write.
2. Aliases the memory read multiple, memory read line, and memory write and invalidate to the basic memory commands (i.e., memory read and memory write).
3. Supports PCI_LOCK.

4.1.1 Internal PCI2040 Registers Used for Controlling PCI Interface

PCI configuration registers

PCI configuration space is accessed via PCI configuration read and PCI configuration write cycles. These registers may be accessed using byte, word, or double-word transfers. The important of these are:-

Table 4-1. PCI 2040 Configuration Register

REGISTER NAME				OFFSET
Device ID		Vendor ID		00h
Status		Command		04h
Class code		Revision ID		08h
BIST	Header type	Latency timer	Cache line size	0Ch
HPI CSR memory base address				10h
Control space base address				14h
GPBus base address				18h
Reserved				1Ch
Reserved				20h
Reserved				24h
Reserved				28h
Subsystem ID		Subsystem vendor ID		2Ch
Reserved				30h
Reserved	Reserved	Reserved	Capability pointer	34h

Reserved				38h
Max_Lat	Min_GNT	Interrupt pin	Interrupt line	3Ch
Reserved		Reserved		40h
GPIO output data	GPIO direction	control GPIO input	data GPIO select	44h
Reserved	Reserved	Reserved	GPIO interrupt type	48h
Diagnostic	Reserved	Miscellaneous control		4Ch
Power management capabilities		PM next-item pointer	PM capability ID	50h
Reserved		PM control/status		54h
HPI CSR I/O base address				58h
Reserved	HS_CSR	HS next-item pointer HS	capability ID	5Ch
Reserved	Reserved	Reserved	Reserved	60h
Reserved				64h-FFh

PCI command register

This register is provided to enable coarse control over a device's ability to generate and respond to PCI cycles. As far as PCI-2040 is concerned only 4 bits can be written. Rests all are hardwired. The details of those bits are as follows:-

Table 4-2. Writable bits of PCI command register

BIT No/Name	Description
8th bit (SERR_EN) System error (SERR) enable.	This bit is an enable for the output driver on the SERR pin. If this bit is cleared and a system error condition is set inside PCI2040, then the error signal will not appear on the external SERR pin.
6th bit (PERR_EN) Parity error response enable.	This bit controls whether or not the device responds to detected parity errors. If this bit is set, then the PCI2040 responds normally to parity errors. If this bit is cleared, then the PCI2040 ignores detected parity errors.
1st bit (MEM_EN) Memory space enable.	This bit enables the device to respond to memory accesses to any of the defined base address memory regions. If this bit is cleared, then the PCI2040 will not respond to memory-mapped accesses.
0th bit (IO_EN) I/O space control	This bit enables the device to respond to I/O accesses within its defined base address register I/O regions.

HPI control and status registers

The PCI2040 provides a set of registers specifically for interfacing with the HPI port. These registers are called the HPI control and status registers (HPI CSRs), and they may be memory- and I/O-mapped.

The HPI CSR memory base address register provides the mechanism for mapping the HPI CSRs into memory space. When mapped into memory space, the HPI CSRs may be accessed using bytes, words, or double-word transfers. Memory mapping the HPI CSR registers is recommended.

Table 4-3. CSR Memory Base Address Register

Bit number	Description
31–12	Available address bits. These bits can be written by the host in order to allow initialization of the base address at startup. The PCI memory address space is on the 4-Kbyte boundary.
11–4	Unavailable address bit. Bits 11–4 return 00h when read.
3-0	Hardwired to 0 is pci2040

Table 4-4. HPI Control and Status Registers. These are Memory or I/O mapped. The base address is stored in Base Address Registers in PCI 2040 Configuration Registers.

REGISTER NAME		OFFSET
Interrupt event set		00h
Interrupt event clear		04h
Interrupt mask set		08h
Interrupt mask clear		0Ch
Reserved	HPI error report	10h
HPI DSP implementation	HPI reset	14h
Reserved	HPI data width	18h

4.2 PCI2040 Host Port Interface

The PCI2040 HPI interface is used to access TI's TMS320C54X or TMS320C6X DSP chips. The devices connected to the HPI interface are memory-mapped in host memory. The host system processor accesses the HPI interface via slave accesses to PCI2040. The DSP devices can generate interrupts, and the PCI2040 passes these interrupt requests to the PCI bus via INTA. See Section 3.7, Interrupts, for more information on PCI2040 interrupts. The HPI port on DSP devices is a parallel port that allows access to the DSP's memory space and internal registers. The PCI2040 has to configure the HPI interface on the DSP by accessing the DSP's HPI control register (HPIC). Other DSP HPI registers include the HPI data register (HPID) and the HPI address register (HPIA). See Section 6, DSP HPI Overview for more information on DSP registers.

4.2.1 Identifying Implemented Ports and DSP Types

The PCI2040 supports up to four DSPs of both the C54x and C6x types. It may be useful for generic software to discover what number and type of DSPs are connected to the PCI2040. This is accomplished by using the HPI DSP implementation register and HPI data width register in the HPI control and status register space. The HPI DSP implementation register identifies how many DSPs are implemented and what HCSn outputs are connected, and the HPI data width register identifies whether the HPI port per connected DSP is 8 bits (C54x) or 16 bits (C6x).The HPI DSP implementation register and HPI data width register may be loaded from a serial ROM. Also, these registers are implemented as read/write so intelligent software can load them with the proper values.

