PCI Bus Applications on FPGAs

Introduction

The Peripheral Component Interconnect (PCI) bus is a high-bandwidth, "plug-and-play" bus designed to meet the performance demands of the peripherals of today's high-performance PCs and workstations and their large bandwidth applications. It is rapidly becoming widely accepted in the computer industry as it opens doors to performance demanding applications such as video and audio systems, graphics accelerator boards, 3D native signal processing, network adapters, data acquisition, and data storage devices. Development of PCI products requires strict adherence to the PCI Local Bus Specification.

Continuous evolution of the PCI specification and specific needs of each application demand a flexible PCI solution. This makes programmable logic in general and FPGAs in particular ideal candidates for the PCI interface. Designing a PCI interface can take several man-months. It is the intention of this application note to provide an overview of the PCI bus and its associated transactions, and to present an example design for a PCI target device that has been implemented in a Cypress FPGA. This note covers the basics of PCI, an example PCI target design, and design issues a PCI designer will encounter. The PCI design files may be obtained by contacting the Applications Group at (408) 943−2456.

PCI Bus

The PCI spec 2.1 specifies the PCI operating speed to be 0 to 33 MHz with 32-bit synchronous bus, expandable to 64 bits. 66-MHz PCI bus speed is also specified to allow future migration. The PCI has a potential transfer rate of 132 MB/s. This value will double/quadruple when the bus is expanded to 64 bits or/and the speed is increased to 66 MHz. PCI is also specified at both 5-volt and 3.3-volt operations, and is processor independent. All PCI devices have a "configuration space" that enables PCI to be a "plug-and-play" solution. Configuration of add-in boards and components is done automatically through software.

PCI Architecture

The PCI bus is the backbone of the I/O and memory devices of the computer (see *Figure 1*). Processor independent, the PCI bus is accessed by the CPU via a CPU local bus to PCI bridge device. The I/O and memory devices hang off the PCI bus and transact in an initiator/target (master/slave) relationship.

PCI Interface Signals

A PCI interface device must have 47 pins. A PCI initiator device has 2 additional pins, which brings the total number of required pins to 49. Optional pins provide 64-bit operation, JTAG boundary scan, target locking, cache support, and interrupt expandability to the PCI bus (see *Figure 2*).

There are five different types of PCI signals:

in	input only signal
out	output only signal
t.s.	bidirectional, three-state input/output pin
s.t.s.	sustained three-state signal; an active LOW signal driven by one agent at a time and must be precharged HIGH before floating. A pull-up resistor is provided by the central resource to sustain the signal in the HIGH state.
o.d.	open drain signal so multiple devices share this signal as a wired-OR

