

Table of Contents

1. Introduction	1
2. Getting Started	3
3. Design Methodology	3
4. LogiCore PCI Interface Operation.....	4
5. Signal Descriptions.....	5
6. Building a Complete PCI Design.....	17
7. Customizing the LogiCore PCI Interface.....	18
8. General Design Guidelines.....	25
9. Target Data Transfers and Control	28
10. Initiator Data Transfers and Control.....	29
11. Data Flow Control Signals.....	30
12. Target-Initiated Terminations	30
13. Automatic Wait-State Insertion.....	31
14. Handling Burst Transfers	32
15. Tips for Building an Initiator Controller	35
16. Controlling Initiator Transactions.....	36
17. Design Validation Process	38
18. Compliance Process	43
19. Appendix A: Pinout, Configuration	44
20. Appendix B: Resources.....	47
21. Appendix C: Waveforms.....	49

LogiCore Facts

LC-DI-PCIM-C, LC-DI-PCIS-C

Recommended Experience Level

FPGA Design ¹	Moderate to Advanced
PCI Protocol and Design	Moderate to Advanced

Resources Required

Packed CLBs	152
Occupied CLBs	268
IOBs	53+

Performance

System clock f_{max}	0-33 MHz
------------------------	----------

Supported Devices

Supported Devices	% Utilized ²
Master: XC4013E-1PQ208C	26%/47%
Slave: XC4013E-2PQ208C	

Design files

Schematics	VIEWlogic
Netlist	XNF
VHDL	Instantiate XNF netlist
Verilog	Instantiate XNF netlist

Verification

VHDL model	Generated by user
Verilog	Generated by user
Testbench	VIEWsim Command Files

Required Core Tools

XACTstep 5.2.1/6.0.1 or later

SUPPORT: XILINX WILL PROVIDE TECHNICAL SUPPORT FOR THIS LOGICORE™ PCI PRODUCT WHEN USED AS DESCRIBED IN THIS USER'S GUIDE OR SUPPORTING APPLICATION NOTES. XILINX CANNOT GUARANTEE TIMING, FUNCTIONALITY, OR SUPPORT OF THIS LOGICORE™ PCI PRODUCT IF IMPLEMENTED IN DEVICES NOT LISTED ABOVE, OR CUSTOMIZED BEYOND THAT REFERENCED IN THIS USER'S GUIDE, OR IF ANY CHANGES ARE DONE IN SECTIONS OF THE DESIGN MARKED AS "DO NOT MODIFY".

Notes

¹ Experience building high-performance, pipelined FPGA designs using Xilinx software, Floorplanner, TIMESPECs and guide files recommended.

² Packed CLBs/Occupied CLBs



1. Introduction

This design guide describes how to use the Xilinx LC-DI-PCIM-C LogiCore PCI Master interface and the Xilinx LC-DI-PCIS-C LogiCore PCI Slave interface products. Each Xilinx LogiCore PCI Interface is a fully-integrated, tested, and validated schematic-based PCI Local Bus interface design. The module enables faster implementation of production and prototype FPGA-based PCI applications. The LogiCore PCI Interface is:

- Verified for PCI local bus, revision 2.1, protocol and timing compliance.
- Optimized and pre-placed module for XC4013E FPGA.
- A general-purpose design that can be customized for specific interface requirements by the user and integrated with additional logic.
- Pre-implemented for faster time-to-volume and reduced engineering risk.

The LogiCore PCI Interface provides a high-quality foundation design. The module minimizes the engineering effort required to develop a PCI host interface—saving months of development time. This reduces risk and allows more time to focus on important system-level aspects of the design.

1.1 Overview

The LogiCore PCI Interface is a PCI interface building block created for an XC4013E FPGA device. The detailed schematics form the core PCI interface design. Further customization by the user tailors the design for specific board-level design requirements. Combining a custom user-interface with a fully-tested PCI interface produces a single-chip PCI I/O adapter.

The LogiCore PCI Interface is optimized for both the XC4000E FPGA architecture and for the XACT^{step} version 6.0.1/5.2.1 software design flow. Placement constraints, XACT-Performance timing constraints, and a placement guide file guarantee PCI timing requirements.

1.2 PCI 2.1 Compliance

The LogiCore PCI Interfaces have been extensively verified using the VirtualChips VHDL PCI bus simulation model and a Xilinx-created VIEWsim PCI protocol test-bench. Both models verify the PCI interface functions according to the test scenarios specified in the **PCI Protocol Compliance Checklist** published by the PCI Special Interest Group (PCI-SIG). The PCI test suite consists of 27 test scenarios, each designed to test compliance of a specific PCI bus protocol. Refer to **LogiCore PCI Interface Protocol Checklist** for a complete list of Xilinx test scenarios, which includes some tests beyond those specified in the PCI-SIG's checklist.

IMPORTANT!



Due to the complexity of the PCI interface, Xilinx can only guarantee PCI compliance of the LogiCore PCI Interface, as provided, and cannot provide any guarantees for user designs.

1.3 LogiCore PCI Interface Features

- Complete 32-bit PCI Interface
- PCI Local Bus Compliant - Revision 2.1
- Target-Only (Slave) or Target/Initiator (Master) PCI support
- 100% programmable single-chip solution

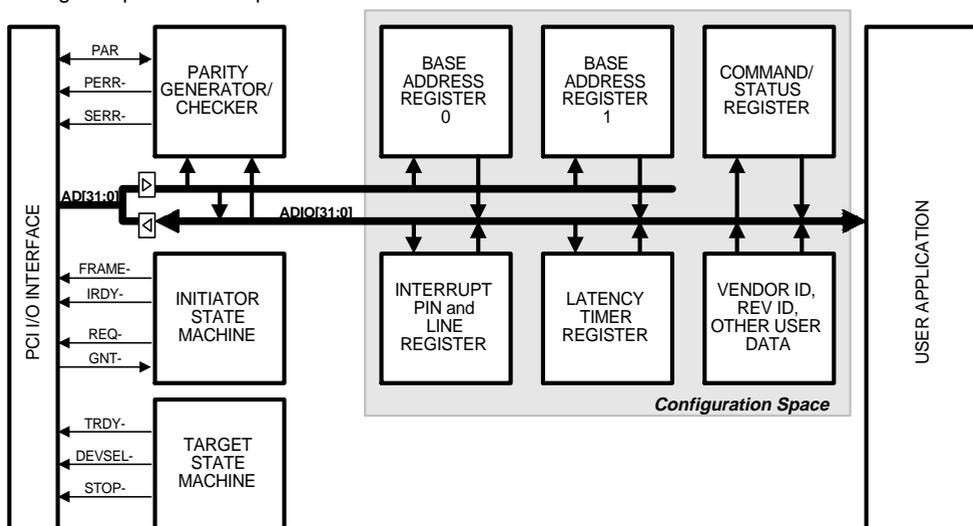


Figure 1. LogiCore PCI Interface block diagram.

