PCI Target Designs Using Ultra37000 CPLDs

Introduction

The Peripheral Component Interconnect (PCI) bus is a high-bandwidth, "plug-and-play" bus protocol 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. There are many PCI features, not all of which may be required for any given application. This makes programmable logic ideal for PCI interface applications. Not only can the PCI interface be tailored for a specific application, but additional logic can be incorporated into the programmable logic. This can make a PCI programmable logic solution more attractive than an ASIC solution.

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 example designs for a PCI target device that has been implemented in Cypress Ultra37000 CPLDs. This note covers the basics of PCI and documents the capabilities of two PCI target designs and the associated test bench to confirm functionality.

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 are accessible from the PCI bus and transact in an initiator/target relationship. The initiator is sometimes called a master and the target is sometimes referred to as a slave. For the purposes of this note the terms initiator and target are used instead of master and slave.

PCI Bus

The PCI spec 2.1 specifies the PCI operating speed from 0 to 66 MHz with a 32-bit synchronous bus, expandable to 64 bits. PCI is 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 Interface Signals

A PCI interface device must have at least 47 pins. A PCI initiator device has 2 additional pins REQ# and GNT#, which brings the total number of required pins to 49. Optional pins provide 64-bit operation, JTAG boundary scan, target locking,

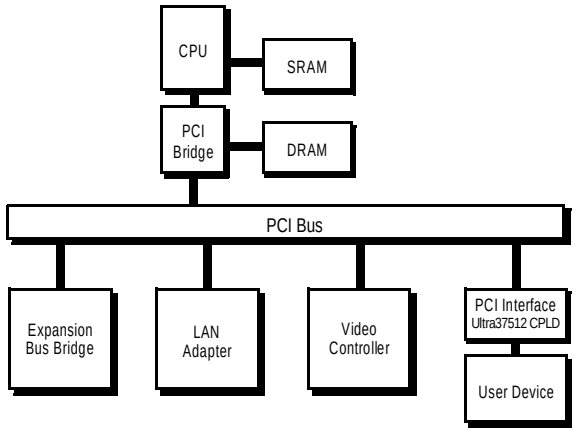

Figure 1. PCI Architecture

cache support, and interrupt expandability to the PCI bus (see *Figure 2*). Active LOW signals end in a "#" character in this figure. For a target device the direction of the required signals changes as follows. The C/BE# signal is input to the device. The FRAME# and IRDY# signals are input to the device. The TRDY#, STOP#, DEVSEL#, and PERR# signals are outputs of the device.

There are five different types of PCI signals: All of these types are used in the PCI target designs except for type "out".

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

PCI Bus Commands

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

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 de-

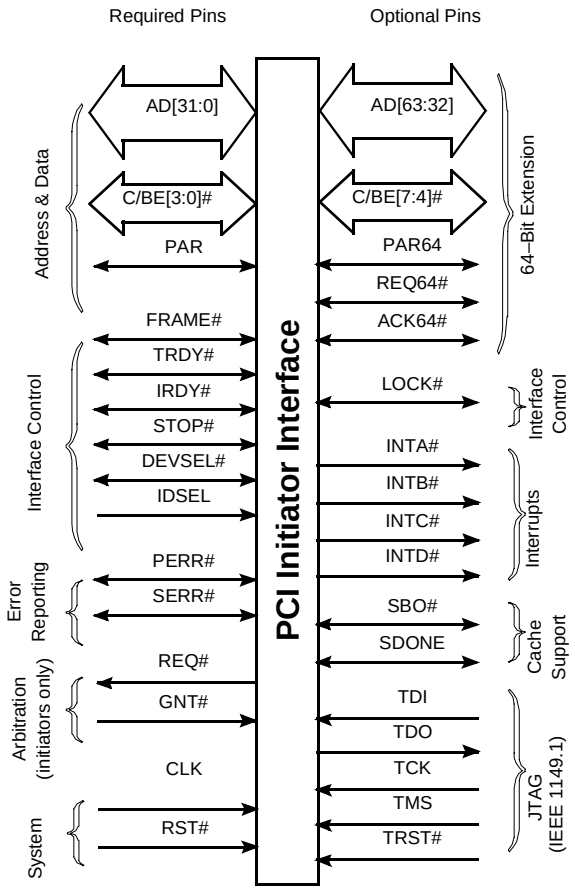

Figure 2. PCI Initiator Pin Diagram and Signal Direction

vices 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. The target designs only implement a small portion of this configuration space and return zeros for those functions not supported as required by the PCI specification.

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 3*). One other header type is defined, type 01, which pertains to PCI-to-PCI bridges. The two-digit type refers to the two LSB bits in the 32-bit address AD.

Device ID - Device identification number issued by the vendor

Vendor ID - Vendor identification number issued by the PCI Special Interest Group (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

Table 1. Required Pins

Pin Name	Type	Description
AD[31:0]	t.s.	32-bit bidirectional multiplexed address/data bus
C/BE[3:0]#	t.s.	Command/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 for configuration read and write transactions
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.	Four Interrupt pins

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

The latter 192 bytes of the 256 bytes of configuration space are defined for device-dependent operations. 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 3*) 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. The target designs compare the top 16 bits of the address to determine an address hit.

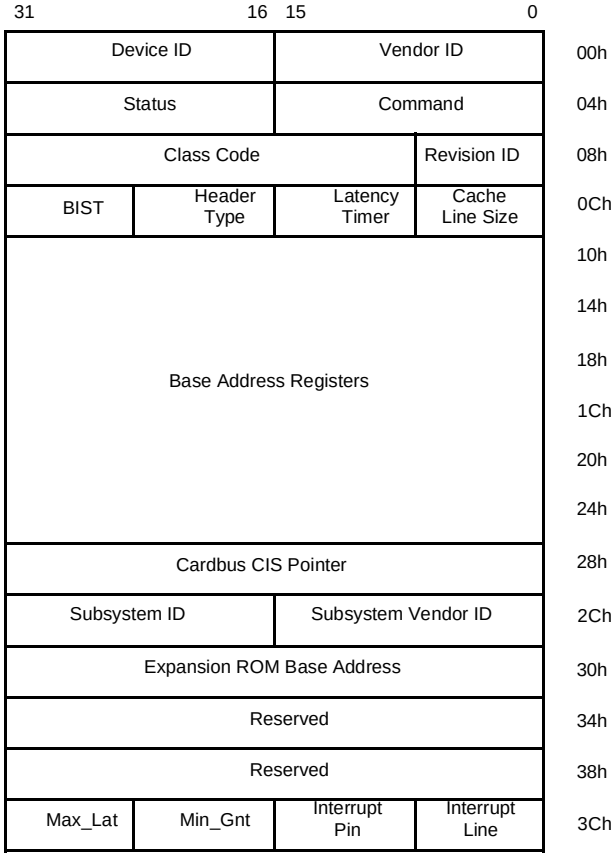

Figure 3. Type 00h Configuration Space Header

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 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 waits for a target device to respond and claim the transaction by asserting its 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 the initiator to the target.