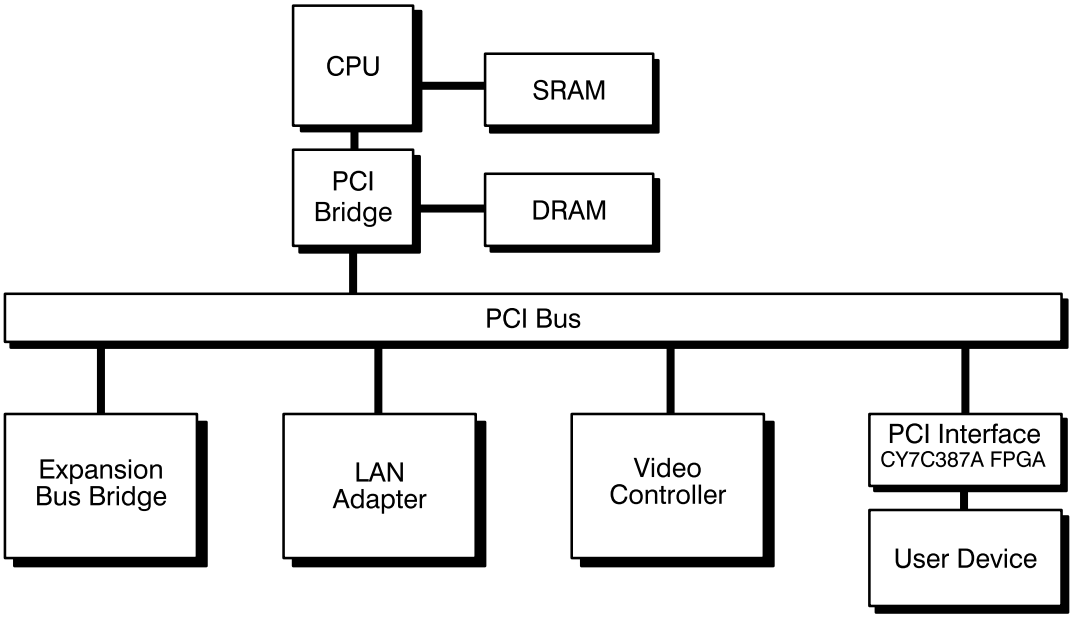

Figure 1. PCI Architecture

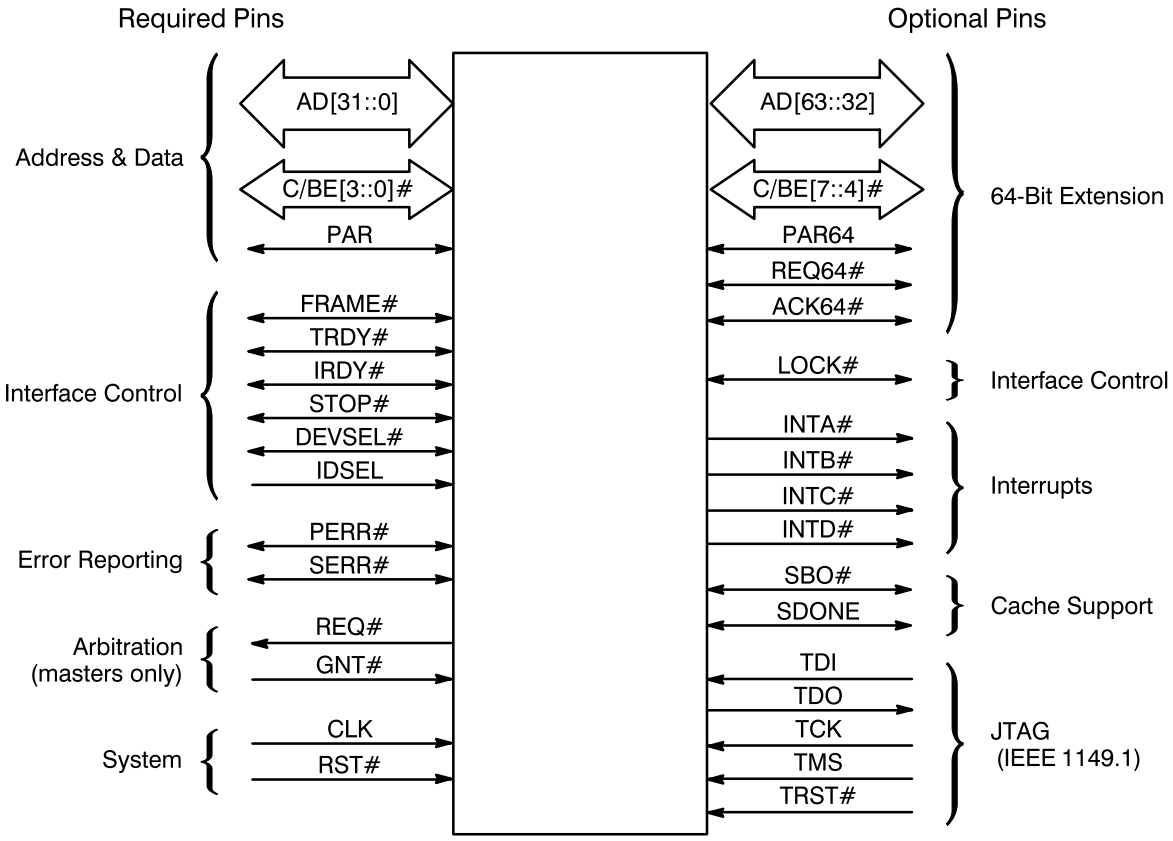

Figure 2. Pin Diagram

Table 1. Required Pins

Pin Name	Type	Description
AD[31:00]	t.s.	32-bit bidirectional multiplexed address/data bus
C/BE[3:0]#	t.s.	Byte enables for the four bytes of the 32-bit AD line
PAR	t.s.	Parity bit for even parity over AD and C/BE lines
FRAME#	s.t.s.	Indicates the duration of a transaction
TRDY#	s.t.s.	Target ready signal, indicates that the target is ready to perform a data transfer
IRDY#	s.t.s.	Initiator ready signal, indicates that the initiator is ready to perform a data transfer
STOP#	s.t.s.	Target signal to induce retry, disconnect, or abort
DEVSEL#	s.t.s.	Target signal to claim the current transaction on the bus
IDSEL	in	Individual device selector signal
PERR#	s.t.s	Parity error during the data phase
SERR#	o.d.	Parity error during the address phase or special cycle
REQ#	t.s.	Initiator bus request arbitration signal
GNT#	t.s.	PCI bus arbiter grant signal to requesting initiator
CLK	in	PCI system clock
RST#	in	PCI system reset signal

Table 2. Optional Pins

Pin Name	Type	Description
AD[63:32]	t.s.	64-bit address/data extension pins
C/BE[7:4]#	t.s.	64-bit byte enable extension pins
PAR64	t.s.	64-bit parity bit
REQ64#	t.s.	Initiator 64-bit bus request arbitration signal
ACK64#	t.s.	PCI bus arbiter 64-bit grant signal to requesting initiator
LOCK#	s.t.s.	Target locking signal
INTA−D#	o.d.	Interrupt pins

Note:

PCI also supports two optional pins for cache support and five optional pins for JTAG support.

PCI Bus Commands

PCI initiators begin a transaction by placing a command on the bus. This command defines what action will be performed during the current transaction. *Table 3* shows all PCI bus commands and their 4-bit values.

Table 3. PCI Bus Commands

C/BE[3:0]#	Command
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

PCI Configuration Space

The configuration space, a required feature of all PCI devices, is what makes PCI a plug-and-play solution. During system configuration, the PCI bus is scanned to determine the configuration requirements for all agents on the bus. All PCI devices must implement 256 bytes of configuration space which holds configuration information such as device identification, device status, functionality enables, and base address registers for address space assignments.

Configuration Space Header

The first 64 bytes of the 256 byte configuration space are known as the configuration header. This application note describes the header currently used for most I/O and memory devices, the type 00 header (shown in *Figure 2*).

Device ID – Device identification number issued by the vendor.

Vendor ID – Vendor identification number issued by the PCI SIG.

Revision ID – Device-specific revision identification number issued by the vendor.

Header Type – Identifies the layout of the second part of the predefined 64-byte header.

Class Code – Identifies the generic function of the device.

Base Address Register – Register for address space location assignment.

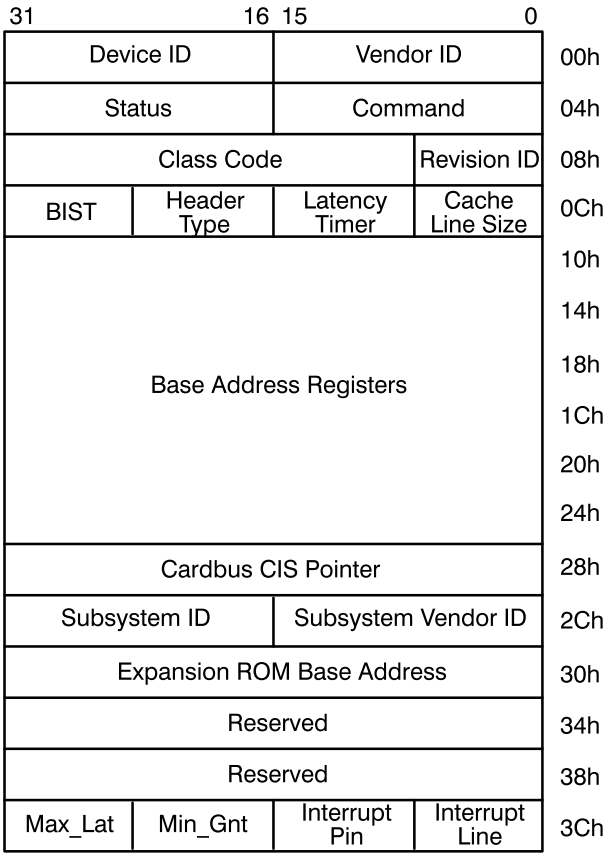

Figure 3. Type 00h Configuration Space Header

The latter 192 bytes of the 256 bytes of configuration space are a device-dependent region. A PCI-compliant device does not have to implement unused portions of the configuration space as registers. However, a value of zero must be returned when unused locations are read.

The 32-bit-wide register lines in the configuration space are addressed on 32-bit word boundaries. Hence the register sequence (see the right side of *Figure 2*) is 00h, 04h, 08h, 0Ch, etc. The 32-bit registers comprise four bytes, each of which may be accessed when the corresponding Byte Enable is asserted.

Address Space

A PCI device's address space is relocatable. The system assigns areas of address space by writing address values to the device's Base Address registers. The amount of address space a device needs is also determined by examination of the Base Address registers within the configuration space of that device. To determine how much address space a device on the bus requires, the system writes the value xFFFFFFFF to the Base register, and then reads the register. The number of zeroes returned in the least significant position determines how much address space the device requires. For example, if a device returns the value xFFFFFF80, the arbiter knows that this device requires 128 bytes (7 zero bits, 2 ^ 7 = 128).

The zeroes in the least significant positions of the Base Address registers should be implemented as hard-wired zeroes. More hard-wired zeroes provide a larger amount of address space with a smaller number of bits to compare to determine an address hit. In contrast, less hard-wired zeroes translate to a smaller amount of address space for the device, but more bits to compare to determine an address hit. For some devices, the number of bits to compare to determine an address hit affects how fast a PCI device can claim a transaction as a target.

Transaction Waveforms

All PCI read/write transactions are inherently burst transfers. The length of the burst is determined by the FRAME# signal provided by the initiator (master) of the transaction. Transactions begin with a single address phase followed by one or more data phases.

Figure 4 shows a basic read operation. Prior to clock 1, the initiator is assumed to have arbitrated for control of the bus, and has received permission to use the bus for a transaction. After clock 1, the initiator places the address of the desired device and the command for the device on the bus while asserting the FRAME# signal. On clock 2, because FRAME# is sampled LOW for the first time, all devices on the PCI bus are required to latch in the address and command on the bus, and begin decoding the address to determine transaction ownership. After clock 2, the initiator (master) waits for a target (slave) device to respond and claim the transaction by asserting the DEVSEL# signal.

Because this is a read transaction, the target is required to wait a clock to induce a turn around cycle on the A/D bus to prevent contention of the bus as control switches from initiator to target.