1.3.1 Initiator Functions (available in LC-DI-PCIM-C only)

- Initiate Memory Read, Memory Write, Memory Read Multiple (MRM), and Memory Read Line (MRL) commands
- Initiate I/O Read and I/O Write commands
- Initiate Configuration Read and Configuration Write commands
- Bus Parking

1.3.2 Target Functions (available in both LC-DI-PCIM-C and LC-DI-PCIS-C products)

- Type 0 Configuration Space Header
- Up to 2 Base Address Registers (memory or I/O with adjustable block sizes from 16 bytes to 256 Mbytes, slow decode speed)
- Parity Generation (PAR) and Parity Error Detection (PERR# and SERR#)
- Memory Read, Memory Write, Memory Read Multiple (MRM), Memory Read Line (MRL), and Memory Write and Invalidate (MWI) command support
- I/O Read and I/O Write command support
- Configuration Read and Configuration Write command support
- 32-bit data transfers, burst transfers with linear address ordering
- Target Abort support
- Target Retry and Target Disconnect support
- Full Command/Status Register support

1.4 Functional Blocks

The LogiCore PCI Interface is partitioned into five major blocks, plus the user application, as shown in Figure 1. These functional blocks include:

1. PCI I/O Interface
2. Parity Generator/Checker
3. Target State Machine
4. Initiator State Machine
5. Configuration Space

1.4.1 PCI I/O Interface Block

The I/O interface block handles the physical connection to the PCI bus including all signaling, input and output synchronization, output three-state controls, and all request/grant handshaking for bus mastering.

1.4.2 Parity Generator/Checker

Generates/checks even parity across the AD bus, the CBE lines, and the PAR signal. Reports data parity errors via PERR- and address parity errors via SERR-.

1.4.3 Target State Machine

This block manages control over the PCI interface for Target operations. The states implemented are a subset of equations defined in "Appendix B" of the **PCI Local Bus Specification**. The controller is a high-performance state machine using state-per-bit encoding. State-per-bit encoding has narrower and shallower next-state logic functions that more closely match the Xilinx FPGA architecture.

1.4.4 Initiator State Machine (LC-DI-PCIM only)

This block manages control over the PCI interface for Initiator operations. The states implemented are a subset of equations defined in "Appendix B" of the **PCI Local Bus Specification**. The Initiator Control Logic also uses state-per-bit encoding for maximum performance.

1.4.5 PCI Configuration Space

This block provides the first 64 bytes of Type 0, version 2.1, Configuration Space Header (CSH) to support software-driven "Plug-and Play" initialization and configuration. This includes Command, Status, Latency Timer, Interrupt Pin, Interrupt Line, and two Base Address Registers (BARs), as shown in Figure 1. These BARs illustrate how to implement memory- or I/O-mapped address spaces. Each BAR sets the base address for the interface and allows the system software to determine the addressable range required by the interface. Using a combination of Configurable Logic Block (CLB) flip-flops for the read/write registers and CLB look-up tables for the read-only registers results in optimized packing density and layout.

1.4.6 User Application and Burst FIFOs

The LogiCore PCI Interface provides a simple, general-purpose interface with a 32-bit data path and latched address for de-multiplexing the PCI address/data bus. The general-purpose user interface allows the rest of the device to be used in a wide range of custom interface applications requiring programmable logic.

Typically, the user application contains burst FIFOs to increase PCI system performance. PCI derives its performance from its ability to support burst transfers. The performance of any PCI application depends largely on the burst transfer capability of the interface device. Integrated read/write FIFOs, built from the on-chip synchronous RAM available in XC4000E devices, support data transfers in excess of 33 MHz.

2. Getting Started

This document and the LogiCore PCI macro assume that the reader has

- A thorough understanding of the PCI Local Bus
- Previous Xilinx FPGA design experience, specifically with the larger members of the XC4000 FPGA family. Prior experience with XACT–Performance™ and the Floorplanner software is beneficial.
- Experience building high-performance, heavily-pipelined designs.

Xilinx *strongly* recommends that those unfamiliar with PCI obtain a copy of the **PCI Local Bus Specification** and read the reference book called **PCI System Architecture**, included with the product, before using this LogiCore design. PCI is a fairly demanding, high-performance application. Users that have never done a Xilinx design before should allow sufficient time to learn the Xilinx design environment.

To start a design using the LogiCore PCI Interface, you will need:

- **PCI Local Bus Specification**, Revision 2.1, dated June 1, 1995 or later.
- VIEWlogic schematic entry and simulation software.
- VIEWlogic libraries for the Xilinx XC4000E FPGA family.
- Xilinx XACTstep version 6.0.1/5.2.1 or later software supporting the XC4000E FPGA family.
- Optional Perl (version 5.0) software (available via the World-Wide Web, see section 20.6) to run selected utilities.
- Optional Adobe Acrobat software to view on-line documentation available on the CD-ROM. Adobe Acrobat is included with the Xilinx XACTstep software.
- A workstation or a 100-MHz or faster Pentium™ PC with at least 32 Mbytes of RAM and 10 Mbytes of available disk space to hold the design and verification files.

3. Design Methodology

The LogiCore PCI Interfaces are highly optimized with floorplanned layout for the XC4013E FPGA device. Relative placement constraints (RLOCs) provide easy implementation of the PCI interface. Pre-placed and pre-routed guide files guarantee timing performance on critical control signals like IRDY-, TRDY-, and FRAME-.

As in ASIC devices, FPGAs benefit from floorplanning, especially for designs like the LogiCore PCI Interfaces. These placement attributes help the Partition, Place and Route (PPR) tools to quickly achieve optimal routing results.

The user connects the LogiCore PCI Interface to other modules to complete the design. For example, to complete a PCI adapter card interface using the XC4013E, a designer could create a single-page schematic with the Initiator/Target LogiCore PCI Interface component together with the required user application. Next, the user compiles the design using the XACTstep software and the PCI place and route constraints file for the Xilinx XC4013E-2PQ208C. At this point the design can be simulated or downloaded to the target device.

The entire schematic describes a generic PCI interface. If possible, the logic contained in the schematic is trimmed during the design compilation process. The design can be used as is or tuned to meet a specific requirement, according to the guidelines described in this user's guide.

3.1 Modular Construction

The LogiCore PCI Interface, as provided, only supports the XC4013E-2PQ208C FPGA device. The Target-Only and Target/Initiator options require a fixed amount of CLB resources for the core PCI interface. Table 1 shows the percentage of free programmable logic space available after integrating the LogiCore PCI Interface. Note that the Target-Only (Target) design has more free space than does the Target/Initiator (Initiator) design. This is because the Initiator control logic consumes extra resources beyond those used in Target-Only designs.

Table 1. Estimated XC4013E CLB Utilization Chart for LogiCore PCI Interface Plus 16 x 32 FIFO

Device	Package	I/O	Logic Blocks	% Free CLBs	
				Target	Initiator
XC4013E	PQ208	160	576	60%	50%
	PQ240	192			

Only the XC4013E device is supported, as delivered. Table 2 presents possible alternative, unsupported solutions. None of these options has been verified through compliance testing.