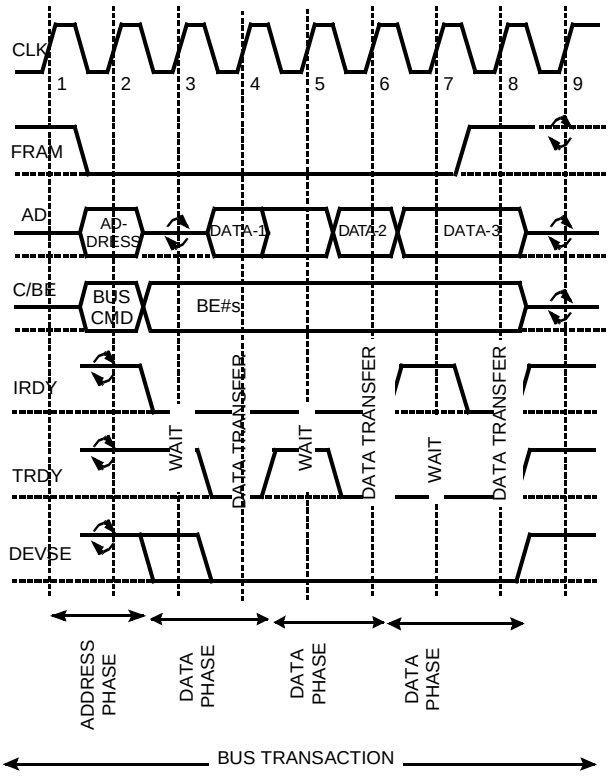

Figure 4. Read Transaction Waveform

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 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 initiator controls the bus for the entire duration of the transaction.

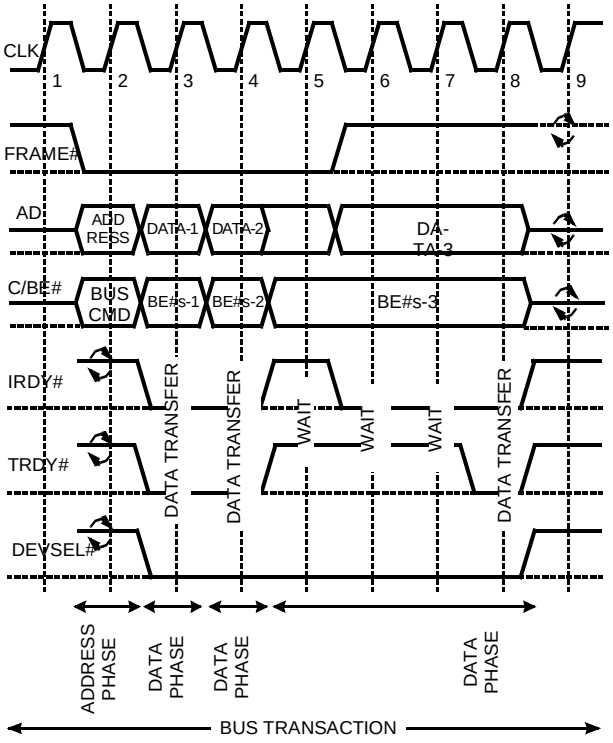

Figure 5. Write Transaction Waveform

Claiming the Transaction

Figure 6 shows the speed at which a PCI device can claim a 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. 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. The target designs are “medium” speed.

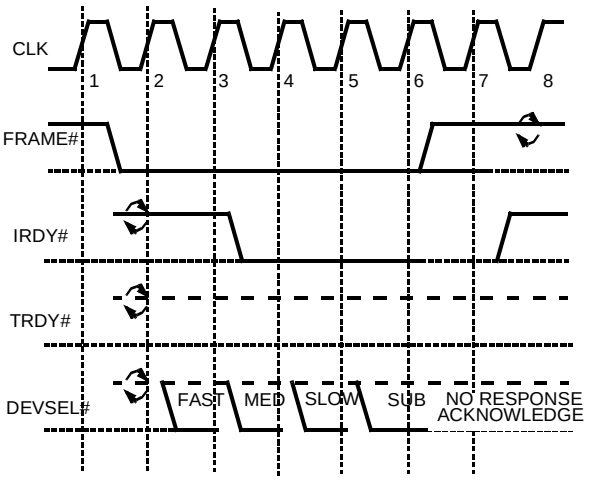

Figure 6. Transaction Claiming Speed

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. Even parity means that the PAR bit is HIGH if there are an odd number of HIGH lines in the address and the command lines and is a LOW if there are an even number of HIGH lines in the address and the command lines. 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.

Ending the Transaction

The initiator signals the end of a transaction by deasserting FRAME# and asserting IRDY# which tells the target that one more data phase will take place and then the current transaction will end. PCI provides several methods whereby the initiator directly ends a transaction. Additionally PCI provides several means whereby the target signals to the initiator to end the transaction.

There are three scenarios whereby 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 the 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 ends the transaction on the sixth clock.

There are four ways that the target can signal to the initiator to terminate the transaction:

1. Disconnect with data—The target may induce a disconnect by asserting both STOP# and TRDY# which completes a single data phase transfer with data and instructs the initiator to end the transaction.
2. Disconnect without data—The target may induce a disconnect by asserting STOP# and de-asserting TRDY# which completes a single data phase transfer without data and instructs the initiator to end the transaction.
3. 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 same 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. The initiator must attempt the same transaction to the target at a later time if the target aborts with retry.
4. 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# and TRDY#. A target abort can additionally be signalled by a target if it can not implement a particular initiator transaction. The initiator must not attempt to repeat a transaction with a target that has terminated a transaction with a target abort.