Data transactions are controlled by three signals: FRAME#, IRDY#, and TRDY# signals. (See signal description above for definition of signals.) The actual transfer of data occurs only on clocks where both IRDY# and TRDY# signals are asserted. If either or both signals are not asserted, then a wait state occurs.

After clock 3, the target is ready to provide the first piece of data, and the initiator is ready to receive it. Both the IRDY# and TRDY# signals are asserted, and on clock 4 a data transfer takes place. Also on this clock, because the target senses that the FRAME# signal is still asserted, it knows that the transaction is not complete and the initiator expects more data. On the next clock (clock 5), the target is not ready (TRDY# deasserted), and so a wait state is induced. On clock 6, a data transfer occurs since both ready signals are asserted. On clock 7 the initiator is not ready, so the IRDY# signal is deasserted, inducing a wait state. The initiator knows that it desires only one more piece of data, and when the IRDY# signal is asserted on clock 8, the FRAME# signal is deasserted. A data transfer takes place on this clock since the TRDY# signal is also asserted. Also on clock 8, the target samples the FRAME# signal. Since the FRAME# signal is deasserted, the

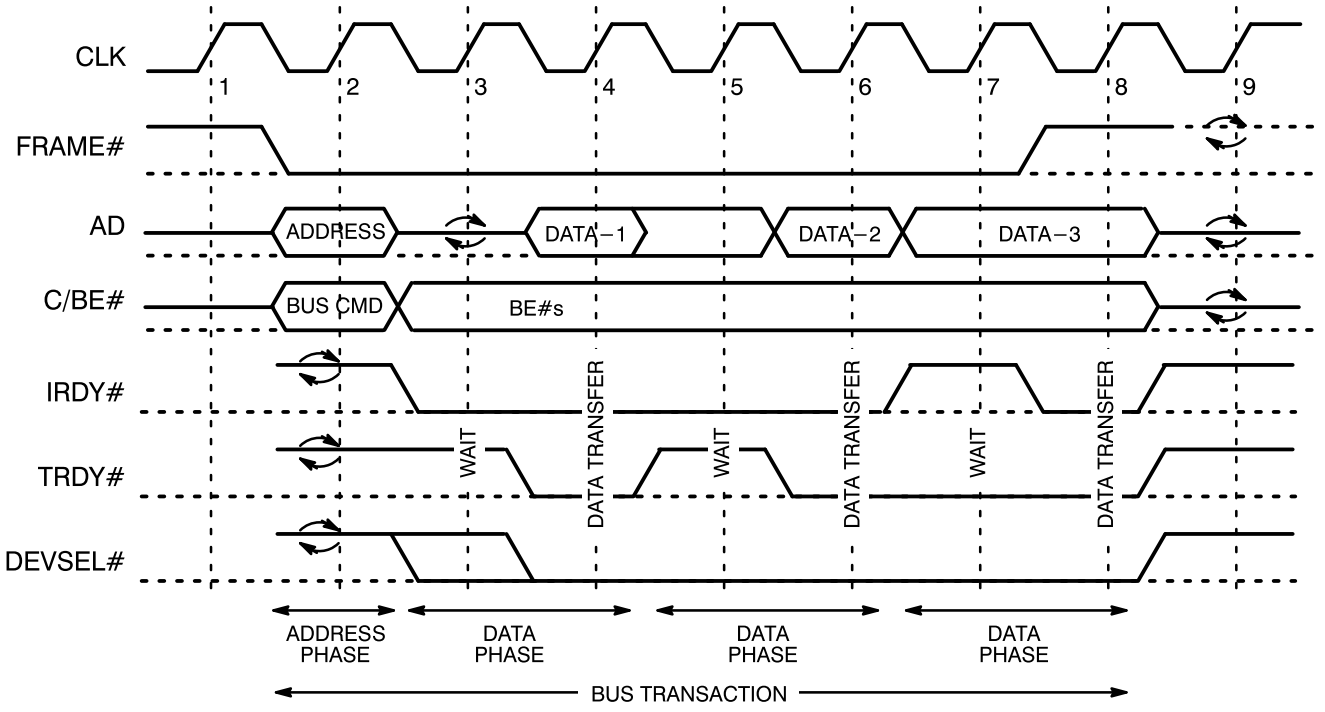

Figure 4. Read Transaction Waveform

target device is informed that the transaction is over. On clock 9, FRAME#, A/D bus, and C/BE bus are turned around for one cycle, and the control signals are precharged HIGH before being three-stated. This completes the read operation.

Once a ready signal is asserted, it may not be deasserted until the data transfer takes place or the transaction is aborted.

Figure 5 shows a basic write operation. The rules of transaction are exactly the same as the read operation, with the exception that a turn around cycle on the A/D bus right after the address phase is unnecessary since the same agent controls the bus for the entire duration of the transaction.

Claiming the Transaction

Not all PCI devices can capture the address, decode it, and claim the transaction within a single clock after an initiator begins a transaction. PCI targets may take up to 3 clocks after the initial address phase to assert the DEVSEL signal (see *Figure 6*). In *Figure 6*, the address phase occurs at clock 2. If a target can assert its DEVSEL# signal by clock 3, it is considered a "fast" response device. Assertion of DEVSEL# on clock 4 would be "medium" and clock 5 would be "slow." The sixth clock is reserved for subtractive decoding devices. If a target device has not asserted DEVSEL# by the sixth clock, the initiator may terminate the transaction.

Parity

Parity generation is required for all PCI devices. In general, parity checking is usually required. On read transactions, it is the responsibility of the target to generate parity. On write transactions, parity generation is the responsibility of the initiator.

Parity in PCI is even parity over the AD bus, C/BE bus, and the parity line. The generated parity bit is available one clock after valid values on the buses are transferred. A parity error is reported two clock cycles after the valid values have been transferred (i.e., one clock after the parity bit was available). Because parity is calculated over the entire AD bus, the signals on the AD bus must be held stable even if they are undefined.