Table 2. Estimated XC4000E CLB Utilization Chart for Unsupported Alternative Solutions (LogiCore PCI Interface Plus 16 x 32 FIFO)

Device	Package	I/O	Logic Blocks	% Free CLBs	
				Target*	Initiator*
XC4008E	TQ144	120	324	40%	30%
	PQ160	129			
	PQ208	144			
XC4010E	PQ160	129	400	55%	45%
	PQ208	160			
XC4020E	PQ208	160	784	70%	65%
	PQ240	193			

* Preliminary estimates

3.2 Selecting the Right Speed Grade

Fully-compliant 33 MHz PCI applications require the XC4000E speed grade shown in Table 3. The XC4013E-1 is recommended for all Master 33 MHz designs, al-

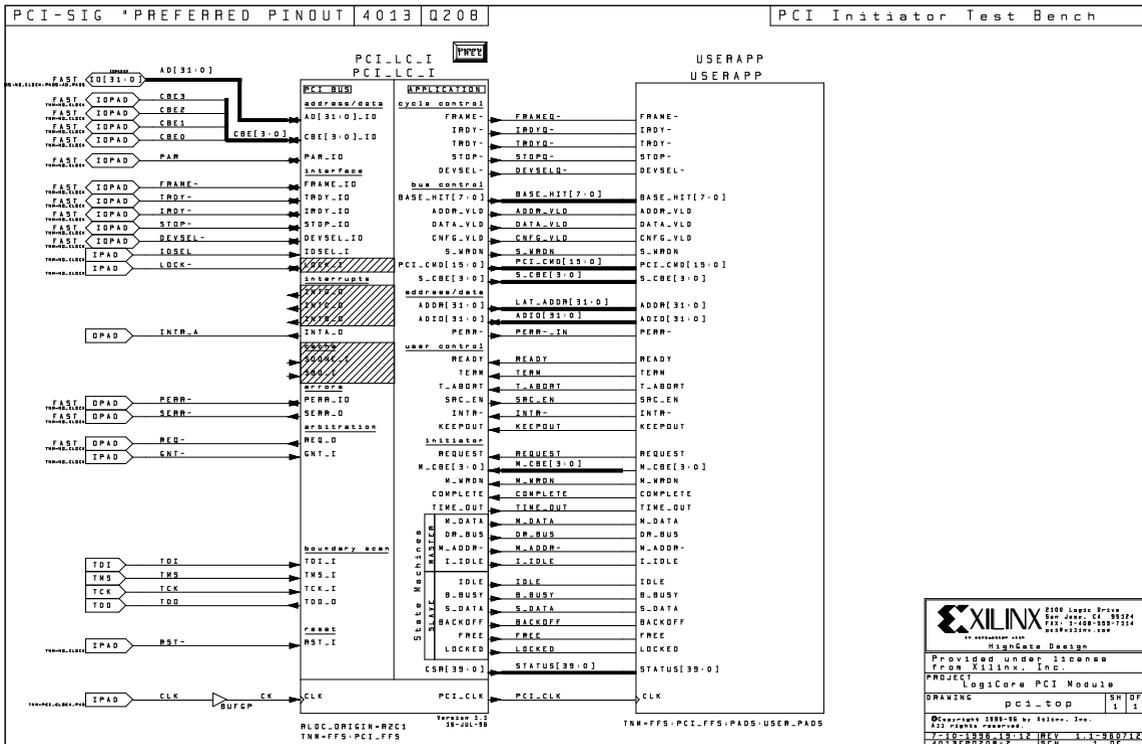


Figure 2. LogiCore PCI Schematic (pci_top.1).

though Target applications will be able to use the XC4013E-2 speed grade.

Table 3. Speed Grade Required for PCI Applications

Bus Speed	Slave	Master
25 MHz	-2	-2
33 MHz		-1

4. LogiCore PCI Interface Operation

The LogiCore PCI Interfaces are designed to interface between the PCI bus and the application interface. An example design with a PCI interface and a user application is shown in the top-level schematic block diagram (pci_top.1), shown in Figure 2. This example consists of the LogiCore PCI Interface (pci_lc_i.1). The user interface provides a complete custom interface to the user application, userapp.1. The PCI interface module, however, needs only minimal customization. The major address/data bus and a small number of control signals connect the PCI interface to the user application.

4.1 LogiCore PCI Interface (PCI_LC_I)

This section provides an operational description of the LogiCore PCI Interface (PCI_LC_I). The important features of the PCI-to-User-Interface and User-Interface-to-PCI transactions are presented in this section.

The PCI LogiCore Interface supports all the basic PCI functions including:

- Type 0 configuration space support
- I/O read/write functions
- Memory read/write functions

A summary of these basic features as well as enhancements are outlined below.

The Target design incorporates two base address registers:

- Base Register 0 (BAR0) configured for memory space with an 8-bit decode (16 Mbyte block)
- Base Register 1 (BAR1) as I/O space with a 28-bit decode (16 byte block)

These registers may be modified by the user to decode from 4 to 28 bits of address space, mapped into either memory or I/O space.

The PCI bus commands are decoded, latched, and available on the PCI_CMD[15:0] bus from the LogiCore PCI Interface. The commands supported in the design are shown in Table 4.

Full parity support for all read/write functions is largely transparent to the designer. Other embedded PCI control

functions include command decoding, latching functions, output enables, and specific PCI state-machine functions.

4.2 Supported PCI Commands

PCI bus commands direct the Target according to the type of access that the Initiator is requesting. PCI bus commands are encoded on the CBE[3:0] lines during the address phase. Table 4 illustrates the PCI bus commands currently supported by the LogiCore PCI Interface.

The LogiCore Target interface can receive and process a Memory Write and Invalidate. However, the LogiCore Initiator interface does not support the Memory Write and Invalidate command because it does not track the cache line size.

Table 4. PCI Bus Commands

CBE [3:0]	Command	PCI Master	PCI Slave
0000	Interrupt Acknowledge	No*	Ignore
0001	Special Cycle	No*	Ignore
0010	I/O Read	Yes	Yes
0011	I/O Write	Yes	Yes
0100	Reserved	Ignore	Ignore
0101	Reserved	Ignore	Ignore
0110	Memory Read	Yes	Yes
0111	Memory Write	Yes	Yes
1000	Reserved	Ignore	Ignore
1001	Reserved	Ignore	Ignore
1010	Configuration Read	Yes	Yes
1011	Configuration Write	Yes	Yes
1100	Memory Read Multiple	Yes	Yes
1101	Dual Address Cycle	No*	Ignore
1110	Memory Read Line	Yes	Yes
1111	Memory Write Invalidate	No*	Yes

* *The Initiator can present these commands. However, they either require additional user-application logic to support them or have not been thoroughly tested.*

5. Signal Descriptions

The top-level schematic block diagram (`pci_top.1`), shown in Figure 2, is a sample of a PCI function and user application.

The interface signals are grouped into functional sections with the required PCI bus interface signals on the left-hand side of the interface symbol and all user interface signals on the right-hand side of the symbol.

5.1 PCI Bus Interface Signals

Table 5 defines the interface signals that comprise the PCI Local Bus Interface. Most of the signals are common to both the PCI Target and PCI Initiator/Target modules. Pin locations are device and package dependent. See the appropriate constraint file (`*.cst`) for specific device configurations.

5.2 User Interface Signals

The user interface to the LogiCore PCI Interface provides most of the module's internal data paths and state machine control signals. This provides ultimate flexibility for customized user applications.

Table 6 describes the interface signals available on the user interface. Most of the signals are common to both the PCI Target and PCI Initiator/Target modules.