Table 4-5. HPI DSP Implementation Register

BIT	FIELD NAME	TYPE	DESCRIPTION
15–4	RSVD	R	Reserved. Bits 15–4 return 0s when read.
3	DSP_PRSNT3	RWU	DSP3 present. Bit 3 indicates if the DSP3 is present on the HPI interface.
2	DSP_PRSNT2	RWU	DSP2 present. Bit 2 indicates if the DSP2 is present on the HPI interface.
1	DSP_PRSNT1	RWU	DSP1 present. Bit 1 indicates if the DSP1 is present on the HPI interface.
0	DSP_PRSNT0	RWU.	DSP0 present. Bit 0 indicates if the DSP0 is present on the HPI interface.

Table 4-6. HPI Data Width Register

BIT	FIELD NAME	TYPE	DESCRIPTION
15–4	RSVD	R	Reserved. Bits 15–4 return 0s when read.
3	DWIDTH3	RWU	When bit 3 is set, the HPI[3] data bus is 16 bits (C6x). When bit 3 is 0, it is 8 bits (C54x).
2	DWIDTH2	RWU.	When bit 2 is set, the HPI[2] data bus is 16 bits (C6x). When bit 2 is 0, it is 8 bits (C54x)
1	DWIDTH1	RWU	When bit 1 is set, the HPI[1] data bus is 16 bits (C6x). When bit 1 is 0, it is 8 bits (C54x).
0	DWIDTH0	RWU	When bit 0 is set, the HPI[0] data bus is 16 bits (C6x). When bit 0 is 0, it is 8 bits (C54x).

4.2.2 Decoding the Address of PCI cycle to get Memory Mapping, Chip Select and HPI Register Access Control

The PCI2040's control space base address register is a standard PCI base address register requesting 32K bytes of control space non pre-fetchable memory to access up to four DSPs. The PCI2040 claims PCI memory access transactions that fall within the 32-Kbyte

memory window by comparing the upper 17 bits of the PCI address (PCI_AD31–PCI_AD15) to bits 31–15 (AVAIL_ADD field) in the control space base address register.

Table 4-7. Control Space Base Address Register

31-15 (AVAIL_ADD)	Available address bits. Bits 31–15 allow the host to map the PCI2040's 32K bytes of control space into memory. See Sections 3.5.2, DSP Chip Selects, and 3.5.3, HPI Register Access Control, for details on addressing the control space.
14-4	RSVD
3-0	Hardwired to 0.

The PCI2040 provides four chip select outputs (HCS3–HCS0) that uniquely select each HPI port DSP (or other HPI peripheral) per transaction. When a PCI cycle is claimed, the chip select is determined by decoding bits 14 and 13 of the PCI address. PCI_AD14 and PCI_AD13 determine the chip select according to Table-3.8.

Table 4-8. PCI2040 Chip Select Decoding

PCI_AD(14–13)	CHIP SELECT ASSERTED
2'b00	HCS0
2'b01	HCS1
2'b10	HCS2
2'b11	HCS3

The HCNTL1 and HCNTL0 terminals are driven by the PCI2040 to select the DSP HPI register and access mode on a cycle-by-cycle basis. When a cycle is claimed by decoding PCI_AD31–PCI_AD15, the HCNTL1 and HCNTL0 control signals are determined by decoding bits 12 and 11 of PCI address. PCI_AD12 maps to HCNTL1 and PCI_AD11 maps toHCNTL0, and the selected HCNTL1 and HCNTL0 are driven to the HPI interface when the cycle is forwarded.

The PCI address bits PCI_AD10–PCI_AD0 are not forwarded to the HPI interface, and these address bits are not decoded by PCI2040 for any purpose. This 2-Kbyte of addressable space per DSP (and control) allows the host to directly map 2K bytes of host memory to the HPI interface for each DSP. This allows for fast memory block copies rather than an I/O port mechanism. The PCI2040 does not automatically generate accesses to the HPI address registers based upon PCI_AD10–PCI_AD0, and it is left to software to synchronize the HPI address register with copies to and from HPI memory space.