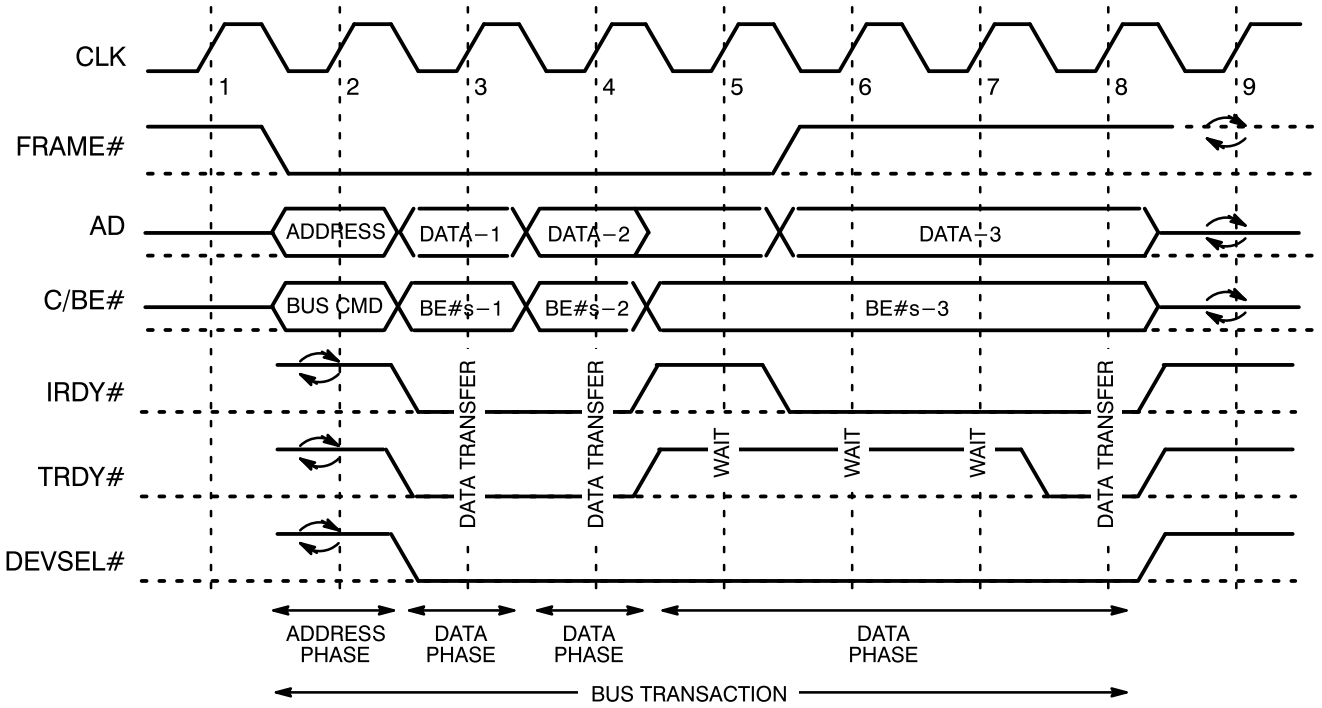

Figure 5. Write Transaction Waveform

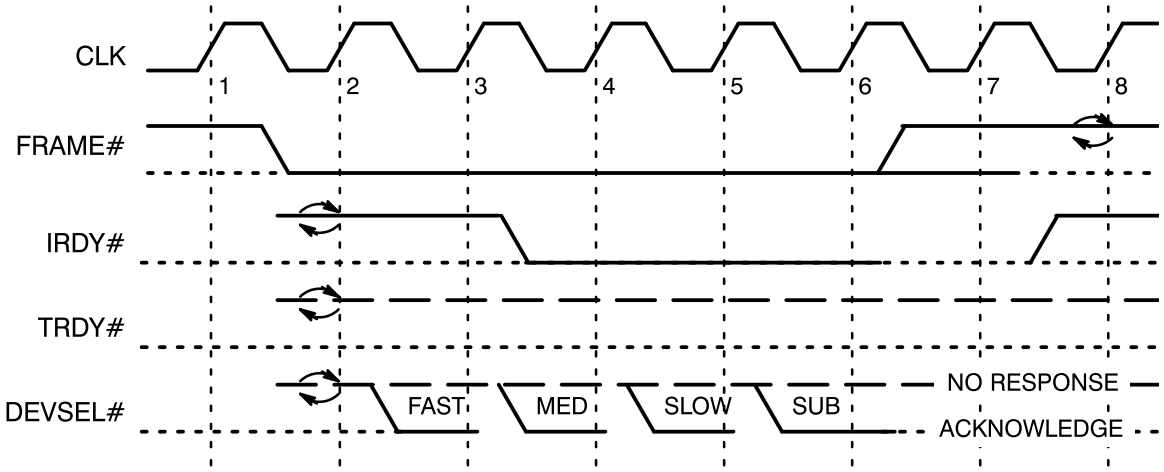

Figure 6. Transaction Claiming Speed

Aborting the Transactions

PCI provides a method for premature transaction termination.

There are three scenarios when an initiator may terminate a transaction.

1. The transaction has completed normally, and so the initiator ends the transaction.
2. The initiator's latency timer has expired and arbitrator has deasserted the initiator's GNT# signal. The initiator is allowed one last data transfer once the latency time-out is sensed.
3. No target has responded to an initiator request within five clock cycles after FRAME# was asserted. The initiator will end the transaction in the sixth clock.

There are three types of target terminations:

1. disconnect – When a data phase is very long, the target may induce a disconnect to free the bus. To signal a disconnect, the target must assert both STOP# and TRDY#. One last data transfer takes place, and the initiator ends the transaction.

2. retry – If a target cannot respond to the current transaction at the current time, the target may signal a retry, indicating to the initiator to try the transaction again at a later time. For example, if a target is currently locked for exclusive access by another initiator, then the target would signal a retry. In a retry, no data is transferred. A target can signal a retry by asserting the STOP# signal and keeping the TRDY# signal deasserted.

3. target-abort – If a target encounters a fatal error, then the device may signal an abort. The abort is signalled by asserting STOP# and deasserting DEVSEL#. The TRDY# signal must also be kept deasserted.

Recommended Device Pinout

The PCI spec recommends the pinout shown in *Figure 7*.

A PCI Target Application

To introduce designing for PCI applications, a target PCI implementation is presented. This example design can be modified to suit any specific needs.

Design Overview

1) The Features

The first step is to decide what features the PCI Target interface is going to have. A PCI-compliant interface with the following features is desired.

- 0 to 33 MHz bus clock speed operation
- 32-bit Addr/Data bus
- Burst cycles
- Wait state support
- Fully customizable address space size: 1 byte to 4 Gbytes
- Two base address registers (more may be implemented if necessary)
- Configuration, I/O, and Memory read and writes
- Parity generation, with checking option
- Target-abort and retry support

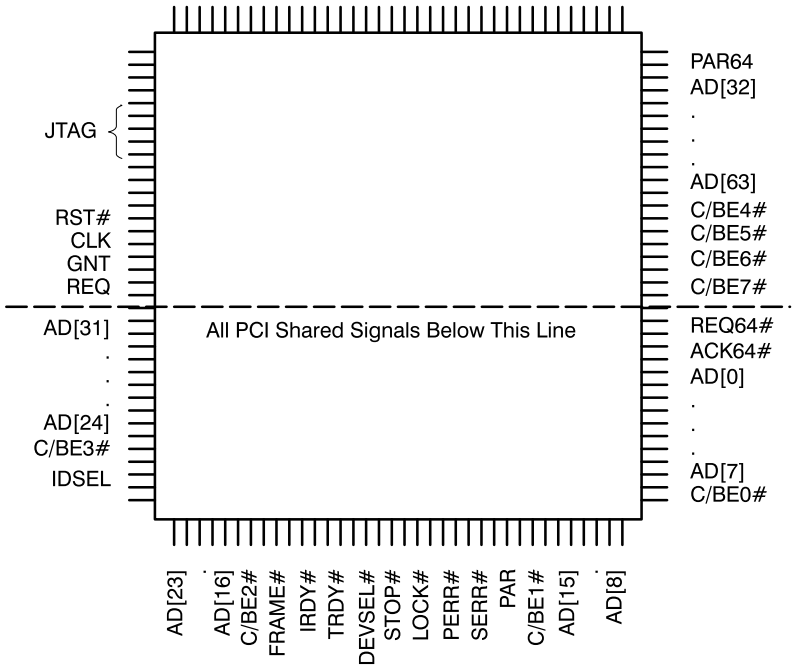

Figure 7. Recommended Pinout

- State machines and configuration space implemented in VHDL for easy high-level user modification
- Generic back-end user interface

2) Handling the Address Phase

The target needs to latch the data on the first cycle that the FRAME# signal is sampled LOW. The PCI specification allows both the address and FRAME# signals only a 7-ns set-up time. In some cases, the logic necessary to determine that FRAME# has transitioned to the asserted state and then enable all 36 bits to the register would take longer than 7 ns. To make it more robust, it was decided to put an input register on the A/D bus that would latch data every clock tick. In parallel, the asserted FRAME# signal would "wake" the PCI state machine. In this manner, the device will have the address stable for an entire clock cycle. A second register is needed to memorize the address, command, and IDSEL lines.