Recommended Device Pinout

The PCI spec recommends the pinout shown in *Figure 7* to minimize signal length stubs. The assignment of the pins to specific locations on the connector is defined in the PCI specification for 5-volt and 3.3-volt cards and for 32-bit and 64-bit cards. See "Connector Pin Assignments" section in the PCI specification.

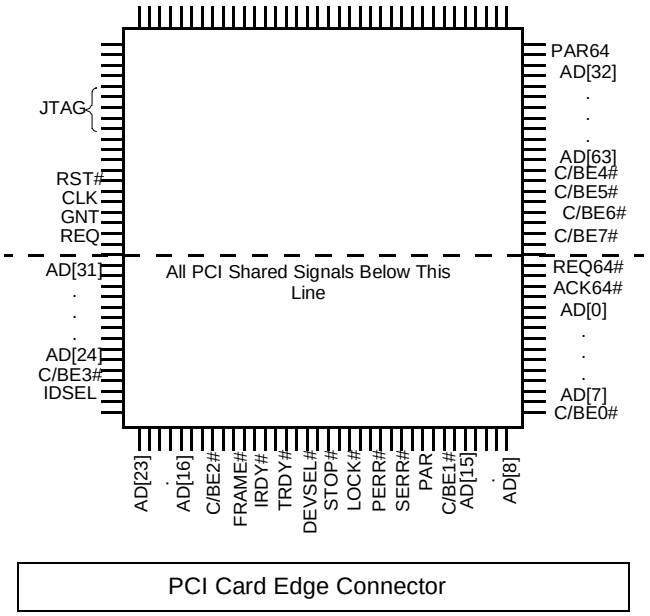

Figure 7. Recommended Pinout

Timing

Table 4 shows the timing requirements for all the target design parameters. *Table 5* shows the timing requirements for the initiator parameters Req and Gnt parameters. These signals are required by the initiator design to request and be granted permission to use the PCI bus, with permission granted by an arbitrator.

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. Req/Gnt Pin Timing Specification

Symbol	Description	Min.	Max.
t_{val}	Clock to Data Valid for Req/Gnt	2	12
t_{su}	Input Set-Up Time for Req	12	
t_{su}	Input Set-Up Time for Gnt	10	

Interrupts

Whenever a device wants to call an interrupt, it does it through the PCI bus mechanism. For example, there may be data waiting to be transferred on a target and so it would signal an interrupt to make the system aware of its desire for a transfer.

The interrupt connects to the PCI bus through one of four lines (INTA#, INTB#, INTC#, INTD#). In the Cypress PCI designs, only INTA# is used since one interrupt is supported. Multiple devices could be connected to the same interrupt pin.

A PCI device wanting to request an interrupt drives its INTA# LOW. Multiple devices can drive the interrupt line (INTA#) LOW, since it is open drain, which causes the system to generate an interrupt.

Driving this bus interrupt line LOW causes the interrupt service routine to check who's issuing the interrupt. It does this by polling all the devices on the chain connected to that interrupt line, such as all the devices on the INTA# chain or all the devices on the INTB# chain. The order that the devices are polled depends on the way the interrupt bus was designed, which is not specified in the PCI Specifications.

The devices are serviced in the order that they are polled. That is, the interrupt service routine checks each device in that interrupt chain, one at a time. If it finds that an interrupt is pending (Interrupt Status register config address 40h bit 0 asserted) it services the interrupt by running the appropriate device driver code. The interrupt controller would continue sequentially servicing any other pending interrupts in the chain.

As each interrupt is serviced, the target three-states its INTA# signal. Once all of the devices wanting interrupts have had their interrupts serviced, INTA# would no longer be driven LOW by any of them and the line is returned HIGH by the pull-up device on the interrupt line. See specific interrupt information related to the target designs later in this application note.

The rest of this application note covers specifics of the Cypress PCI target reference designs and what PCI features are implemented.

PCI Target Designs

Two PCI target designs are offered, a basic design and an advanced design. The basic design transfers data in non-burst mode, meaning only one byte of 32 bits is transferred per transaction. The advanced design provides burst mode support which enables more than one byte of data to be transferred per transaction. Burst mode transfers enable large amounts of data to be transferred on a single transaction thereby yielding higher bus performance. This is the major difference between the advanced PCI target design and the basic target design. Additionally, the advanced design supports interrupt and abort handling. Both designs incorporate "medium" transaction claiming speed since the DEVSELn signal can become active one clock after FRAMEn is sampled asserted. The remainder of this application note explains the capability of these two designs.

General Description

The block diagram of *Figure 8* shows the functionality of both designs. The sub-blocks are divided into level1 and level2 blocks, which are separated in time by one clock delay. The arrows show the direction of data flow. Lines with no arrows indicate data flow in both directions or the signals in the group may have different directions.

The pin groupings are shown in *Table 6* and *Table 7*.

Features

The designs offer the following features:

- 32-bit, 33-MHz operation is supported for the Ultra37000 and Ultra37000V families.
- Back-end support is included for FIFO (seamless connection to Cypress synchronous FIFOs).
- Parity and system error generation and reporting is available for address and data respectively (SERR#, PERR#).
- Four commands are supported: Configuration Read/Write, and Memory Read/Write.
- One 32-bit base address is supported.
- There are 32-bit user data ports, separate input and output.
- Intelligent Target signaling is supported in the advanced design: disconnect when FIFOs are full/empty; throttling when write FIFO almost full.
- Interrupt handling is supported in the advanced design. Configuration space settings enable programmable transaction requests to ignore or process the interrupt.
- Abort handling is supported in the advanced design.