4.3 Read Write Procedure

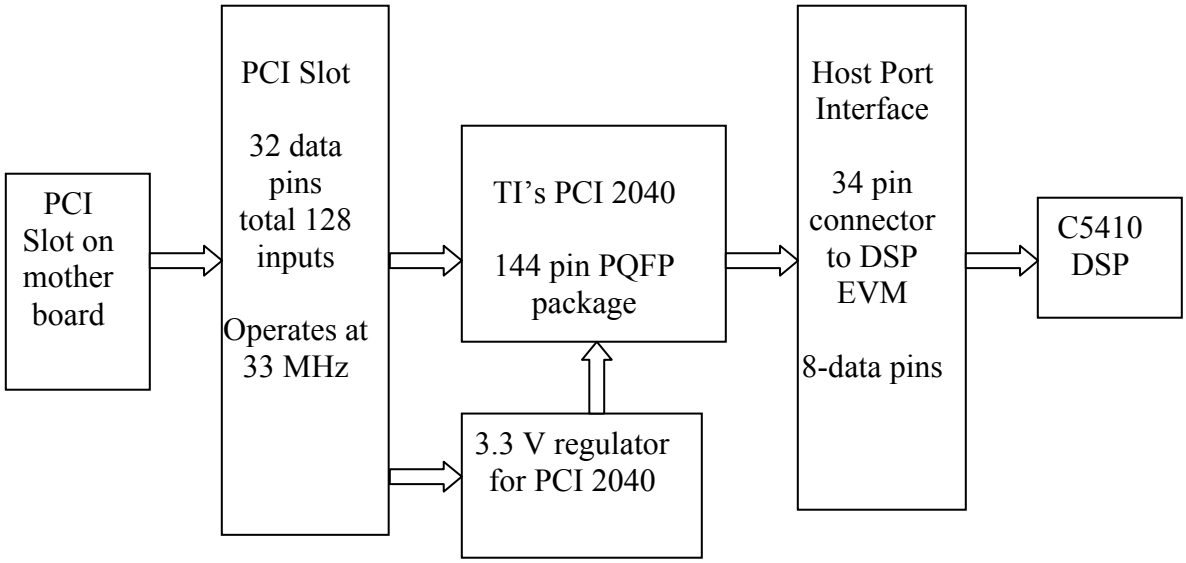
The following procedure illustrates how to read and write HPI space, and covers some of the initialization that must be done to successfully transfer data to and from DSP memory via the HPI data register. After a power-on reset (GRST):

- PCI2040 preloads several registers if a serial ROM is implemented, and this rewrites the HPI implementation and HPI data width registers (software can also rewrite these registers).
- HPI CSR memory base address register is programmed to provide a pointer to the HPI control and status registers.
- Control space base address register is programmed and 32K bytes of memory are allocated.
- The PCI command register is programmed to allow PCI2040 to respond to memory and I/O cycles.
- Software must clear the HPI reset register to remove the reset assertion to the DSPs.
- When PCI2040 decodes a PCI address within the 32-Kbyte memory control space window, it claims the cycle and decodes the chip select, HCNTL1 and HCNTL0, to pass to the HPI interface.
- The host initializes the BOB or HWOB bit in the HPI control register to choose the correct byte alignment. This results in an HPI cycle to the DSP's HPI control register.
- The host then initializes the HPI address register with the correct HPI memory address. By loading the HPI address register, an internal DSP HPI memory access is initiated and the data is latched in the HPI data register.
- If this is a read:
  - The host performs a read of the HPI data register. During the read, the contents of the first half-word data latch appear on the HADn pins when the HWIL signal is low and contents of the second data latch when the HWIL signal is high.
  - If auto-increment is selected, then it occurs between the transfer of the first and second bytes. This allows back-to-back HPI data register accesses without an intervening HPI address register access.
- If this is a write:
  - The first data latch of HPI data register is written from the data coming from the host while HWIL is low and the second data latch when HWIL is high. If communicating with C6x, then the correct combination of byte enables must also be used.
  - If auto-increment is selected, then it occurs between the transfer of the first and second bytes.

5. Interfacing DSP to HPI by PCI 2040

We have completed the interfacing the DSP to HPI. This section mainly presents the work done by us to complete the interfacing. This section includes the block diagram, circuit diagram, system configuration used and design issues

5.1 Block Diagram



Host CPU: Pentium I (166 MHz)
DSP kit used: TMS320C54XX Evaluation Module.

5.2 Power and Signaling Levels

The PCI2040 supports both 3.3-V and 5-V signal environments. This is accomplished by the VCCH and VCCP clamping rails. These two rails are not power rails. They only clamp the signals at the rail voltage (3.3 V or 5 V). All I/O buffers are powered by the VCC rail. Because VCC powers both the core and the I/O buffers, it must always be at 3.3 V. Table describes the different voltage combinations supported by the PCI2040.

Table 5-1:- PCI 2040 Signaling Levels

PCI Bus Signaling Level	HPI Bus Signaling Level	V_{CCP}	V_{CCH}	V_{CC}
5	5	5	5	3.3
5	3.3	5	3.3	3.3
3.3	5	3.3	5	3.3
3.3	3.3	3.3	3.3	3.3

We have selected 5 V PCI signaling level. DSP provides us 3.3 I/O signaling levels for HPI interface. So for our circuit V_{CCP} pin of PCI 2040 will have level of 5 V, and other V_{CCH} pins will be connected to 3.3 volt power supply. We will have to generate 3.3 on-board. So we are using a separate power supply circuit.

5.3 Power Supply using 3.3 V regulator

PCI 2040 works at I/O voltage level of 3.3 volts. As we are using 5 V I/O convention We are using 5 V directly from PCI Bus and converting it to 3.3 V by using low drop out voltage regulator TPS 7333Q. The circuit diagram of the power supply is as shown in figure below:-