Address compare logic is needed to compare the latched address with all implemented base address registers from the configuration space. This means that the base address registers have to be directly connected to the inputs of the compare logic. For flexibility, the address compare logic is pipelined. In the event that the address from the PCI matches a base address register, a hit signal is asserted.

An asserted hit signal, or an asserted IDSEL signal for configuration transactions, will cause the control logic to claim the transaction by asserting the DEVSEL# signal. Concurrent to the address compares, the command will be decoded.

3) Handling the Data Phases

When PCI performs a write operation on the device, it is undesirable for the PCI bus to have to wait for the back-end user device to be ready to accept the data. Therefore, a FIFO-like structure is needed to reduce latency. The size of the FIFO structure should be customizable without affecting the rest of the design's logic. For this design, a single 36-bit register is used.

To handle burst cycles, an address counter for the back-end user device must be included. This counter is stepped on every data transfer. Since this design performs 32-bit transfers, addressing must be on double word aligned boundaries.

For read operations, parity must be generated and made available one clock after the data transfer takes place.

4) The Control Logic

The control logic must be able to handle the PCI protocols, support burst transfers, wait states, and still meet the 2- to 11-ns clock-to-out time, and 7-ns set-up time of PCI bus signals. It should also provide all the internal control signals and user interface signals. Because of its high-level nature, the control logic should be implemented in VHDL. To meet clock-to-out times, output should be registered.

5) The Configuration Space

Many designs will only use 00h to 0Bh configuration registers and Base Address registers. The configuration space implementation should have these registers and the mechanism for writing to and reading from those registers. In addition, the register information is used internally by direct means (as opposed to using a read operation) so the contents of the registers need to be accessible by the rest of the design. For example, the Base Address registers need to be connected directly to the inputs of the address comparator logic. Since customizing the design for real applications will involve modifications to the configuration space, the configuration space registers is implemented in VHDL.

Block Diagrams and Data Paths

Figure 8 shows the top-level block diagram of a PCI target interface developed using the criteria of the previous section.

CONTROL: (VHDL) This block contains the PCI and user state machines and the logic that determines the internal control signals as well as the bus signals.

C_SPACE: (VHDL) This block is the VHDL implementation of the configuration space. "Hardwired" values are easily set in the VHDL source code and registers are manipulated behaviorally.

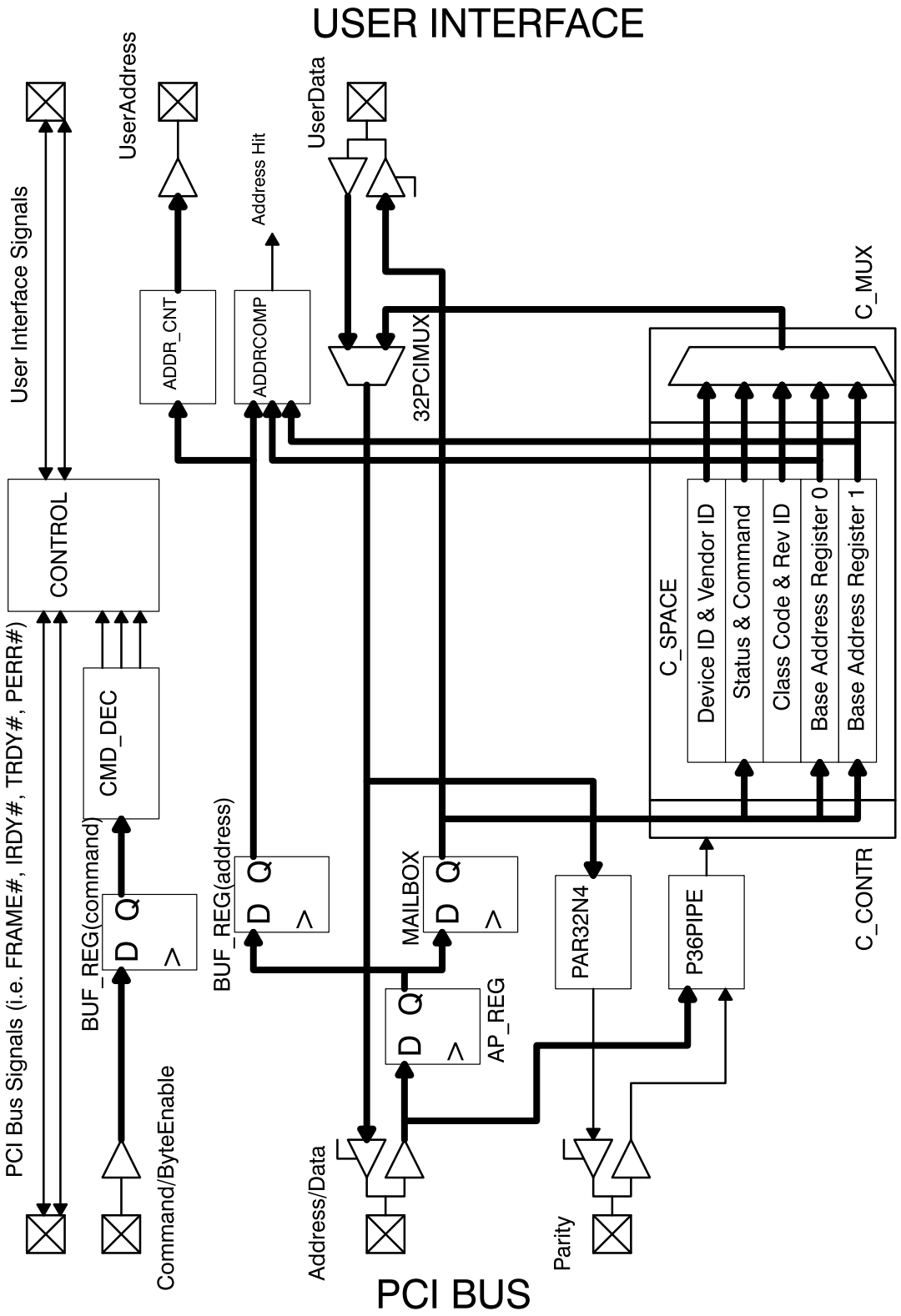

Figure 8. Target Design Block Diagram

AP_REG: (Schematic) This block acts as the input register for the A/D bus, C/BE bus, and the IDSEL signal. On every clock, the values on these signal lines are registered into this block.

BUF_REG: (Schematic) This is the storage register for the address and the command taken from the PCI bus at the beginning of each transaction. It is enabled by the CONTROL block and cleared upon reset or at the close of a transaction.

CMD_DEC: (Schematic) The command is decoded into single-bit enable lines. The decoded command, the I/O and Memory access enable, and IDSEL line determine which function signal will be raised. All unimplemented memory transactions are treated as either the respective mem read or write.

ADDRCOMP: (Schematic) This pipelined address comparator takes two cycles to determine an address hit. The number of bits compared can range from 1 to 32 bits. If less than 16 bits need to be compared, the pipelined configuration usually is not necessary since a hit can be determined within one clock cycle.

MAILBOX: (Schematic) This is a data-holding register for I/O or Memory writes. This block may be changed to a multileveled FIFO to reduce latency between burst transfers.

PAR32N4: (Schematic) This block calculates even parity over the 32-bit A/D bus and the 4-bit C/BE bus in one clock cycle. The output is registered to delay the valid parity bit one clock in accordance with the PCI spec.

ADDR_CNT: (Schematic) The initiator provides the beginning address for a transfer. On burst transfers, the target device is responsible for stepping the beginning address appropriately for its local user side. This block counts the address on data transfers.