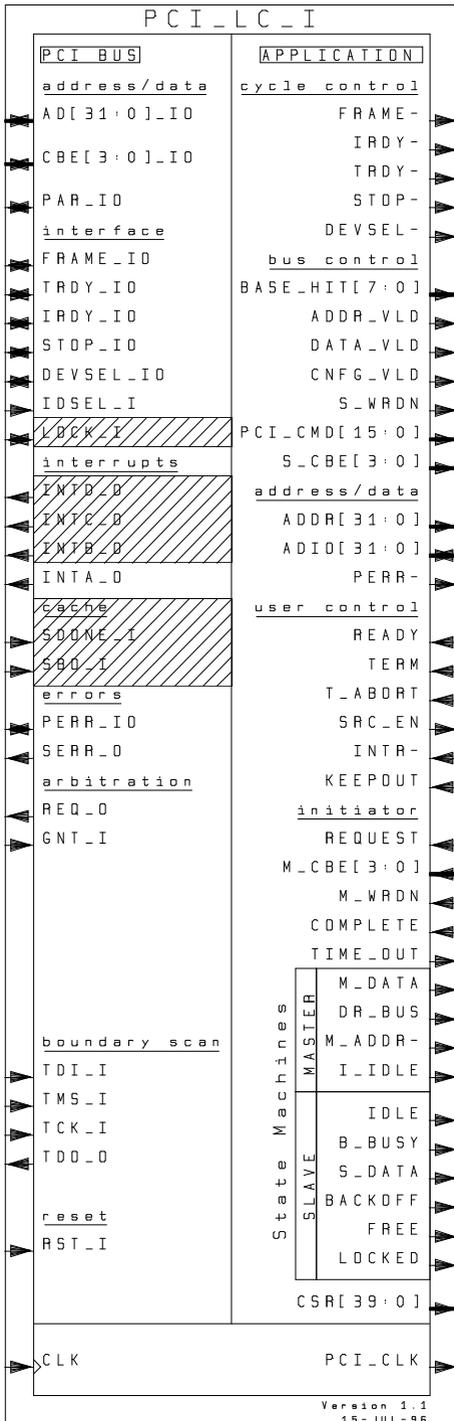


Figure 3. PCI_LC_I interface symbol.

Table 5. PCI Bus Interface Signals

Signal Name	Target	Initiator	Functional Description
Address/Data			
AD[31:0]_IO	I/O	I/O	<p>PCI Address/Data Bus - time-multiplexed address/data bus. Each bus transaction consists of an address phase followed by one or more data phases.</p> <p>The ADDR_VLD signal follows the address phase on the PCI bus indicating that the address is now available on the ADIO[31:0] internal bus. During an Initiator transaction, the back-end application should provide address on the ADIO[31:0] internal bus when M_ADDR- is asserted Low.</p> <p>The data phase is indicated by s_DATA during a Target access or by M_DATA during an Initiator operation.</p>
CBE[3:0]_IO	In	I/O	<p>PCI Command/Byte Enable - time-multiplexed bus command and byte enables. Bus commands (shown in Table 4) are asserted during an address phase on the bus. Byte enables are asserted during data phases.</p> <p>During a Target access to the LogiCore interface, the CBE[3:0]_IO lines drive the s_CBE[3:0] lines to the user application. The command presented on CBE[3:0]_IO is decoded, latched during the address cycle and presented on PCI_CMD[15:0]. Byte enables are presented during the Target data phases (s_DATA).</p> <p>When operating as an Initiator, the user application drives the CBE[3:0] lines using the m_CBE[3:0] lines on the LogiCore interface. Any command can be presented, but the user application must be able to support any command that it issues. A list of supported commands is shown in Table 4. The user interface must provide valid byte enables during the Initiator data phases (M_DATA).</p>
PAR_IO	I/O	I/O	<p>PCI Parity signal - generates/checks even parity across AD[31:0]_IO and CBE[3:0]_IO.</p> <p>When the LogiCore macro is the source of the data (Target Read, Initiator Write, or Initiator address phase), the macro generates even parity across AD[31:0]_IO and CBE[3:0]_IO and presents the result on PAR_IO one cycle after the values were presented on AD[31:0]_IO and CBE[3:0]_IO. The LogiCore macro always supplies PAR_IO when it provides data.</p> <p>When the LogiCore macro receives data (Target Write, Initiator Read, address phase presented by another agent), the macro checks for even parity across the AD[31:0]_IO and CBE[3:0]_IO presented one cycle earlier and the current PAR_IO input. Parity errors are reported via PERR_IO two cycles after data is presented and also reported by Detected Parity Error bit in the Status Register (CSR31).</p>

Signal Name	Target	Initiator	Functional Description
Interface			
FRAME_IO	In	I/O	<p>Frame - driven by the bus master to indicate a bus transaction. FRAME_IO is asserted Low for the duration of the operation and is de-asserted during the last data cycle to identify the end of the transaction.</p> <p>When operating as an Initiator, the LogiCore interface will only assert FRAME_IO after</p> <ul style="list-style-type: none"> ▪ receiving GNT_I and ▪ the bus is idle (IRDY_IO and FRAME_IO de-asserted) and ▪ the Bus Master Enable bit (CSR2) is set in the Command Register and ▪ the user application has a REQUEST pending (though the REQ_O pin need not be asserted). <p>When operating as an Initiator, the LogiCore interface will de-assert FRAME_IO after</p> <ul style="list-style-type: none"> ▪ the user application asserts COMPLETE, or ▪ receiving a termination from the addressed Target (Target Retry, Target Disconnect, or Target Abort), or ▪ <i>not</i> receiving a DEVSEL- from the addressed Target (Master Abort), or ▪ the Initiator's Latency Timer has expired, if enabled, and the system arbiter is no longer asserting GNT_I.
TRDY_IO	Out	I/O	<p>Target Ready - indicates that the target module is ready to complete the current data phase. When TRDY_IO is asserted Low, the Target is ready to transfer data.</p> <p>During a Target Read operation, the LogiCore macro automatically inserts one TRDY- wait-state after each successful data transfer, except the last (see section 13.1). This allows time for the macro to provide new data between transfers.</p> <p>During a Target transaction, TRDY- is controlled by the READY signal from the user application and the LogiCore Target state machine.</p>
IRDY_IO	In	I/O	<p>Initiator Ready - indicates that the Initiator of the access is able to complete the current data phase. When IRDY_IO is asserted Low, the Initiator is ready to transfer data.</p> <p>During an Initiator Write operation, the LogiCore macro automatically inserts one IRDY- wait-state after each successful data transfer, except the last (see section 13.1). This allows time for the macro to provide new pipelined data between transfers.</p> <p>The macro also automatically inserts one IRDY- wait-state during an Initiator burst transfer, before de-asserting FRAME_IO after the user application asserted COMPLETE and the current data transfer completes (see section 13.2). This extra wait state is not inserted on single transfers.</p> <p>During an Initiator transaction, IRDY- is controlled by the READY signal from the user application and the LogiCore Initiator state machine.</p>
STOP_IO	Out	I/O	<p>Stop - indicates that the Target has requested to stop the current access. The Target uses STOP_IO to signal a Disconnect (terminate with or without data transfer after the first transfer), Retry (terminate with no data transfer on the first transfer), or Target Abort (serious problem, no data transfer).</p> <p>The TERM signal from the user application directly controls the STOP_IO signal. When TERM is asserted, STOP_IO will be asserted on the next clock edge if the macro is involved in a Target access.</p>