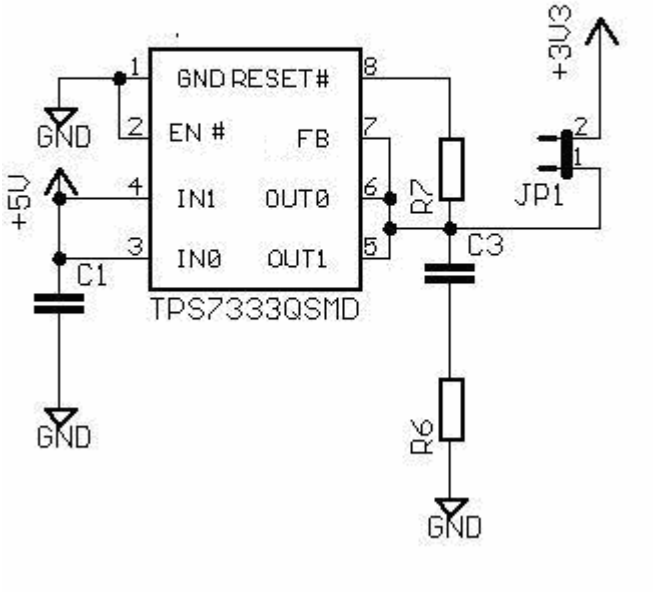


Figure 3. Voltage regulator

5.4 Pull – up Resistors

The pull up resistors forms a crucial part of the circuit. The PCI2040 require pullup resistors for various PCI and HPI signals.

PCI Pull-up Resistors

All PCI control signals require pull-up resistors on the motherboard to ensure they are at a stable state when no agent is actively driving the signal. Pull-ups should be implemented on the motherboard only; expansion boards or add-in cards should not provide pull-up resistors for the PCI control signals. The PCI signals which require pull-ups are FRAME, TRDY, IRDY, DEVSEL, STOP, SERR, PERR, LOCK, INTA, INTB, INTC, INTD, and when used REQ64 and ACK64. Pull-ups are not required on point-to-point or shared 32-bit signals, as bus parking ensures their stability.

Table 5-2: Minimum and typical PCI Pull-up Resistor Values

Signaling Rail	R_{min}	R_{typ}
5 V	963 kΩ	2.7 kΩ ± 10%
3.3 V	2.42 kΩ	8.2 kΩ ± 10%

HPI Pull-up Resistors

Pull-up resistors are required on all unused HPI and GPbus inputs in order to prevent oscillation and increased power consumption. Pull-up resistors are not required on HAD[15:0] due to the fact that the PCI2040 drives these signals to stable values during idle times.

Table 5-3. HPI/GP Bus Pullup resistor

HPI/GPbus	Signal Pullup Voltage
GPINT#	V_{CCH}
HINT[3:0] #	V_{CCH}
HRDY5x[3:0]	V_{CCH}

The GPRDY signal is used to notify the PCI2040 whether the device on the GPbus is ready for a transaction. Because this signal is active low, pulling this signal up will always put the GPbus in a not ready state. As we are not using GPbus we have pulled up this signal. The recommended value of all pull-ups mentioned above is 10kΩ.

DSP Pull-up Resistors

The PCI2040 uses only one data strobe (HDS) and does not implement HAS. Because of this, the HAS on the DSP needs to be pulled up and one of the two HDS lines on the DSP needs to be pulled up. The other HDS line needs to be connected to the HDS signal on the PCI2040. The pull-up value to be used is 10kΩ.

Other Pull-up Resistors

PME: A 43 kΩ pull-up resistor to 3.3 V is required on the open-drain PME (pin 68).
HSENUM: A 43 kΩ pull-up resistor to 3.3 V is required on the open-drain HSENUM (pin71). A pull-up resistor may already exist on the motherboard.
Two-Wire Serial Interface Signals, SDA and SCL: When implementing the serial EEPROM via generic two-wire serial bus interface on general-purpose terminals GPIO0 and GPIO1, pull-up resistors are required on the open-drain SCL and SDA signals. For typical PCI2040 serial EEPROM applications, a pull-up of between 2 kΩ and 10 kΩ is recommended. We have given a provision for EEPROM in the circuit.

Bypass Capacitors

Low inductance ceramic chip capacitors are best for bypass capacitors. A value of 0.1μF is recommended for each of the power supply pins V_{CC}, V_{CCP}, and V_{CCH}.