C_CONTR: (Schematic) This block decodes the address and enables the addressed register within the configuration space.

C_MUX: (Schematic) This is a 32-bit, 4-to-1 mux, exclusively selecting configuration registers 00h, 04h, 08h, and 10h. If other configuration registers need to be addressed, a larger mux must be used. The 4-to-1 mux was chosen because it fits in a single level of logic cells. When nonimplemented registers are addressed, the block outputs zeroes.

32PCIMUX: (Schematic) This is a 32-bit, 2-to-1 mux, selecting between the local user data bus and the configuration space register output mux C_MUX.

State Machines

Within the CONTROL block, there are two state machines: the PCI state machine (*Figure 9*), which handles PCI bus protocols, and the User state machine, which handles transactions on the user interface (*Figure 10*).

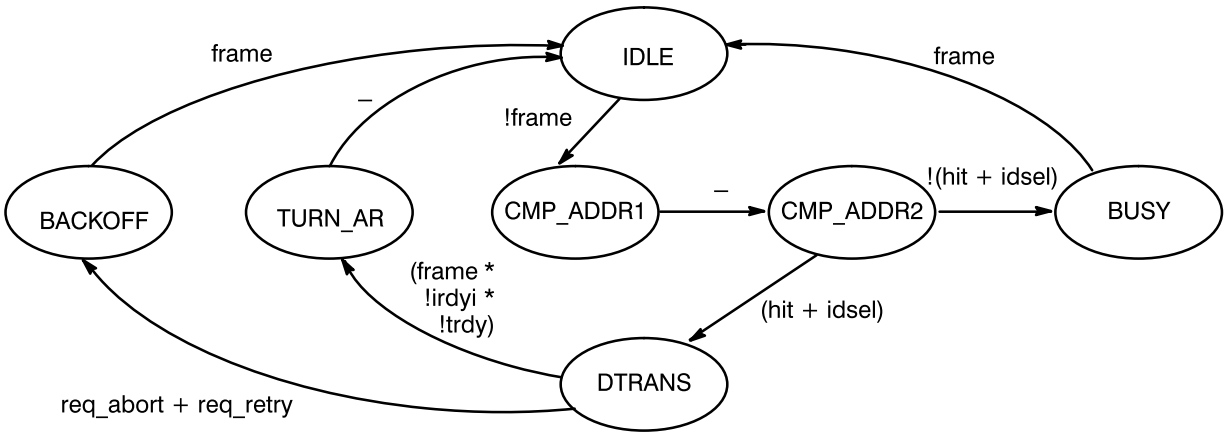

Figure 9. PCI Interface State Machine

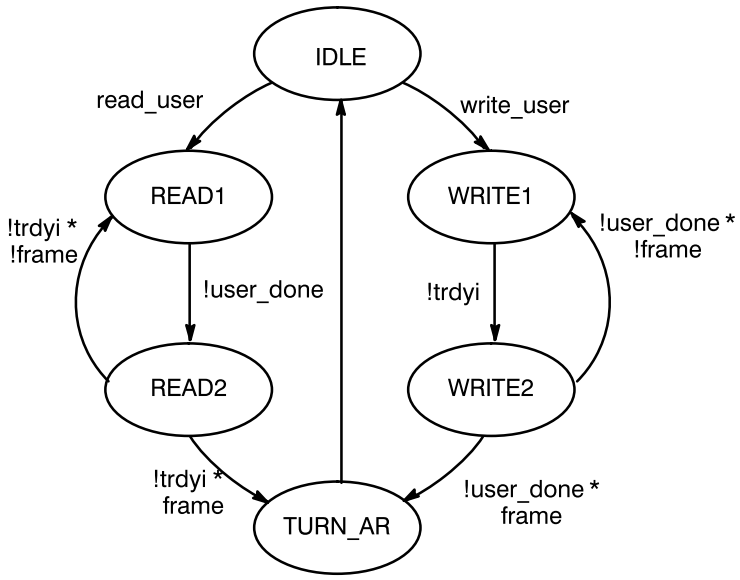

Figure 10. User Interface State Machine

PCI State Machine

IDLE: The device waits in this state while the PCI bus is idle. When the FRAME# signal indicates a transaction is beginning (becomes asserted), the FSM moves to the CMP_ADDR1 state.

CMP_ADDR1: This is the first stage of the address and command decoding pipeline. On the next clock, the FSM moves to the CMP_ADDR2 state.

CMP_ADDR2: This is the second stage of the decoding pipeline. At this point, the address hit and command are determined. If it is an address hit or a configuration operation is occurring on that device, then the FSM moves to the DTRANS state. Otherwise, it goes to the BUSY state.

BUSY: In this state, the PCI bus is engaged in a transaction that the device is not a part of. The device will wait in this state until the bus goes idle again.

DTRANS: All data transfers occur in this state. When the device determines that it is involved with the last data transfer of the transaction, the FSM will move to the TURN_AR state on the next clock. If an abort is sensed, then the FSM will move to the BACKOFF state.

TURN_AR: Signals on the PCI bus are precharged and three-stated, and the A/D bus is brought to high impedance. The FSM moves to the IDLE state on the next clock.

BACKOFF: The device induces a target abort or a retry in this state. When the transaction is closed, the FSM moves to the IDLE state.

User State Machine

IDLE: The user FSM stays in this state while the user interface is inactive. When the device is involved with either a read or a write transaction involving the user interface, the FSM moves to the READ1 or WRITE1 state appropriately.

READ1: In this state, the user interface prompts the user device for the requested piece of data. When the user device responds with valid data, the FSM moves to the READ2 state.

READ2: The device has the requested data ready, and waits for the initiator to pick it up. Once the transfer takes place, the FSM either moves to the READ1 state for burst transfers, or to the TURN_AR state.

WRITE1: In this state, the device receives the data from the initiator. Once the data transfer takes place, the FSM moves to the WRITE2 state.

WRITE2: Data is available in the data FIFO. The user device is prompted for a data write. When the

user device signals that the transfer is completed, the FSM moves back to the WRITE1 state for burst transactions, otherwise it moves to the TURN_AR state.

TURN_AR: This is the final state of the user FSM before going back to IDLE.

Design Interaction Description

To demonstrate the operation of this PCI target design, a description of the waveforms are analyzed.

Scenario 1: Configuration Write

[These scenario descriptions follow the simulation waveforms produced in ViewSim. Simulate design using command file PCICR.CMD with the PCI target design, 75 ns.]

At the beginning of the transaction (clock 0), this target device senses on the clock that FRAME# has been asserted. Because the AP_REG captures all information on the A/D and C/BE buses and the IDSEL line on every clock, the target knows that the address is held within the AP_REG.

On clock 1, the BUF_REG is enabled so that the address and command can be stored. Between this clock and the next, the IDSEL line is found to be asserted and the CONTROL logic determines that DEVSEL should be asserted. The command is also decoded to be a configuration write operation, and the bits [7:2] of the address are decoded by C_CONTR to enable the appropriate C_SPACE register.

On clock 2, the internal DEVSEL signal is captured by the DEVSEL output register to meet the 2- to 11-ns clock-to-out timing spec. The target samples IRDY# asserted, and knows that valid data is on the bus. The CONTROL block asserts the internal TRDY# signal.

On clock 3, the initiator samples the DEVSEL signal asserted and knows that the transaction has been claimed. The TRDY# output register asserts the PCI bus TRDY# signal. Valid data is contained in the AP_REG.