Table 6. Front End Interface

Pin Groups	Name	Type
Interface/Control	FRAMEn	I
	IRDYn	I
	TRDYn	O
	STOPn	O
	DEVSELn	O
	IDSEL	I
Address/Data	AD[31:0]	I/O
	C/BEn	I
Parity	Par	I/O
Error Reporting	SERRn	O
	PERRn	O
Interrupt Request	INTAn	O
System	CLK	I
	RST	I

Table 7. Back End Interface

Pin Group	Name	Type	Target Description
Address/ Data	DATAIN [31:0]	I	Data input from read FIFO
	DATAOUT [31:0]	O	Data output for write FIFO
	BE[3:0]	O	Byte enable for data
Interface/ Control	WRITE_ENn	O	Write FIFO enable
	WRITE_ FULLn	I	Write FIFO full
	WRITE_ AFULLn	I	Write FIFO almost full (advanced design only)
	READ_ENn	O	Read FIFO enable
	READ_ EMPTYn	I	Read FIFO empty
Interrupt Request	EXT_INT	I	Target request an interrupt (advanced design only)
	ABORT	I	Target request an abort (advanced design only)

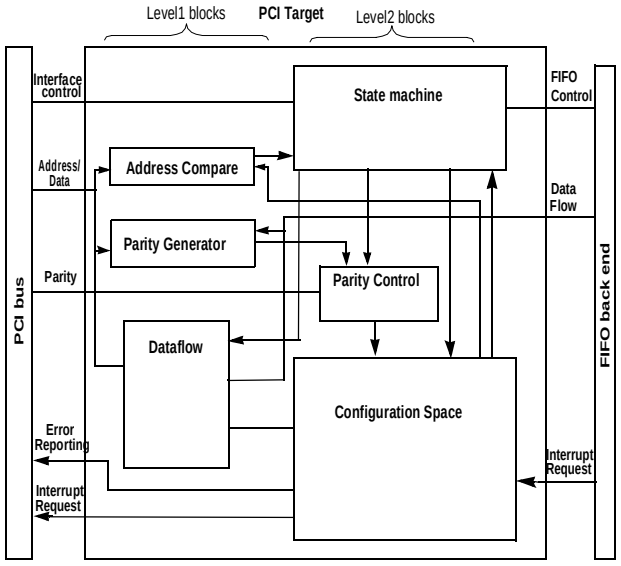

Figure 8. PCI Target Block Diagram

The PCI Target design is the interface between the PCI bus and a synchronous FIFO as shown in *Figure 8*. The FIFO then interfaces with the actual target device.

Some logic blocks have additional sub-blocks. The address/data port flows unidirectionally towards the address compare and parity generator blocks but bidirectional towards the dataflow block.

The configuration space and target state machine processes are run concurrently since their results are needed at the same time. However, signaling an error from the parity control block is delayed by one clock cycle: its result is synchronously used by the configuration space block the next cycle. Following is a brief description of each of the function blocks.

Address Compare

The address compare compares the configuration memory base address, 10h, with the address on the PCI bus, indicating the result to the target state machine.

Parity Generator

The parity generator calculates an even parity over the combined ADDR and CBE# lines, 36 bits wide, on the PCI bus indicating the result to the parity control block.

Dataflow

The dataflow block, *Figure 10*, directs the flow of data between the PCI bus, configuration memory and back-end, according to the target state machine control. For both advanced and basic designs the data and command/byte enable is latched in a 36-bit register. A single 32-bit register is used for both reading and writing. The advanced design has an additional 32-bit register for storing data for burst read operations, rf_reg2. During the read operation data is loaded into the ad_reg directly from the FIFO or from the 32-bit register. The purpose of the 32-bit register is to always have valid data available on the bus. It is further explained in the memory read section.

Target State Machine

The target state machine, *Figure 9*, monitors the interface control lines, the back-end control signals, and the status of the device via the configuration memory. The state machine interface control lines direct the data flow and the appropriate back-end control signals. The basic design consists of three states: idle, compare, and transfer. State idle is the inactive state. State compare is entered when the initiator begins a transaction on the bus and the target inspects the address to determine if the transaction belongs to the target. If so, then the state machine enters the transfer state and issues the appropriate write or read control signals based on the loaded instruction. The advanced design also includes a fourth state, backoff, which is entered to properly disconnect from the PCI bus upon interrupt and abort operations.

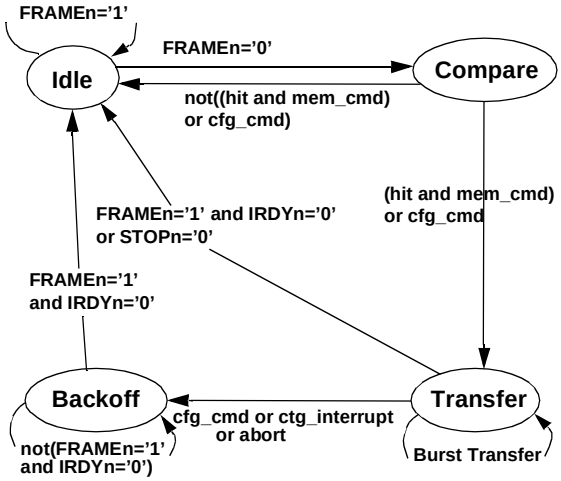

Figure 9. State Machine

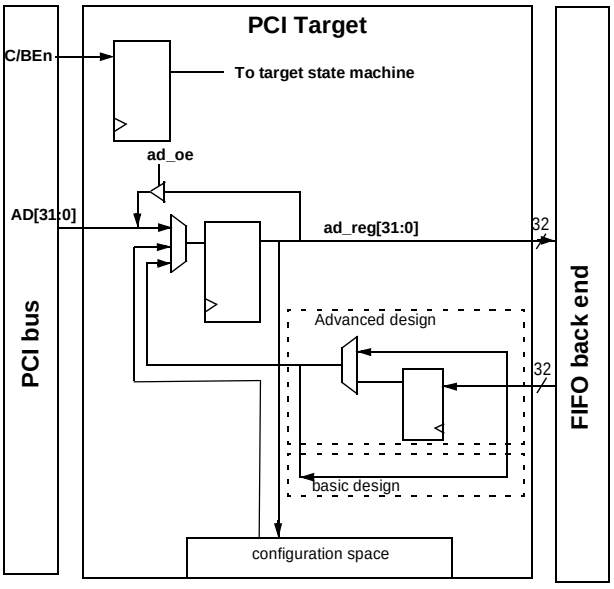

Figure 10. Dataflow Block

Parity Control

Parity control either directly outputs the result of the parity generator, when parity is generated from the target as during read operations, or indicates whether the bus parity matches the one generated from the parity generator, when parity is generated from the initiator. The target state machine signal selects which function is done. Parity is the result of the xoring of all the AD and CBEn signals, a total of 36 bits. PCI requires even parity to be generated, which means that the number of ones on these 36 bits plus the parity bit is an even number. Parity is valid one clock after the AD/CBEn bits are latched. PERRn or SERRn is valid one clock after the parity is valid. Parity generation requires three passes through the CPLD.