5.5 Detailed Circuit Diagram

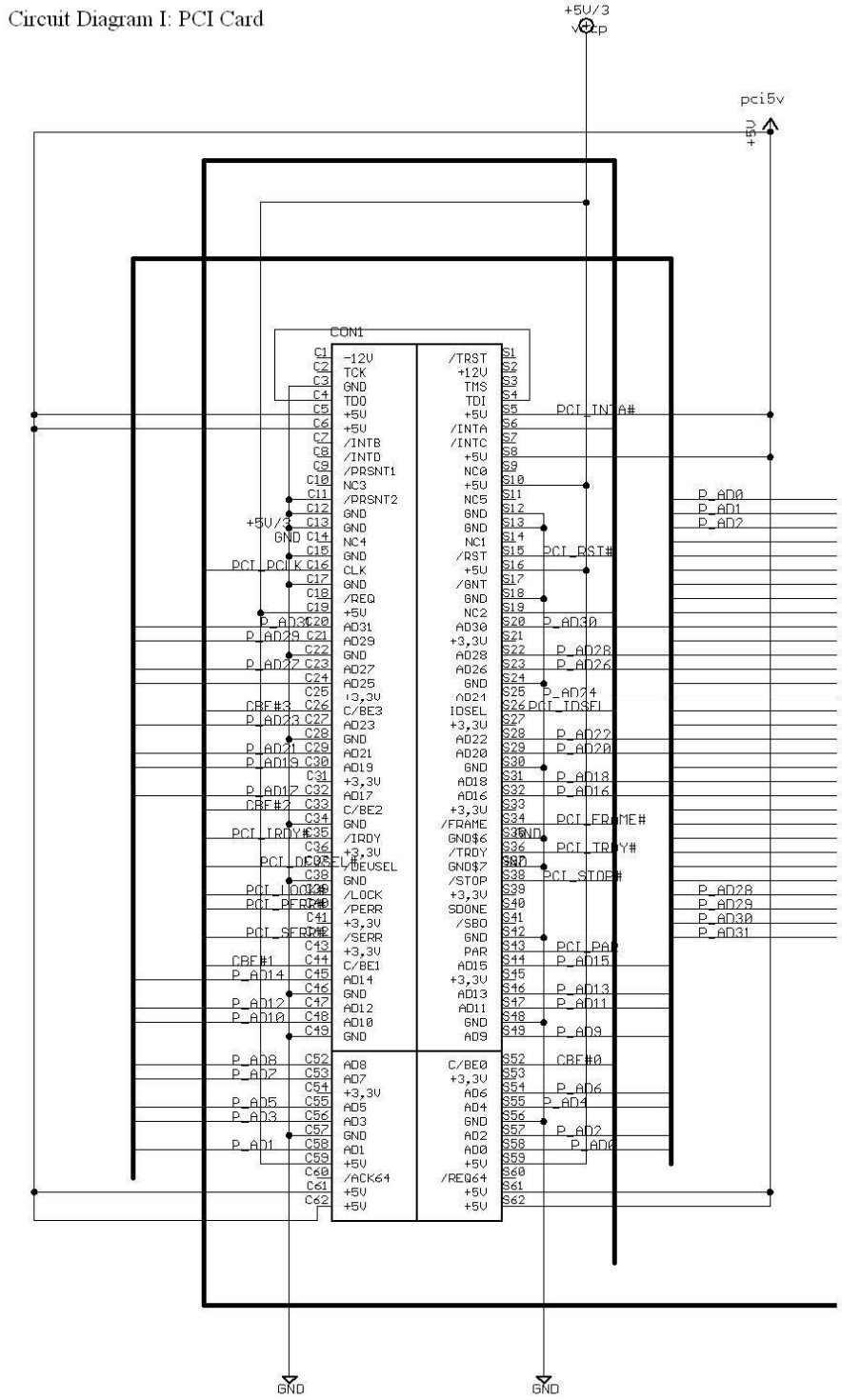


Figure 4. Detailed circuit diagram of PCI card

Circuit Diagram II :
PCI 2040 and the Host Port Interface of the DSP

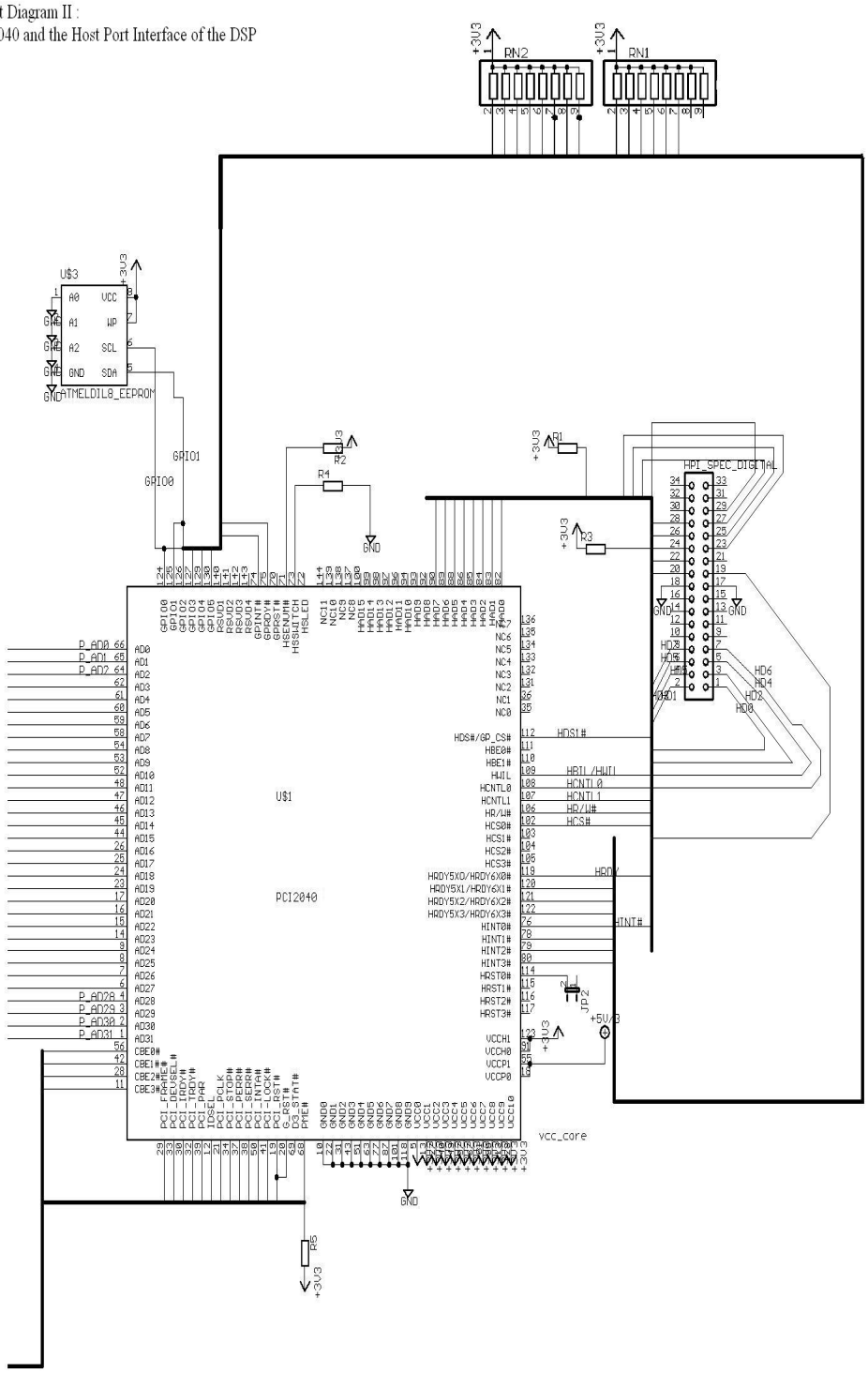


Figure 5. Detailed circuit diagram of PCI card (contd.)

6. Details of the Software

We have to write a device driver for PCI-2040. The driver we have made enables us to read from and write to any DSP memory location. We can manually verify the same using code composer studio for the DSP that has a provision to show DSP memory. We explain here the important software considerations, the algorithm for reading and writing in DSP memory and give some examples with the timing diagrams for the same.

6.1 Decoding of Chip Select, HCNTL0 and HCNTL1

As we have explained earlier**,** During the PCI address phase, AD[12:11] determine which HPI register in the DSP is being accessed. The AD[12] maps to HCNTL1 and AD[11] maps to HCNTL0. We will be using chip select 0. This can be selected by AD[14:13] by making them 00. So we have the following:

PCI address	**HPI register where data would be written**
0000	Control Register(HPIC)
1000	Address Register(HPIA)
0800	Data Register(HPID) with autoincrement
1800	Data Register without autoincrement

6.2 HPI Interface Initialization

The following initialization steps are needed to proper working of HPI.

- Setting the **HPI CSR memory or I/O base address register** to provide access to the HPI control/status registers in the PCI2040 and **the control space base address register** to map the HPI bus into host memory space. We take values set in by BIOS for these Base Address Registers (BARs). BIOS automatically allots memory space to each PCI device so we consider it better to use the memory allocated by BIOS.
- The **command register** then needs to be programmed to allow the PCI2040 to claim memory and I/O cycles. Typically we set memory enable and make I/O enable to zero.
- Setting the **HPI data width** and **implementation registers:** These two registers must be programmed either by the serial ROM or software so the PCI2040 knows which DSPs are available and what type of HPI interface to use (8-bit or 16-bit).
- Software then programs the **HPI reset register** to de-assert reset on the HPI bus.

6.3 Setting DSP memory Address and Data R/W Procedure

After this initialization procedure, memory ‘read’ and ‘write’ operations can be done. But before accessing DSP data, the host must first initialize HPIC, in particular BOB (bit 0 and

bit 8), and then the HPIA register. The initialization must occur in this order because the state of BOB affects the HPIA register access.