On clock 4, the data from the AP_REG is written to the C_SPACE register according to the byte enables also held within the AP_REG. The CONTROL logic samples FRAME# deasserted and knows that this is the last transaction. Both the TRDY# and DEVSEL# signals are deasserted.

On clock 5, the CONTROL logic three-states the bus signals, and resets the address compare and BUF_REG blocks. The transaction is complete.

Scenario 2: Configuration Read

[In ViewSim, use command file PCICR.CMD with the PCI target design, 315 ns.]

This transaction works like the Configuration Write transaction with a few differences:

On clock 1, the A/D bus is floated by the initiator to turn control of the data bus over to the target. When the CONTROL logic asserts DEVSEL, it turns on the output enable to the A/D bus.

The address held in BUF_REG causes C_MUX to select the appropriate configuration register bus. For this design, only 32-bit registers at 00h, 04h, 08h and 10h are selected. Other addresses will cause the C_MUX to randomly select one of the four buses.

The 32PCIMUX selects between configuration and IO/Memory reads. The output of 32PCIMUX goes directly to the output pins of the A/D bus.

Scenario 3: I/O or Memory Write

[In ViewSim, use command file PCIMW.CMD with the PCI target design, 560ns.]

At the beginning of the transaction (clock 0), this target device senses on the clock that FRAME# has been asserted. The AP_REG captures all information on the A/D bus every clock, therefore the target knows that the address is held within the AP_REG.

On clock 1, The BUF_REG is enabled so the address and command can be stored. While this happens, the address compare pipeline begins it's first phase. Between this clock and the next, an address hit is determined and the CONTROL logic determines that DEVSEL should be asserted. The command is also decoded at this time.

On clock 2, the DEVSEL signal is captured by the DEVSEL output register to meet the 2- to 11-ns clock-to-out timing spec. The user address counter

is loaded with the offset address from BUF_REG. The ADDR_CNT is enabled after this clock to load it with the offset address on the next clock.

On clock 3, the initiator samples the DEVSEL signal asserted and knows that the transaction has been claimed. The target waits with the TRDY# signal deasserted until an asserted IRDY# signal. The CONTROL logic senses that the IRDY# signal is asserted, and thus knows that valid data is on the bus. The CONTROL logic then prepares to assert the TRDY# signal on the next clock, and enables the MAILBOX register.

On clock 4, both IRDY# and TRDY# signals are asserted so both agents know that a data transfer took place. The enabled MAILBOX register collects the contents of the AP_REG (previous clock held valid data also since IRDY# was already asserted). The TRDY# signal is immediately deasserted. If the CONTROL logic sample FRAME# to be asserted (indicating a burst transfer), then the target would prepare to perform another data transfer. Otherwise, the CONTROL logic will end the transaction just like the Configuration Write transaction.

On clock 5, the CONTROL logic for the user interface side senses that the MAILBOX contains data to be written to the user device. The USR_WRITE strobe is asserted, and the CONTROL logic waits for the user device to respond with an asserted USER_DONE.

On clock 6, USER_DONE is sampled asserted, and the CONTROL logic 'clears' out the MAILBOX register and increments the ADDR_CNT.

Scenario 4: I/O or Memory Read

[In ViewSim, use command file PCIMR.CMD with the PCI target design, 560 ns.]

This operation works like the I/O or Memory Write with these differences:

On clock 1, the A/D bus is floated by the initiator to turn control of the data bus over to the target. When the CONTROL logic asserts DEVSEL, it turns on the output enable to the A/D bus.

After clock 2, the CONTROL logic prompts the user device with the USR_READ strobe.

On clock 3, the USER_DONE signal is sampled asserted. This is a 'pass-through' read design so the user device must hold the data values so the PCI bus can read them.

On clock 4, the CONTROL logic prepares the TRDY# signal so that a data transfer may take place on the next clock.

On clock 5, both IRDY# and TRDY# signals are asserted. If the CONTROL logic senses that FRAME# is asserted at this point (indicating a burst transfer), then the target prepares to perform another data transfer. Otherwise, it closes the transaction in the normal fashion.

PCI Target Interface Timing Specifications

Table 4. PCI Bus I/O Timing Specification

Symbol	Description	Min.	Max.
t_{val}	Clock to Data Valid	2	11
t_{on}	Float to Active Delay	2	–
t_{off}	Active to Float Delay	–	28
t_{su}	Input Set-Up Time	7	–
t_{h}	Input Hold Time	0	–
t_{cuc}	Clock Cycle Time	30	–
t_{high}	Clock High Time	12	–
t_{low}	Clock Low Time	12	–

Table 5. User Interface I/O Timing Specification

Symbol	Description	Min.	Max.
t_{val}	Clock to Data Valid	2	14
t_{su}	Input Set-Up Time	14	–

Critical PCI Design Issues

There are several considerations that arise when designing PCI applications using an FPGA. For an FPGA to be able to handle the demands of PCI, it must have several necessary characteristics. These

characteristics include: speed, generous routing, a large number of pins, a large amount of logic resources, and many registers. This section will describe some critical issues and possible solutions to implementing PCI applications with an FPGA.

(1) Bused signals set-up time is no greater than 7 ns. (PCI spec 7.6.4.2)

Problem 1: The Address/Data bus needs to be tapped by several blocks: the address decoders for each Base Address, the address registers, the data registers for memory and I/O transactions, and the data bus for the configuration space. This fanout can add considerable loading to the bus, thus inhibiting the input drivers and increasing the input delays beyond the 7-ns set-up time, even with the FPGA's short input delay.

Solution 1: A 36-bit input register can be implemented using D-type flip-flops. These flip-flops will latch whatever is on the Address/Data bus every clock. This increases the availability of the data from 7 ns to 30 ns, with the trade off of adding one clock cycle. This additional clock cycle, however, does not significantly impact the performance of the device for several reasons: during the address phase, addresses must be latched anyway; and during data phases, another PCI spec forces the extra wait state.

Problem 2: Some bused signals such as IRDY# and FRAME# are used combinatorially to determine other outputs. In some cases, the combinatorial delay to the registered outputs and states of the state machine take longer than 7 ns, thus giving an invalid registered output or state.

Solution 2: Remember that there is a clock input delay to the device. The actual set-up time is input delay minus the clock delay. In the event that this difference is still greater than 7 ns, then care should be taken to minimize fanout of the input signal, and to place the logic near the pin. In most cases, this is taken care of automatically by the place and route tool by placing a constraint on the signal. To do this, run SpDE and open up the design .CHP file. Run the path analyzer. Click on options, and display all paths that start from the critical input signals (i.e., IRDY# and FRAME#). Place constraints on the critical signals paths (e.g., type "5.0" ns in the constraint column for all critical paths) and rerun placer tools.

2) Bused signals must be driven valid between 2 and 11 ns after CLK. (PCI spec 7.6.4.2)

Problem: Many delays contribute to a signal's total delay. These delays include: the clock input to flip-flop delay, the clock to Q output delay, the combinatorial delay, all routing delays, and the final signal to output pin delay. Particularly for programmable logic, these delays are on the order of nanoseconds (as opposed to picoseconds, as is the case in ASICs). The total clock to output delay quickly passes the 11-ns spec.

Solution: The quickest, easiest, and most robust solution is to register the outputs. However, this solution has the trade off of adding one additional clock cycle. For long delay calculations, pipelining may be used to reduce variables. By doing the necessary calculations in a previous stage, the final stage can have a shorter total delay. Because PCI has wait states, the pipelining solution may be used.

3) All inputs require no more than 0 ns of hold time after CLK. (PCI spec 7.6.4.2)

Problem: Devices must have to be able to latch data with a 0-ns hold time.