Signal Name	Target	Initiator	Functional Description
DEVSEL_IO	Out	I/O	<p>Device Select - indicates that the Target has decoded the address presented during the address phase and it matches one of the Target's Base Address Registers (BARs). Address decoding is distributed within the PCI system. Each Target monitors each address cycle to determine if it is the agent addressed in the current transaction.</p> <p>The LogiCore PCI Target responds with a slow decode speed. If the presented address matches one of the macro's Base Address Registers, then the LogiCore asserts DEVSEL_IO Low on the third clock cycle after the first clock cycle where FRAME_IO is asserted Low.</p>
IDSEL_I	In	In	<p>Initialization Device Select - indicates the LogiCore macro is the target of a configuration operation. IDSEL_I is asserted High while AD[1:0]_IO='00' indicating a Configuration access.</p> <p>IDSEL_I is usually resistively coupled to one of the higher-order address lines. The specific address line will depend on the card slot's order in the system configuration chain.</p>
LOCK_I	In	I/O	<p>Lock - indicates the Initiator has gained exclusive access to a target. <i>Not Supported</i></p>
Interrupts			
INTD_O	OD	OD	<p>Interrupt D - indicates the LogiCore PCI Interface requests an interrupt. <i>Not Supported.</i></p>
INTC_O	OD	OD	<p>Interrupt C - indicates the LogiCore PCI Interface requests an interrupt. <i>Not Supported.</i></p>
INTB_O	OD	OD	<p>Interrupt B - indicates the LogiCore PCI Interface requests an interrupt. <i>Not Supported.</i></p>
INTA_O	OD	OD	<p>Interrupt A - indicates the LogiCore PCI Interface requests an interrupt. INTA_O is an open-drain output and should be driven by a flip-flop. The flip-flop is cleared by the interrupt handling routing.</p>
Cache (NOT SUPPORTED)			
SDONE_I	N/A	N/A	PCI SDONE_I signal. <i>Not Supported.</i>
SBO_I	N/A	N/A	PCI SBO_I signal. <i>Not Supported.</i>
Error Signals			
PERR_IO	Out	I/O	<p>Parity Error - indicates the module has detected a parity error as the target of a write data transfer or the initiator of a read data transfer. Checks for even parity over the AD[31:0]_IO, CBE[3:0]_IO, and PAR_IO signals. Parity errors are reported two clock cycles after the data transaction appeared on the AD[31:0]_IO and CBE[3:0]_IO lines.</p> <p>Parity error reporting on PERR_IO is enabled by setting the Report Parity Errors bit (CSR6) in the Command Register.</p> <p>Parity errors, except those during Special Cycles, are always reported in the Status Register (CSR31). Additionally, an Initiator reports that it has detected a parity error during a transaction where it was the bus master. The error is reported via the Data Parity Error Detected bit (CSR24) in the Status Register if the Report Parity Errors bit (CSR6) is set in the Command Register.</p>
SERR_O	OD	OD	<p>System Error - indicates that a parity error was detected during an address cycle. SERR_O does not report parity errors during a Special Cycle. SERR_O is asserted Low on the third clock after FRAME_IO is first recognized as asserted Low. System errors are reported on the Signaled System Error bit (CSR30) in the Status Register if the SERR- Enable bit (CSR8) is set in the Command Register. Open-drain output.</p>
Arbitration			
REQ_O	N/A	Out	<p>Request PCI Bus - indicates to the arbiter that the LogiCore PCI Initiator requests access to the bus. The Initiator may only request the bus when it has been enabled by setting the Bus Master Enable flag as bit 2 in the Command Register (CSR2). REQ_O is directly controlled by the REQUEST input from the user application.</p>

OD = Open-drain output. N/A = Not Applicable.

Signal Name	Target	Initiator	Functional Description
GNT_I	N/A	In	Grant PCI Bus - indicates that the arbiter has granted the bus to the LogiCore PCI Interface. Once GNT_I is asserted and REQUEST is asserted by the user application, the LogiCore macro performs an Initiator transaction under the conditions specified in the FRAME_IO entry. If GNT_I is asserted and there is <i>not</i> a pending REQUEST or the Bus Master Enable bit is not set, then the macro performs Bus Parking (see DR_BUS entry in Table 6).
Boundary Scan (uses XC4000E dedicated boundary scan function)			
TDI_I	In	In	Test Data Input - boundary scan serial data input.
TMS_I	In	In	Test Mode Select - boundary scan command input.
TCK_I	In	In	Test Clock - boundary scan clock input.
TDO_O	Out	Out	Test Data Output - boundary scan serial data output.
RST_I	In	In	Global Reset - resets all internal flip-flops and forces all outputs to a high-impedance state. Uses the dedicated XC4000E global set/reset and global three-state function. Resets the contents of the Command/Status Register. Disables the Initiator functionality until the system software sets the Master Enable bit in the Command Register. Disables memory or I/O Target accesses until the system software sets the Memory Enable or I/O Enable bits in the Command Register.
CLK	In	In	PCI Clock - input from PCI bus that drives the entire LogiCore module, and synchronous user applications. The frequency of CLK ranges from DC to 33 MHz. The CLK input must be driven from a primary global clock buffer (BUFGP). The buffer is not integrated in the macro to allow for easier porting to other Xilinx FPGA technologies.

N/A = Not Applicable for Target-Only function.