On devices with extended on-chip RAM, the host should also initialize the XHPIA bit of the HPIC before accessing the HPIA register. The XHPIA bit can be initialized in the same write access to the HPIC that initializes BOB. By writing a one to the XHPIA bit, the host gains access to the seven extended HPI addresses. The host then writes the HPIA register with the seven LSBs designating the value of the extended addresses (HPIA 16:22). When initializing the extended HPI addresses, the same value should be written for the first and second bytes of the access. After initializing the extended addresses, the host must perform another access to the HPIC, writing a zero in the XHPIA bit to regain access to the lower sixteen address bits in the HPIA register. Hence we must follow the following sequence to set a complete address. We are using C5410 DSP which has extended memory space.

Steps to set correct address:

- **Setting the extended HPI address**: This involves first writing 0x1010 to control register and then writing 0x0000 to address register if we don't want to access extended space else writing 0x0001 if we want to access extended space.
- **Setting the DSP memory address we want to access**: This involves first writing 0x0101 to control register (we are sending LSB first so BOB bit =1) and then writing the appropriate address in the address space.

Now we are prepared to write data to the data register. This data will automatically go to the corresponding memory of the DSP. We present here two examples with timing diagrams to explain this Data Write.

Examples of Transactions Targeting the C54X

The control space base address (PCI offset 14h) contains FFEF0000h.

PCI Word Write

In the first example depicted in Figure 6, a PCI write transaction with address FFEF1800, byte enables of 1100b, and a single data phase of the PCI bus occurs. The data is DDCCBBAAh. The PCI2040 takes this PCI transaction and translates it to an 8-bit host port transaction. The event flow is as follows:

1. The host port is idle.
2. HCNTL0 and HCNTL1 are driven high indicating to the C5410 that this transaction is going to target the HPID without auto-increment enabled. The HR/W is driven low indicating to the C5410 that this transaction is a write.
3. HCS0 is asserted indicating that this transaction is targeting DSP0. The first byte or half-word is driven onto the HAD bus. The upper eight data lines (HAD15–HAD8) are not used. Only the lower eight data lines are used when communicating with the C5410. Also, during clock 3, the HDS is asserted. During this time, the C5410 latches the values of HCNTL1, HCNTL0, HWIL, and HR/W.
4. The PCI2040 samples the state of HRDY5X0. If the C5410 indicates it is not ready, then the PCI2040 waits until the C5410 indicates it is ready before it de-asserts HDS and HWIL.