Configuration Space

Figure 11 shows what portions of the configuration space are implemented in the target designs.

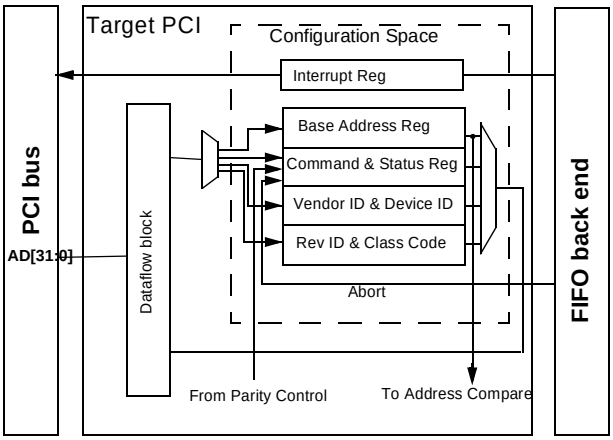

Figure 11. Configuration Space

Read and write operations on the configuration space originate from the initiator that owns the PCI bus, the parity control block, or the back end device for aborts and interrupts. Operations that initiate from the system via the PCI bus, permit the system to understand the characteristics of the target such as the target functionality, memory requirements, etc. The configuration space gives the target device its "plug and play" capability.

The base address register performs two functions which are to inform the system whether the target device accesses I/O space or memory space and how much memory is required by the target. It is also used by the address compare block to determine if the PCI bus transaction belongs to the target, when a "hit" occurs.

The command register has three locations used which are memory access enabled, parity error report, and system error report. Parity error report enables the PERRn signal to be driven back to the initiator, and the system error report enables the SERRn signal to be driven back to the initiator.

The status register has four locations used which are detected parity error, detected system error, signaled target abort, and medium DEVSELn speed. The abort location is only implemented in the advanced design.

Only the LSB of the interrupt register is used in the designs. The interrupt space is only implemented in the advanced design.

The vendorID identifies the manufacturer of the PCI device. The deviceID identifies the device type. The revID identifies the revision number. The class code identifies the basic function of the device. The vendorID is assigned by the PCI SIG to assure uniqueness.

Interrupt Control

The interrupt control is controlled by settings in the configuration space block. Only the LSB of an 8-bit register is used in the advanced design. See additional information on interrupt control later in this applications note.

Front End and Back End Signals

Table 6 and *Table 7* show the front and back end signals. Below is a description of the back end signals since the front end signals are explained in *Table 1*. All signals that end in a 'n' are active LOW signals.

WRITE_ENn must be asserted on the clock edge that data is put on the DATAOUT port driving into the FIFO.

WRITE_AFULLn can be asserted by the back-end device to throttle, slow down, the flow of data: the target now only accepts one new word of data per transaction. This signal is only available in the advanced design when the FIFO is close to being full.

READ_ENn is asserted one cycle before data is read from the DATAIN port from the FIFO.

EXT_INT only needs to be strobed HIGH for one cycle for the target to signal an interrupt to the PCI bus.

ABORT only needs to be strobed HIGH for one cycle for the target to go into target-abort mode. This should not be used to end transactions but is reserved for critical failures in the back-end, such as the back-end not being able to complete a particular transaction.

VHDL Code Detailed Description

There are four VHDL design files for the PCI Target designs: PCI_PKG.VHD, CONFIG.VHD, PARGEN.VHD and FIFO_TGT.VHD. They are described below.

PCI_PKG.VHD contains customizable constants in the configuration space such as base address size, vendor ID, device ID, revision ID, class code, etc. It also contains global constants used by the target, and component declarations for entities.

CONFIG.VHD contains the behavioral description of the configuration memory. Register use is minimized to conserve resources, hence reading/writing to most locations has no effect. Parity error signalling is done from this entity.

PARGEN.VHD is the parity generation code: a 36-bit even parity generation. Parity generation takes three passes through the CPLD logic array to implement. The synthesis_off attribute directs the *Warp* fitter to create partial results on a single pass through the CPLD that are then fed back for additional passes to implement the logic. The attribute synthesis_off statements must be uncommented during synthesis and commented during test bench compilation.

FIFO_TGT.VHD contains the behavioral description of the PCI back end FIFO device.

Writing and Reading to Configuration Space and Memory

The advanced PCI Target responds to the following configuration space commands:

Configuration Writes

Address 04h sets the command register bits accordingly. Only bits 1, 6, and 7 are changeable.

Address 10h sets the base address which has a writable length specified in PCI_PKG.VHD.

Address 3Ch sets the interrupt line.

All others have no effect.

Configuration Reads

Address 00h returns the device/vendor information.

Address 04h returns the status/command register values.

Address 08h returns the class code/revision ID.

Address 0Ch returns the BIST, header type (not in basic design).

Address 10h returns the base address.

Address 3Ch returns the interrupt pin and interrupt line (not in basic design).

Address 40h returns interrupt pending info (not in basic design).

All other reads return null strings.

Memory Writes

If the address matches the base address register, the transaction is accepted.

Multiple 32-bit words are read from the bus AD[31:0] and written to the DATAOUT[31:0] backend when the WRITE_ENn is asserted LOW. The basic design only reads out a single word by implementing a target disconnect with data on each data transfer.

For the advanced design, the target accepts data until the initiator indicates the end of the transaction.

For the advanced design, as soon as the WRITE_AFULLn is asserted, the target only accepts single 32-bit words per transaction which gives the back-end time to process the data.

Memory reads

If the address matches the base address register, the transaction is accepted.

A 32-bit word is read the cycle after READ_ENn is asserted LOW. READ_ENn is kept LOW and data is read and put on the bus until the initiator indicates it doesn't want more data or the target runs out of data, which is indicated by the signal READ_EMPTYn. The basic design reads only one word per transaction.