Table 6. Connections to the User Interface

Signal Name	Target	Initiator	Functional Description
Cycle Control			
FRAME-	Out	Out	<p>Frame - driven by the bus master to indicate the start and duration of the access. FRAME- is captured in an input flip-flop by the PCI clock.</p> <p>The Initiator only asserts FRAME- on the clock cycle following the condition where GNT- and REQ- are asserted and FRAME- and IRDY- are de-asserted on the bus (Bus Idle). Note that the Bus Master Enable bit (CSR2) in Command Register, must be set before the Initiator can request the bus.</p>
IRDY-	Out	Out	<p>Initiator Ready - driven by the bus master to indicate that it is ready to transfer data. IRDY- is captured in an input flip-flop by the PCI clock.</p>
TRDY-	Out	Out	<p>Target Ready - driven by the addressed target to indicate that it is ready to transfer data. TRDY- is captured in an input flip-flop by the PCI clock.</p>
STOP-	Out	Out	<p>Stop Transaction - driven by the addressed target to indicate that it wishes to terminate the current transaction. Data may or may not be transferred when STOP- is asserted, depending on whether TRDY- is asserted. STOP- is captured in an input flip-flop by the PCI clock.</p>
DEVSEL-	Out	Out	<p>Device Selected - driven by the addressed target to indicate that it is the target of the current transaction. DEVSEL- is captured in an input flip-flop by the PCI clock.</p>
Bus Control			
BASE_HIT[7:0]	Out	Out	<p>Base Address Hit - indicates that one of the Base Address Registers (BARs) is being addressed. The bus is one-hot encoded as indicated below. The BASE_HIT signals are active for one clock cycle, the cycle preceding the s_DATA state (which corresponds to B_BUSY if the address matches). Only BASE_HIT[1:0] are connected in the design as provided. BASE_HIT0 is pre-configured as a memory base register, BASE_HIT1 is pre-configured as an I/O base register. The base registers can be customized for specific applications as described in sections 7.4.2 through 7.4.4. (see Rev. 2.1 PCI Spec., p.187)</p> <div style="text-align: center;"> <p style="text-align: center;">BASE_HIT[7:0] Bus</p> <p style="text-align: center;">7 6 5 4 3 2 1 0</p> <p style="text-align: center;">Reserved</p> <p>Expansion ROM Address</p> <p>CardBus CIS Pointer</p> <p>Base Register 5</p> <p>Base Register 4</p> <p>Base Register 3</p> <p>Base Register 2</p> <p>Base Register 1</p> <p>Base Register 0</p> </div>
ADDR_VLD	Out	Out	<p>Address Valid - indicates the address phase on the PCI bus and that the address is available on the ADIO[31:0] internal bus.. Latched address information is captured and presented on ADDR[31:0]. The PCI bus command is latched and presented on s_CBE[3:0] and latched, decoded, and presented on PCI_CMD[15:0] during ADDR_VLD. ADDR_VLD is active during both LogiCore Target and Initiator operations but is primarily used by the user application in Target operations. (see Rev. 2.1 PCI Spec., p.246)</p>
DATA_VLD	Out	Out	<p>Data Valid - indicates that a data transaction has occurred on the PCI AD[31:0] bus. DATA_VLD is asserted High on the clock cycle after both IRDY- and TRDY- are Low on the PCI bus and either the Target state machine is in the s_DATA state or the Initiator state machine is in the M_DATA state.</p> <p>When receiving data, DATA_VLD indicates that data is available on the ADIO[31:0] bus lines. When providing data, DATA_VLD indicates that the data was received by the agent on opposite end of the transaction. See section 11.2 for more information.</p>

Signal Name	Target	Initiator	Functional Description
CNFG_VLD	Out	Out	Configuration Valid - indicates the beginning of a potential configuration cycle. Valid for a single cycle, coincident with ADDR_VLD . Does not fully decode a configuration cycle (requires PCI_CMD10 or PCI_CMD11 signals).
S_WRDN (WR_RD-IN)	Out	Out	Slave Write/Read Direction - indicates a Target Write to the user application when asserted High or a Target Read operation from the user application when asserted Low. Was called WR_RD-IN in Version 1.0.
PCI_CMD[15:0]	Out	Out	<p>PCI Bus Command - indicates the current decoded and latched PCI bus operation as defined in Table 4. The PCI_CMD[15:0] bus is a one-hot decoded representation of the CBE[3:0] bus. The command is captured during the address phase on the bus and remains asserted until the next address phase.</p> <div style="text-align: center;"> <p>PCI_CMD[15:0] Bus</p> </div>
S_CBE[3:0] (CBE_IN[3:0])	Out	Out	Slave Command/Byte Enables - indicates the PCI bus command/byte enables for a Target access to the user application. The PCI bus command appears during the address phase (also see PCI_CMD[15:0]). Byte enables are presented during the data phase. Note that byte enables are active low. Was called CBE_IN[3:0] in Version 1.0.
Address/Data			
ADDR[31:0]	Out	Out	Latched Address bus - used for pass-through data cycles and to load user address counters or DMA controllers. Address data is registered on ADDR[31:0] internally by the ADDR_VLD signal and the next rising clock edge.
ADIO[31:0]	I/O (internal 3-state)	I/O (internal 3-state)	<p>Address/Data Bus - the internal bi-directional PCI address/data bus. Used to receive data and to present both address and data information. This bus must be driven using the outputs of internal tri-state buffers (BUFTs).</p> <p><i>Note:</i> The user application should never drive the ADIO[31:0] during a Configuration Read operation. The configuration data is stored internal to the LogiCore PCI Interface. The user application should disable output enables during Configuration Read cycles by using the PCI_CMD10 signal.</p>
PERR-	In	In	Data Parity Error - indicates <i>any</i> parity errors generated on the PCI bus. PERR- is captured in an input flip-flop by the PCI clock.

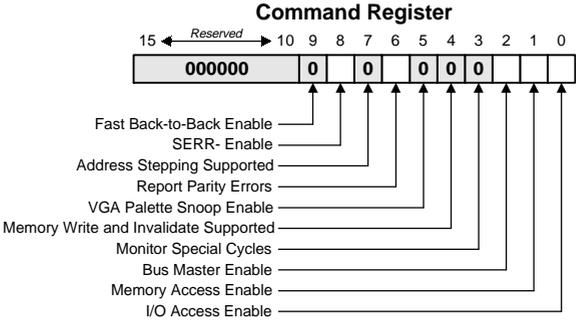
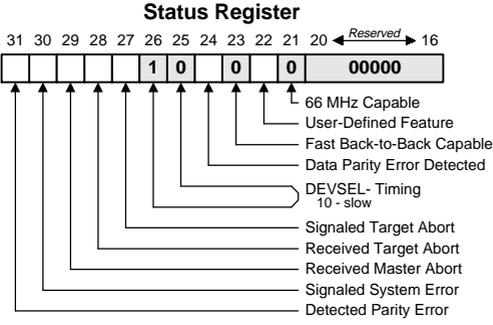
Signal Name	Target	Initiator	Functional Description
User Control			
READY	In	In	<p>Ready - signals that the user application is ready to transfer data. Used in both Target-Only and Target/Initiator applications to drive TRDY_IO during Target access and IRDY_IO during Initiator operations. If de-asserted, wait states are inserted. This signal is timing critical and should be driven from a flip-flop, if possible. (see Rev. 2.1 PCI Spec., p.238)</p> <p>Ideally, the user application should be designed so that it is always READY to send or receive data. If not, READY should be delayed until the application is ready to support a sustained burst transfer. READY cannot be delayed more than 8 clocks (PCI requirement). Avoid asserting, then de-asserting READY within a transfer, if possible, as this complicates the user application logic. See section 11.1 for more information.</p> <p>If always ready, READY should be connected to an FDPE flip-flop with the data input driven Low. This technique preserves placement and routing information contained in the guide file. See the 'testbnch' schematic test design for an example.</p>
TERM	In	In	<p>Terminate Transaction - signals the LogiCore Target interface that the user application is terminating the data flow. Causes PCI Interface to assert STOP_IO. Used in conjunction with READY to force either a Target Retry (TERM asserted, READY de-asserted on the first transfer) or a Target Disconnect (TERM asserted, READY asserted or de-asserted depend on whether data is available). See section 12 for more information. This signal is timing critical and should be driven from a flip-flop, if possible. (see Rev. 2.1 PCI Spec., p.238)</p> <p>If not used, TERM should be connected to an FDPE flip-flop with the data input driven Low. This technique preserves placement and routing information contained in the guide file. See the 'testbnch' schematic test design for an example.</p>
T_ABORT	In	In	<p>Target Abort - an optional input used to signal a serious error condition and requires the current transaction to stop. Causes the LogiCore Target interface to assert STOP_IO and de-assert DEVSEL_IO after first claiming the cycle by asserting DEVSEL_IO. No data will be transferred. (see Rev. 2.1 PCI Spec., p.238).</p> <p>If not used, T_ABORT should be tied Low.</p>
SRC_EN	Out	Out	<p>Source Data Enable - an enable signal used to increment the data pointer when the LogiCore macro is the source of data in a burst application (Target Read, Initiator Write). SRC_EN can be left unconnected in non-burst applications. Similar to DATA_VLD signal but advanced by one data transfer. Indicates when data is loaded into the output flip-flops. See section 14.2.1.</p>
INTR-	In	In	<p>Interrupt - signals an interrupt request from the user application. Active-Low. Generates an interrupt request (INTA_O) on the PCI bus. The INTR- signal should be driven by an FDPE flip-flop (preset on power-up). This flip-flop should be loaded Low to assert the INT_A signal and cleared (set) by the interrupt handling software.</p>
KEEPOUT	In	In	<p>Keep Out - isolates the internal ADIO[31:0] bus from an PCI-side accesses. This allows the user application to perform functions on the ADIO[31:0] bus without interference. Assert TERM High and de-assert READY Low to cause a Target Retry condition on the PCI bus, should another agent attempt access.</p> <p>If not used, KEEPOUT should be connected to an FDPE flip-flop with the data input driven Low. This technique preserves placement and routing information contained in the guide file. See the 'testbnch' schematic test design for an example.</p>