5. Because the state of the HRDY5X0 signal indicates the C5410 is ready, the PCI2040 de-asserts HDS. The C5410 latches the data, AAh, on the rising edge of HDS. The HWIL is driven high.
6. During clock 6, the PCI2040 starts driving the second byte or half word onto the HAD bus. Please note that the PCI bus uses little endian notation. For this reason, the PCI2040 transfers the least significant byte first followed by the next least significant byte.
7. Same as Step 4.
8. Same as Step 5 except the data latched is BBh and the HCS0 is de-asserted indicating the end of the transaction.

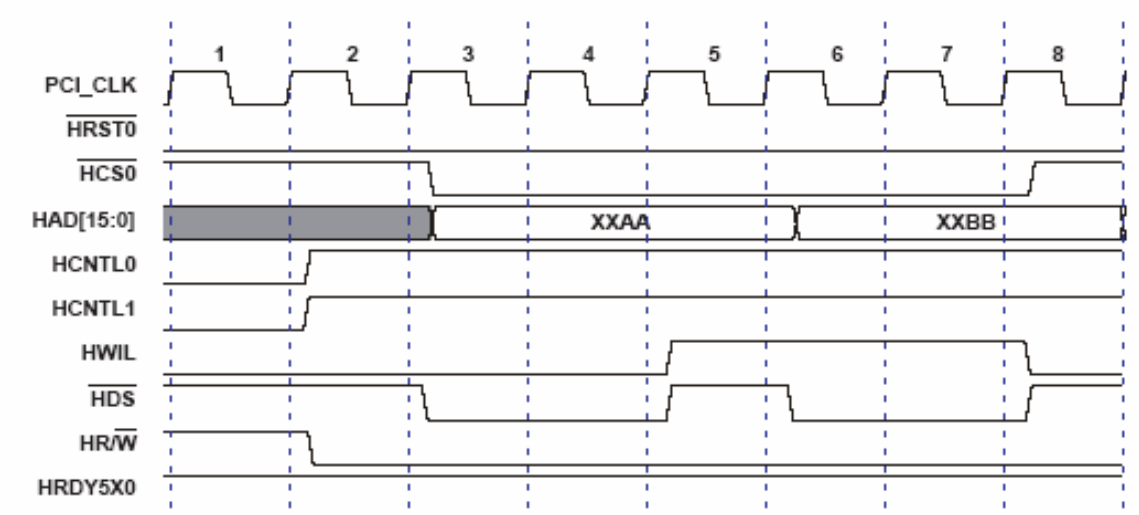


Figure 6. Word Write To HPID without Auto-Increment Enabled

PCI Word Read

The second example outlined in Figure 7 shows how the PCI2040 translates a word read on the PCI bus with a PCI address of FFEF5800h. The event flow is as follows:
1. The host port is idle.
2. HCNTL0 and HCNTL1 are driven high indicating to the C5410 that this transaction is going to target the HPID without auto-increment enabled. The HR/W is driven high indicating to the C5410 that this transaction is a read.
3. HCS2 is asserted indicating that this transaction is targeting DSP0. The first byte or half-word is driven onto the HAD bus. Also during clock 3 the HDS is asserted. During this time, the C5410 latches the values of HCNTL1, HCNTL0, HWIL, and HR/W.
4. The PCI2040 samples the state of HRDY5X0. If the C5410 indicates it is not ready, then the PCI2040 waits until the C5410 indicates it is ready before it de-asserts HDS and HWIL. In this case, the C5410 is not ready.
5. Same as Step 4 but in this case the C5410 is ready.
6. The PCI2040 drives both HDS and HWIL high. The PCI2040 also latches the data on the lower eight data lines (HAD7–HAD0).
7. Same as Step 3.
8. Same as Step 5.
9. Same as Step 6 except the data latched is BBh and HCS2 is de-asserted indicating the end of the transaction. The PCI2040 then places XXXXBBAAh on the PCI bus.