If the WRITE_FULLn is asserted, any memory write operation is terminated by the target with disconnect without data. Similarly, if READ_EMPTYn is asserted, any read operation is terminated by the target with disconnect without data.

Parity errors are always noted by bit 15 of the status register. Any further action depends on whether bit 6, for parity error reporting, or bit 8, for system error reporting, are set. Address parity errors are signalled with SERRn for one cycle. Parity errors are signalled with PERRn for one cycle.

Interrupts during transfers may be turned away with target retry requests depending on the setting in PCI_PKG.VHD. See the Interrupts section for more information.

Back-end aborts are acted upon immediately and no more data is transferred.

Configuration Read Timing Diagram

The PCI Target responds to any configuration read operation with a disconnect with data, asserting TRDYn and STOPn at the same time, giving only one piece of data, 32 bits wide, for the configuration memory address requested. *Figure 12* shows the timing of a configuration read operation sequence.

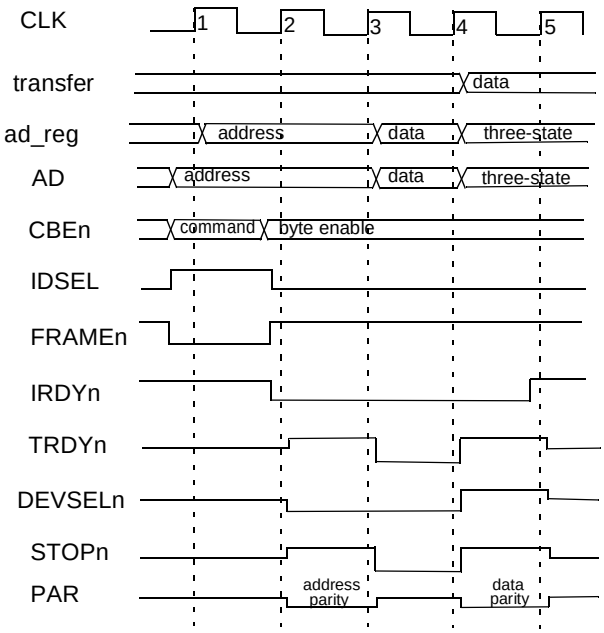

Figure 12. Configuration Read Timing Diagram

Each rising clock edge is numbered and does the following:

1. The address phase: The target sees that its own IDSEL signal has been asserted and that CBEn indicates a configuration read. The IDSEL line must be asserted to initiate any configuration read/write command.
2. The target accepts the transaction and asserts DEVSELn. The generated parity results are also compared to the parity provided by the initiator, PAR. If there is not a match then the SERRn is asserted on the next cycle.
3. The target implements a disconnect with data transaction request, which indicates it gives only one piece of data. The initiator only wants one piece of data as FRAMEn is deasserted and IRDYn is asserted. The next cycle it puts the appropriate parity, PAR, on the bus for this data.
4. The target drives the bus lines HIGH for one cycle, TRDYn, DEVSELn, and STOPn. Note that the transfer data is also shown in this figure and is the data that gets latched into the input register of the initiator. There is one cycle delay between the transfer data and the data output on the PCI bus, ad_reg for read operations. In general when the target outputs to the PCI bus then signal ad_reg and AD are the same signal. When the PCI bus drives data into the target during write operations, there is one clock delay between the AD signal and the ad_reg signal.
5. The target three-states the bus lines and goes to sleep.

Configuration Write Timing Diagram

The PCI Target responds to any configuration write operations with a disconnect with data, asserting TRDYn and STOPn at the same time thus allowing only one piece of data, 32-bits wide, to be written to configuration memory. Data is written to the configuration memory address specified only if that memory location is implemented in the design. *Figure 12* timing also applies to write operations.

Memory Read Timing Diagram

The advanced PCI Target responds to any memory read operations transfer request and gives as many pieces of data, 32-bits wide, as requested, or until the FIFO is empty. *Figure 13* shows the timing of a memory read sequence operation.

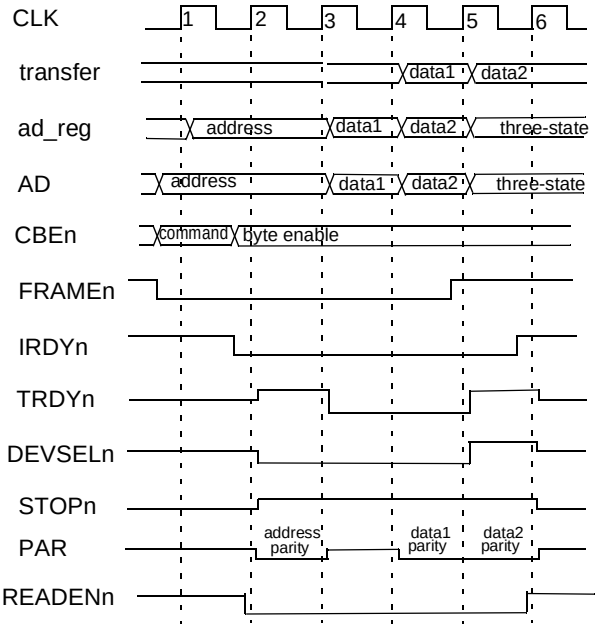

Figure 13. Memory Read Timing Diagram

Each rising edge is numbered and does the following:

1. The target checks if the address matches the base address register and if CBEn indicates a memory read.
2. The target accepts the transaction, asserting DEVSELn, and checks that parity, PAR, matches the address. It then indicates errors with a system error, SERRn asserted on the next cycle if parity does not match. It indicates to the back-end that it needs data by asserting READ_ENn.
3. Read operations commence unless the FIFO is empty. At the beginning of the transaction data is put on the bus on each cycle until either the back-end has no more data or the initiator doesn't want any more data and signals a transaction end. Valid parity, PAR, is put on the bus for each data cycle, the cycle after the data.
4. The second data phase occurs. Continuous transfers would look identical to this cycle.

5. The initiator proceeds to terminate the transaction. The target stops giving data and drives its bus lines HIGH for one cycle, TRDYn, DEVSELn, and STOPn.
6. The target three-states the bus lines and goes to sleep.