Signal Name	Target	Initiator	Functional Description
Initiator-Only Functions			
REQUEST	N/A	In	<p>Request (Initiator Only) - used to request a PCI Initiator transaction. Requests the PCI bus from the bus arbiter. Causes PCI Interface to assert REQ_O if the Master Enable bit (CSR2) is set in the Command Register. Should be asserted at least until the Initiator asserts M_ADDR-. Should not be kept asserted unless the user application requires long burst transfers. Do <i>not</i> keep REQUEST asserted in an attempt to force the bus arbiter to park the bus on the Initiator.</p> <p>The Bus Master Enable bit (CSR2) must be set in the Command Register before REQUEST has any affect on the PCI interface. This should be done by the system configuration software. The Initiator functionality is disabled at power-on and after RST_I is asserted.</p> <p>In most applications, REQUEST is driven by a set-dominant synchronous set/reset flip-flop. The flip-flop is set by the user application requesting the bus and reset when the LogiCore interface asserts its M_ADDR- signal.</p>
M_CBE[3:0] (CBE_OUT[3:0])	N/A	In	<p>Master Command/Byte Enables (Initiator Only) - used by the user application to drive command/byte-enables during Initiator transactions. The command should be presented during the M_ADDR- state (Initiator address phase). The byte-enables should be presented during the data phase(s) (M_DATA state). In 32-bit applications, M_CBE[3:0]='0000' during the data phase. During the data phase, the CBE lines indicate which of the byte lanes is enabled (active-Low). Was called CBE_OUT[3:0] in Version 1.0.</p> <p>Any value provided by the user application appears one clock cycle later on CBE[3:0]_IO when performing an Initiator function. The user application must be able to support any command or byte enables provided.</p>
M_WRDN (WR_RD_OUT)	N/A	In	<p>Master Write/Read Direction (Initiator Only) - driven by the user application to perform an Initiator Write operation when asserted High or an Initiator Read operation when asserted Low. Controls data flow when the user application is operating as an Initiator. Was called WR_RD_OUT in Version 1.0.</p> <p>In many applications, M_WRDN is the M_CBE0 signal captured in a flip-flop, with its clock-enable asserted while M_DATA is de-asserted. See the 'testbnch' schematic test design for an example.</p>
COMPLETE	N/A	In	<p>Complete (Initiator Only) - signals the Initiator state machine to finish the current transaction. COMPLETE is asserted on the next-to-last data transfer on a burst transaction or coincident with REQUEST on a single-cycle transfer. One data cycle after COMPLETE is received, the Initiator state machine will automatically de-asserts IRDY_IO for one cycle and then de-assert FRAME_IO coincident with again asserting IRDY_IO. COMPLETE must be asserted through the end of the data phase (until the Initiator state machine leaves the M_DATA state).</p> <p>This signal is timing critical and should be driven directly from a flip-flop if possible. There should be no more than two levels of logic generating COMPLETE. See the 'testbnch' schematic test design for an example.</p>
TIME_OUT	N/A	Out	<p>Latency Timer Timeout (Initiator Only) - indicates that the Latency Timer counter has timed out and that the user application has exceeded the maximum number of clock cycles allowed by the system configuration software. The Latency Timer is required in all Initiator applications that can burst more than two data words. If not used, the Latency Timer can be disabled by modifying the PCI_LC_I.2 schematic page. This conserves the logic consumed by the Latency Timer.</p> <p>If TIME_OUT is asserted while the system arbiter still asserts GNT-, then the operation continues until either the operation is complete, or the arbiter removes GNT-. If TIME_OUT is asserted and the system arbiter has removed GNT-, then the operation terminates prematurely. Note: The Latency Timer default value is 0, indicating immediate time-out. Be sure that the system configuration software writes a sufficiently large value in the Latency Timer Register to allow the desired transfer size.</p>

N/A = Not Applicable for Target-Only function.

Signal Name	Target	Initiator	Functional Description
State Bits			
Master (Initiator) State Machine (see PCI Spec. Rev. 2.1, page 239)			
M_DATA	N/A	Out	Indicates that the Initiator is in the data transfer state. The M_DATA (data) state occurs unconditionally after the M_ADDR- (address) is asserted.
DR_BUS	N/A	Out	Indicates that the bus is parked on the LogiCore Initiator (the system arbiter is asserting GNT_I , even though the user application is not requesting the bus). The LogiCore Initiator is then responsible for driving the AD[31:0]_IO bus, the CBE[3:0]_IO bus, and the PAR_IO signal to prevent these system bus signals from floating. The actual values driven on these lines are not important. In applications supporting Bus Parking, DR_BUS is NORed with M_ADDR- to enable the tri-state buffers (BUFTs) driving the Initiator's start address on to the internal ADIO[31:0] bus.
M_ADDR- (<i>ADDR, ADDR_BE</i>)	N/A	Out	Indicates that the Initiator is in the address state or is in bus parking. Active-Low signal . Only asserted when the Initiator has received GNT_I , the bus is idle, and the user application has asserted REQUEST . Not truly a state, but combinatorial. M_ADDR- is asserted with a one clock cycle overlap with either the I_IDLE or DR_BUS states. Used to drive the output-enable on the tri-state buffers (BUFTs) driving the Initiator's start address onto the internal ADIO[31:0] bus. The Initiator drives the AD[31:0]_IO outputs with the address presented on the ADIO[31:0] bus and enables the CBE[3:0]_IO outputs to drive the PCI bus command presented on the M_CBE[3:0] bus by the user application. The start address is presented when the Initiator state machine is in the address state. Was called ADDR or ADDR_BE in previous versions of the macro.
I_IDLE	N/A	Out	Indicates that the Initiator is in the idle state. The Initiator is either not enabled, does not have an active REQUEST pending, and has not received GNT_I from the system arbiter. In a Target-only application, the Initiator state machine is always in the I_IDLE state. In an Initiator application, the state machine will always remain in either the I_IDLE (GNT_I = High) or DR_BUS (GNT_I = Low) state when the Master Enable bit (CSR2) in the Command Register is reset (bus mastering disabled).
Slave (Target) State Machine (see PCI Spec. Rev. 2.1, page 236)			
IDLE	Out	Out	Indicates that the Target is in the idle state. There is no activity on the bus.
B_BUSY	Out	Out	Indicates that the PCI bus is busy. An agent has started a transaction (FRAME_IO has been asserted Low) but the Target state machine either has not yet finished decoding the address or has determined that it is not the target of the current operation.
S_DATA	Out	Out	Indicates that the Target is in the data transfer state. The Target has decoded the address and matched it against one of its Base Address Registers or a configuration operation is in progress. The Target has accepted the request and will respond.
BACKOFF	Out	Out	Indicates that the user application asserted TERM (STOP_IO asserted on the bus) and the Target state machine is waiting for the transaction to complete.
FREE	Out	Out	Target state machine state bit to indicate Free state. (<i>Implemented but not tested or officially supported</i>).
LOCKED	Out	Out	Target state machine state bit to indicate Locked state. (<i>Implemented but not tested or officially supported</i>).