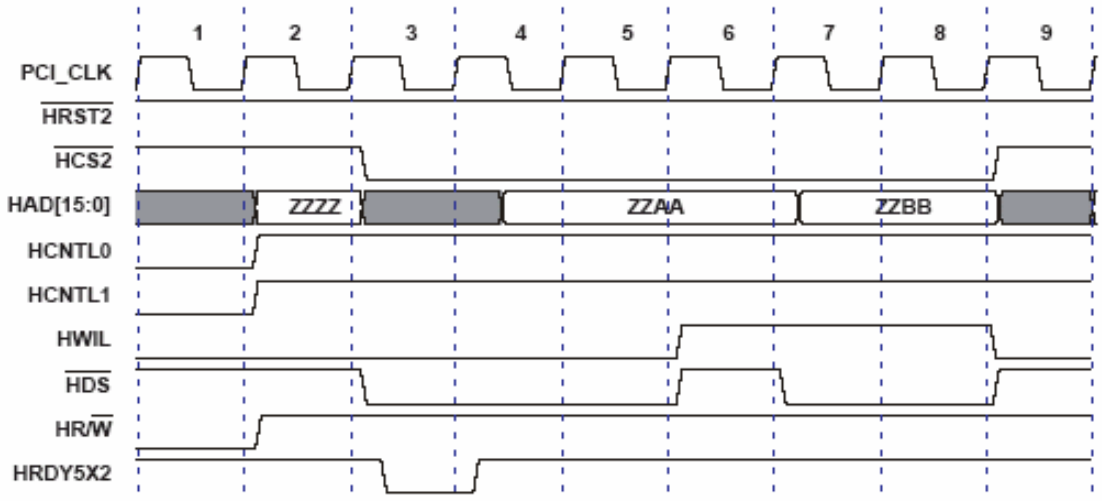


Figure 7. Word Read from HPID without Auto-Increment Enabled

7. Results

We successfully achieved the objectives of the project and established two-way communication between the PC and DSP. We could read and write data at a 1.5MB/s from host. Figure 11 shows the HWIL signal on a CRO during a word write depicting the same. During the data transfer at the above speed we are also able to execute applications on the DSP.

8. Appendix A

Figure 8. The PCI 2040 interface circuitry on a PCB

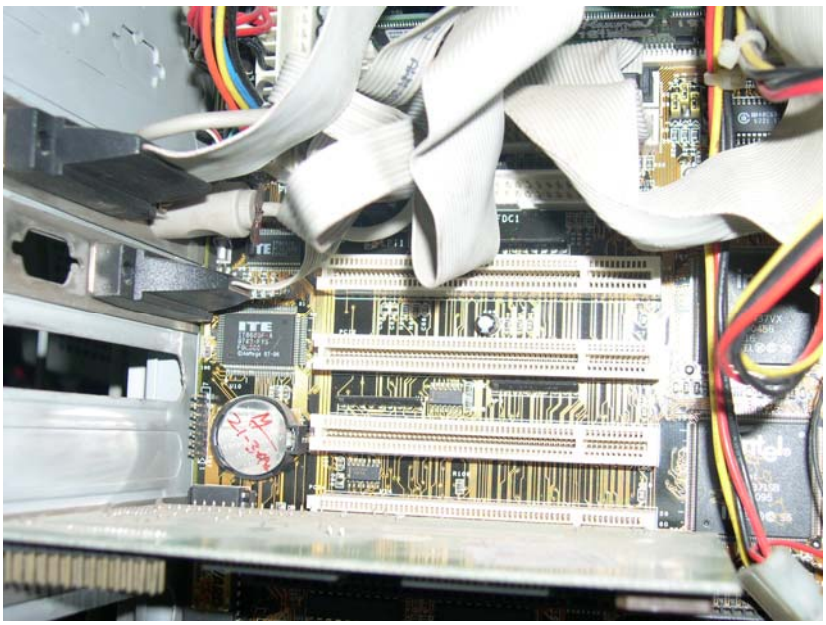


Figure 9. Picture depicting how the hardware sits inside a PC

Figure 10. PCI – DSP Board Connection

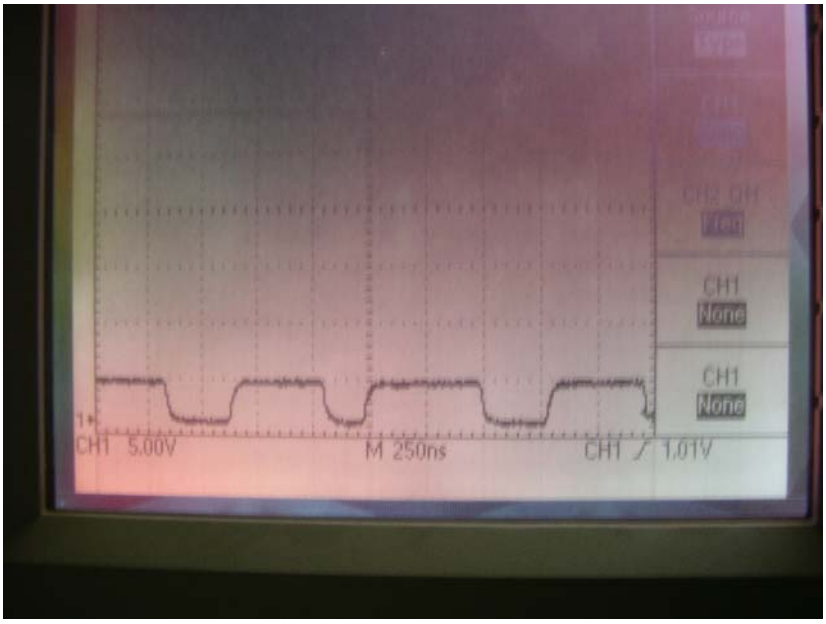


Figure 11. HWIL signal on a CRO depicting data rate of 1.5 MB/s

References

1. Technical Reference, TMS320C54XX Evaluation Module 2001 DSP Development Systems.
2. Data Manual, PCI 2040.
3. PCI Local Bus Specification Rev 2.2
4. Datasheet, TPS7333Q low dropout voltage regulator.
5. Data Manual, TMS320VC5410 Fixed Point Digital Signal Processor.