Memory Read Timing with Initiator Inserted Wait States

During the burst read operation, the initiator can insert wait states at any time which is signalled by driving the IRDYn signal HIGH. When this occurs the data flow of the FIFO must be halted because the initiator is not actually reading the data on the bus. The target design needs one clock cycle to turn off the flow of data down the FIFO. The result is that one word of data is lost. To prevent this data from being lost the design stores this data word in a register. When the initiator again requests data, the IRDYn signal goes LOW, and the initiator gets the correct data that is stored in the register and not a new piece of data from the FIFO. This is accomplished by the incorporation of a 32-bit register and a mux that selects either the FIFO out data or the register data depending on the state of two flags “rf_reg1_vld” and “rf_reg2_vld”. Under normal burst mode operation “rf_reg1_vld” is valid and “rf_reg2_vld” is invalid. When the initiator inserts a wait state “rf_reg2_vld” becomes valid, thereby signaling to the initiator to take the next piece of data from the register. On subsequent data transfers data again is taken directly from the FIFO as the FIFO dataflow turns back on. This logic just described resides in the “dataflow” process in the advanced design.

Memory Write Timing Diagram

The advanced PCI Target responds to any memory write operations and takes as many pieces of data provided by the initiator, or until the FIFO is full. The basic design writes only one piece of data per transaction. The timing is very similar to the memory read timing and is shown in *Figure 14*.

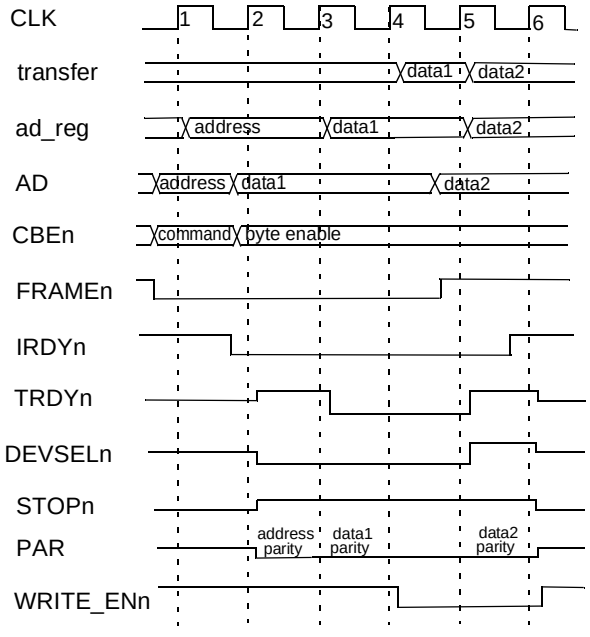

Figure 14. Memory Write Timing Diagram

Each rising edge is numbered and does the following:

1. During the address phase the target sees that its own IDSEL has been asserted and that CBEn indicates a memory write.
2. The target accepts the transaction, asserting DEVSELn, and checks that parity, PAR, matches the address. It indicates errors with a system error, SERRn asserted during the next cycle.
3. The transaction starts and the target indicates its ready to receive data provided the FIFO is not almost full. Data transfer waits for the assertion of WRITE_ENn, which is asserted when both the initiator and the target are ready to receive data.
4. The data transfer occurs. One or more cycles like this would occur to exchange data from initiator to target. Every time valid data is on the back-end DATAOUT port, the WRITE_ENn is asserted. The next cycle [5] it checks that the parity, PAR, matched the data on the bus on the previous cycle.
5. The initiator indicates it doesn’t want more data and ends the transaction, FRAMEn deasserted and IRDYn asserted. The target accepts its last piece of data and drives its bus lines HIGH, TRDYn, DEVSELn, and STOPn, for one cycle.
6. The target three-states its drivers and goes to sleep. It asserts parity error, PERRn, if it detected incorrect parity.

Target-Abort Timing Diagram

The advanced PCI Target responds to any abort operation by doing only one more transfer and then waiting for the initiator to acknowledge the target-abort. The abort feature is not available in the basic design. The target abort timing is shown in *Figure 15*.

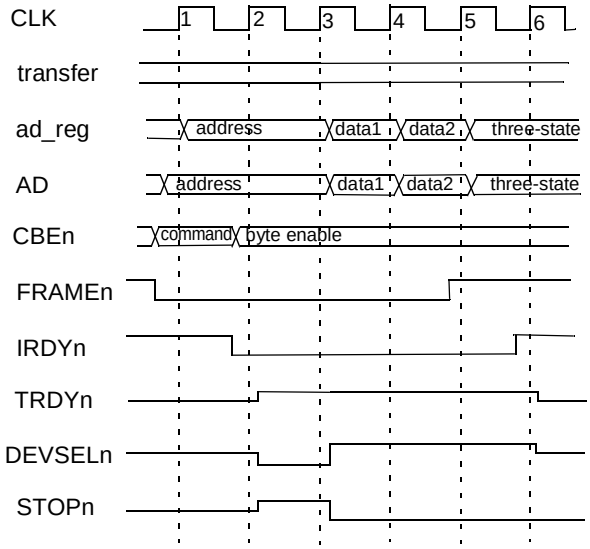

Figure 15. Target Abort Timing Diagram

Each rising edge is numbered and does the following:

1. During the address phase, the target checks for ownership of the transaction from the address compare block.

2. The target claims the transaction by asserting the DEVSELn signal.
3. An abort is indicated by the back-end by deasserting DEVSELn and TRDYn and asserting STOPn. The target indicates a target-abort and waits for the initiator to respond.
4. The initiator proceeds to process the abort by ending the transaction, deasserting FRAMEn on the next cycle.
5. FRAMEn deasserts to signal the end of the transaction.

Interrupt Timing Diagram

The advanced PCI Target responds to any interrupt requests, sensing EXT_INT driven HIGH for one cycle, by driving the INTA line LOW until the interrupt is acknowledged. No further memory operations are permitted until the interrupt is cleared. The INTA signal's open drain function is implemented by setting the signal equal to ground and providing the logic to the output enable of the signal. The interrupt feature is not available in the basic design. Interrupts are cleared by reading the interrupt pending bit in the configuration memory, address 40h. *Figure 16* shows an example of an interrupt timing diagram.