Solution: It is necessary to use a part that can meet the 0-ns hold time spec. The Cypress 38x FPGA family meets the 0-ns hold time.

4) Configuration Space of PCI requires many registers.

Problem: PCI specifies that 256 bytes of register space be implemented.

Solution: Most of the 256 bytes of registers can be implemented as hard-wired zeroes. This reduces the need to use flip-flop resources to implement the configuration registers. In addition, some of the bits within the 32-bit registers may also be hard-wired to some permanent value. As a minimum, the configuration space will probably require a minimum of approximately 40 flip-flop registers for the simplest design.

5) Multiplexed Address/Data bus is routed to several places within the device.

Problem: PCI has a multiplexed address/data bus. The bus is accessed internally by several devices such as registers, FIFO, parity check/generators, comparators, and decoders. This requires the use of several 32-bit muxes.

Solution: Each logic cell of the 38x FPGAs has a cascaded muxing structure that can implement a 4-to-1 mux. By grouping signals into fours (along with their control signals), more optimal performance and utilization can be achieved.

6) PCI device must respond to a transaction within 3 clocks after the Address Phase.

Problem: After the first clock that the FRAME# signal has been asserted by the initiator, the addressed target must respond by asserting the DEVSEL# signal. If a target can respond within one clock, it is considered to be a "fast" response device. If it responds in two clocks it is a "medium" response device, and if it responds in three clocks it is "slow." The fourth clock after the asserted FRAME# signal is reserved for subtractive decoding devices such as bridges. If the initiator is not responded to within four clocks, it will abort the transaction. Therefore, most PCI devices must respond with the DEVSEL# signal within three clocks.

All targets have one clock (the first time FRAME# is sampled LOW) to latch the address and command from the PCI bus. They must immediately begin decoding the address to determine the recipient of the transaction. This is done by comparing the address to all implemented base address registers within the configuration space. If a hit is determined, then the target must assert its DEVSEL# signal. Parity (if enabled) is also checked on the second clock to determine if a parity error occurred during the address phase.

Solution: Pipelining the address compare function will allow the design to meet the timing. Remember that there is only a 7-ns set-up time on the bus, and therefore the first stage of the pipeline must be able to complete within this time (plus accounting for internal clock delay). Registering the DEVSEL# signal will insure the 2- to 11-ns clock-to-out time, but keep in mind that this will add an extra clock to your response time.

7) Parity

Problem: Even parity over the 32-bit A/D bus, 4-bit C/BE bus, and parity signal must be calculated and made available exactly one clock after a valid data transfer. Implementing the parity generator requires several levels of XOR logic. It should also have a small propagation delay to prevent excessive wait states during data transfers.

Solution: A single logic cell in the pASIC family can implement a 3-input XOR. Building the parity generation logic with 3-input XOR yielded a parity generator which utilized a minimal amount of logic cells and routing. The parity generator induced no extra necessary wait states.

8) High Fanout Signals

Problem: Several combinatorially produced signals have a very high fanout. For example, a signal will be used to enable 36 registers at once for data and byte enable latching. Signals with high fanout incur long propagation delays.

Solution: Inherent to all FPGAs, signal delays are often routing dependent. Reducing the number of loads on a signal, thus reducing the number of routing resources, can greatly improve performance. There are several methods for doing this: split buffering, selective buffering, paralleling, and double buffering. For more information on buffering techniques, see Chapter 4 "Design Techniques" of the *Warp3™* User's Guide, SpDE/Warp System section.

Split buffering involves inserting another layer of logic between the signal source and all of its loads. For example, if a signal has 10 loads, the loads can be split into two groups of five. Each group would then be driven by a BUF component, which in turn are driven by the signal source. This reduces the load of the original signal to just two.

Selective buffering is similar to split buffering: an extra buffering layer is inserted. The difference between the two methods is that a few of the original load signals are more timing-critical than the others. In this case, those critical signals should be driven by the original source (the same level as the buffers).

This effectively reduces the load of the original signal, without adding extra logic between the source and the timing-critical signal.

Paralleling has the advantage of no extra layers of logic with the trade off being a complication of the design. This method involves repeating the signal's source logic. For example, if the signal is produced by an AND gate, this AND gate would be repeated (both with the same input values) and each gate would then drive its own group of logic.

Signals with larger fanouts or speed-critical signals should be buffered using the DOUBLE_BUFFER attribute in their design. Keep in mind that every time this technique is used, express wires in the device are used. Using this attribute without discretion can quickly exhaust all available express wires within the device. A second improvement to double buffering is to place the flip-flops in a single column. This has the advantage of shorter signal paths and uses less express wires. To place flip-flops, use the FIXED_FF attribute on the registered signals.

Making Modifications to the Design

Assigning Values to Configuration Registers

Configuration registers DEVICE ID, VENDOR ID, CLASSCODE, and REV ID must be assigned. To do so, edit the configuration space block: C_SPACE.VHD (VHDL). These registers are declared as constants and their assignments may be changed to the appropriate values.

Changing Address Space Size

Different applications will have different address space size demands. Modifications of the design to match size demands is expected. Since a device's address space is determined by the number of hardwired zeroes in the lower bit positions, decreasing the address space size increases the number of bits compared to determine an address hit. Likewise, if the address space size in increased, the number of bits compared goes up. To customize this design to meet an application's address space size demands:

1. Edit the configuration space VHDL.
   - Modify the signal declaration of BASE_ADDR_X to be the appropriate size bit vector.
   - Locate where the base address is assigned a value from the data bus and modify the BASE_ADDR_X and PCI_DATA vector sizes to the appropriate size.
   - Locate where the base address values are sent to the output pins of the configuration space block and modify the BASE_ADDR_X vector and number of concatenated zeroes to the appropriate size.
2. Modify the address compare logic
   - The schematic of the decode logic (xxP_DEC) is a nibble oriented design. The number of bits compared in this circuit should reflect the number of bits necessary to determine an address hit.
3. Modify the user address counter
   - The burst length of a device does not always reflect the size of the address space. In this design the user addressing counter allows double word burst lengths of 16. The size of the counter may be modified to meet the required burst length.
   - Regardless of the size of the burst length counter, all lower bit positions must be sent as output to the user address pins to cover all locations in the allocated address space.

Target Aborts and Retries

The control logic of this design is ready to handle target aborts and retries. However, as the design stands, no logic uses this functionality. (Notice REQ_ABORT and REQ_RETRY inputs to the CONTROL block are grounded.) Logic for determining target aborts or retries may be added, and used to signal the CONTROL block to perform the target abort/retry.

Increasing the Depth of the FIFO

The control logic of this design utilizes a 36-bit register for write operations. PCI interface side logic performs a data transfer when it sees that the regis-

ter is not full. The user interface side logic performs a data transfer when it sees that the register is full. Minor modifications to this logic should be done to support an internal FIFO.

Conclusion

Interfacing with the PCI bus is a task of intricate protocols, timing specs, and data handling. However, the PCI challenge can be met by using PLDs, and in particular, FPGAs. The flexibility, high density, and compliance of Cypress FPGAs make the FPGAs ideal candidates for PCI bus interface applications such as add-in cards.

*Warp*3 is a trademark of Cypress Semiconductor Corporation.

PCI read and write transactions are inherently non-preempted burst transactions. The basic protocols are the same for configuration, I/O and Memory read and writes. All PCI bus devices implement configuration space registers which give PCI its plug-and-play nature.

Because of the many issues involved in PCI interfacing, a designer will inevitably run into a multitude of challenges. Careful planning and use of this application note and reference design can provide a head start in the design process.