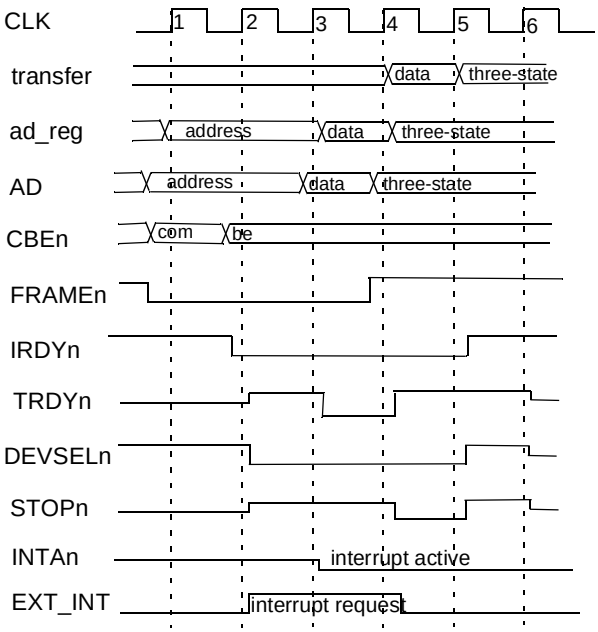

Figure 16. Interrupt Timing Diagram

Each rising edge is numbered and does the following:

1. The FRAMEn signal initiates a transaction.
2. The interrupt is requested from the back-end as the EXT_INT signal is asserted.
3. The interrupt request is indicated on the PCI bus as signal INTAn is asserted. The internal signal cfgINTDISCON is asserted if the DisconnectOnInterrupt configuration space is set.
4. The interrupt request remains LOW until it is acknowledged by having the configuration memory location 40h read, which indicates that an interrupt request is pending. The target signals a disconnect if cfgINTDISCON is asserted by asserting STOPn and deasserting TRDYn.
5. The target drives its signals HIGH as transaction ends and three-states the signals the cycle after.

Conforming to PCI Specs

This design has been written to conform to the PCI operational specifications. If the code is modified in any way, including pin locking restrictions, the user should check to make sure that it still conforms to the PCI Specification. This is best checked by inspection of the testbench functional simulation results.

The report file should be reviewed for the three timing parameters relating to the front-end signals that must be met: t_S, set-up time, must be less than 7 ns; t_{CO}, clock to output, must be less than 11 ns; and t_{SCS}, register feedback through CPLD array to register, must be less than 30 ns.

Interrupts

Interrupts are initiated by driving the back-end signal EXT_INT HIGH for one cycle. The PCI target automatically requests an interrupt. INTAn is driven LOW by the target, and an interrupt pending flag is set.

The address of the Interrupt Status Register within the configuration memory is 40h. Bit 0 indicates that an interrupt is pending when asserted, set HIGH. Note that after the status register has been read once, it is cleared. It is up to the system routine checking to remember that this device wanted an interrupt.

Two configuration memory address locations pertain to interrupts, 3Ch and 40h. In address 3Ch, the bits indicate what type of interrupts that device can generate, in this case 01h for INTAn only. The bits for the interrupt line are used by the BIOS. After power-up/resets they default to FFh. In address 40h, the Interrupt Status register bit 0 determines whether an interrupt is pending. A HIGH indicates that the device is requesting an interrupt.

In PCI_PKG.VHD, the DisconnectOnInterrupt flag can be adjusted 'TRUE' to command the target to disconnect a transaction as soon as an interrupt is requested, as well as turn away all further transaction requests with target RETRYs until the interrupt is serviced.

Target Response to Reset

When the RST signal is asserted all PCI signals are asynchronously three-stated. Additionally the command and status registered bits that are used in the design are set to zero. The state machine asynchronously returns to the idle state.The base address register bits are set to one. The user will want to adjust the "baseaddr_size" constant so that the correct number of bits are set to one upon reset. This number of ones in the base address register determines how much memory space is required by the PCI device.

PCI Testbench

The testbench is written in VHDL so it can be used under many different platforms. The Aldec Development Environment was used to develop and test the code. A FIFO entity is attached to the back of the target to test that the target properly interfaces with the FIFO. A protocol checker verifies that transactions between the target and the initiator meet the PCI Specification with respect to the signals TRDYn, DEVSELn,

STOPn, IRDYn, FRAMEn and PAR. Error messages report simulation violations.

The testbench checks the protocol, per PCI Spec v2.1, of interactions between the target and the initiator and tests the functionality of the target design. It allows the user to easily run many different combinations of Configuration Read/Writes and Memory Read/Writes and allows the user to visually inspect the target response in the form of signal waveforms in the Aldec environment.

VHDL Design Source Code

test_pkg.vhd: This file is the meat of the PCI initiator simulator. It contains PCI test procedures such as "procedure Read-Write" which performs burst read or burst write operations. There are a few levels of hierarchy in this file meaning that there are base functions, other higher level functions that call these base functions, and even higher level functions that call the first level functions. It also provides the VHDL code to simulate the back end FIFO.

testruns.vhd: This file functions like a batch file enabling different combinations of lower level functions defined in the test_pkg.vhd file to be performed.

testbnch.vhd: This is the top level file which contains the port maps for the FIFO interface and the target design to the PCI bus. It runs procedures defined in the runtest.vhd file. It also has a pci_check process that reports error messages if PCI specifications are violated.

To use the testbench to test the design the following three steps need to be done from the Aldec tool:

1. Create a new project.
2. Add all the target files to that project.
3. Add the testbench files to that project.

The attribute synthesis_off statements in the pargen.vhd design file must be commented out before the simulation files can be compiled. They must, however, be uncommented before synthesizing the design.

References

For additional information on the PCI specification and how it works see the following.

- PCI Specification v2.1
- PCI System Architecture by Tom Shanley

Conclusion

This application note provides a tutorial of the PCI specification and provides documentation of the two PCI target designs that can fit into the higher density members of the Ultra37000 family of Cypress CPLDs. The flexibility, and predictable timing of the Ultra37000 family of Cypress CPLDs makes them ideal candidates for PCI bus interface applications.