N/A = Not Applicable for Target-Only function.

Signal Name	Target	Initiator	Functional Description
Status Output			
<p>CSR[39 : 0]</p> <p style="text-align: center;"><i>Command Register</i></p>	Out	Out	<p>Extended Command/Status - Provides the current contents of the Command/Status registers in the Configuration Space Header plus additional values to indicate the status of the current bus transaction. Shaded values indicate read-only locations.</p> <p>The Command Register values are directly set or reset through the system configuration software.</p> <p>All values in the command register, CSR[15 : 0] are either registered or read-only. The current LogiCore PCI interface does not monitor Special Cycles, does not support the Memory Write and Invalidate command, does not support palette snooping, does not support address stepping, and does not support fast back-to-back transactions. Consequently, these bits are read-only and defined a '0'.</p> <p>IMPORTANT: The Bus Master Enable bit must be set in the command register before the Initiator can access the PCI bus. The Memory Enable bit or the I/O Enable bit must be set in the command register before the Target will respond to memory or I/O commands.</p> <div style="text-align: center;"> <p>Command Register</p>  </div>
			<p>The Status Register bits are set automatically by the LogiCore PCI Interface. Individual status bits are reset by the system software by writing a '1' to the bit location to be reset.</p> <p>All values in the status register, CSR[31 : 16] are either registered or read-only.</p> <p>The DEVSEL- Timing bits in the Status Register are read-only and defined as '10' because the current implementation of the LogiCore Target interface always responds with slow decode. The Fast Back-to-Back Capable and 66 MHz Capable bits are read-only and set to '0' because the current LogiCore macro does not support either.</p> <div style="text-align: center;"> <p>Status Register</p>  </div>
<p style="text-align: center;"><i>Status Register</i></p>			

Signal Name	Target	Initiator	Functional Description
CSR[39:0] <i>Transaction Status</i>	Out	Out	<p>The Transaction Status bits are an extension of the standard Command/Status Register bits and reflect the current status of a PCI transaction. The bits are not particular to a transaction involving the user application. These status bits reflect <i>any</i> action on the bus, except for Target Signaled Abort and Master Abort, which are only signaled if the user application was involved in the transfer. CSR[37:32] are combinatorial and are only asserted during the clock cycle after the condition was true on the PCI bus.</p> <p style="text-align: center;">Transaction Status</p> <p style="text-align: center;">39 38 37 36 35 34 33 32</p> <ul style="list-style-type: none"> Valid Data Cycle End of Transaction Normal Termination Target Termination Target Signaled Retry or Disconnect without data Target Signaled Disconnect with data Target Signaled Abort Master Abort (no DEVSEL-)
User Clock			
PCI_CLK	Out	Out	<p>PCI Clock - is the PCI clock driven from via a global primary clock buffer (BUFGP) inside the FPGA device. Use this clock for all flip-flops that are synchronized the PCI system clock.</p>

6. Building a Complete PCI Design

The following list of tasks describes the development steps required to turn the LogiCore PCI Interface into a fully-functioning design integrated with your user application logic. A Target-Only design does not require any of the Initiator steps. However, an Initiator always requires the Target interface. The burst support steps may require four separate sub-steps—read and write operations for both Target and Initiator.

6.1 Target Interface (every design)

- Configure the Base Address Register(s). See Section 7.4.
- Configure the contents of the Configuration Space Header ROM. See Section 7.6.
- Build the interface to the read/write locations in the user application. See Section 9.
- Decide how to signal various target termination conditions, if required by the application. See Section 12.

6.2 Initiator Interface (Target/Initiator designs)

- Build the Initiator “mission” state machine. See Sections 15 and 16.
 - Determine what types of transactions are required. Determine the size of each transfer.
 - Decide how to handle receive various target termination conditions.

- Decide how to arbitrate between a pending Initiator request and an incoming Target transaction.
- Drive the **REQUEST** and **COMPLETE** signals.
- Build the interface to drive address values onto the **ADIO[31:0]** bus, drive commands on the **M_CBE[3:0]** bus, and set the direction of data flow on the **M_WRDN** signal. See Section 16.1.
- Build the interface to the read/write locations in the user application. See Section 10.

6.3 Burst Support (both Target and Initiator)

- Build an address counter and associated control logic. See Section 14.1.
- Providing pipelined source data and responding to various Target and Initiator termination conditions. See Section 14.2.
- Build FIFOs for the specific application, if required. See Section 14.4.
- Build **COMPLETE** logic and transfer control. See Section 16.4.

The `pci_lc_i.1` schematic, as delivered, is configured as a Target/Initiator. Configuring the macro as a Target requires changing a sub-symbol within the LogiCore PCI macro. Table 7 lists the names for the Target-Only or Target/Initiator options.

To change the Target/Initiator design (default) into a Target-Only function,

1. Open the schematic named `pci_lc_i.1`.
2. Select the symbol currently called `master`.
3. Use the VIEWlogic Powerview 'Change Component' command to replace the `master` symbol with `slave`.

`ccomp slave`

Design Note: This document refers to VIEWlogic Powerview commands. The equivalent Workview Office command is



Edit → Replace

7.2 Step 2: Disabling Latency Timer (Target/Initiator)

This step is only required if using the Target/Initiator (LC-DI-PCIM-C) product and the application will only initiate single data transfers (i.e.—never initiate burst transfers). The macro, as delivered, has the Latency Timer enabled.

For applications that do not support bursting or that burst only two words, the Latency Timer function can be disabled. This saves additional logic and routing resources.

To disable the Latency Timer,

1. Open the schematic named `pci_lc_i.1`
2. Select the symbol currently called `on` and labeled as "Latency Timer Enable."
3. Use the VIEWlogic 'Change Component' command to replace the ON symbol with OFF.

`ccomp off`

7.3 Step 3: Disabling Data Source Pipelining Logic

This step is only required if the application only supports single data transfers as either a Target-Only or a Target/Initiator application. If the user application will never transfer more than a single data word per transaction, the pipelining logic controlling the data sourcing can be turned off. This pipelining logic is enabled, by default, in the LogiCore interface.

To disable the data source pipelining logic,

1. Open the schematic named `pci_lc_i.1`
2. Select the symbol currently called `on` and labeled as "Data Source Enable."

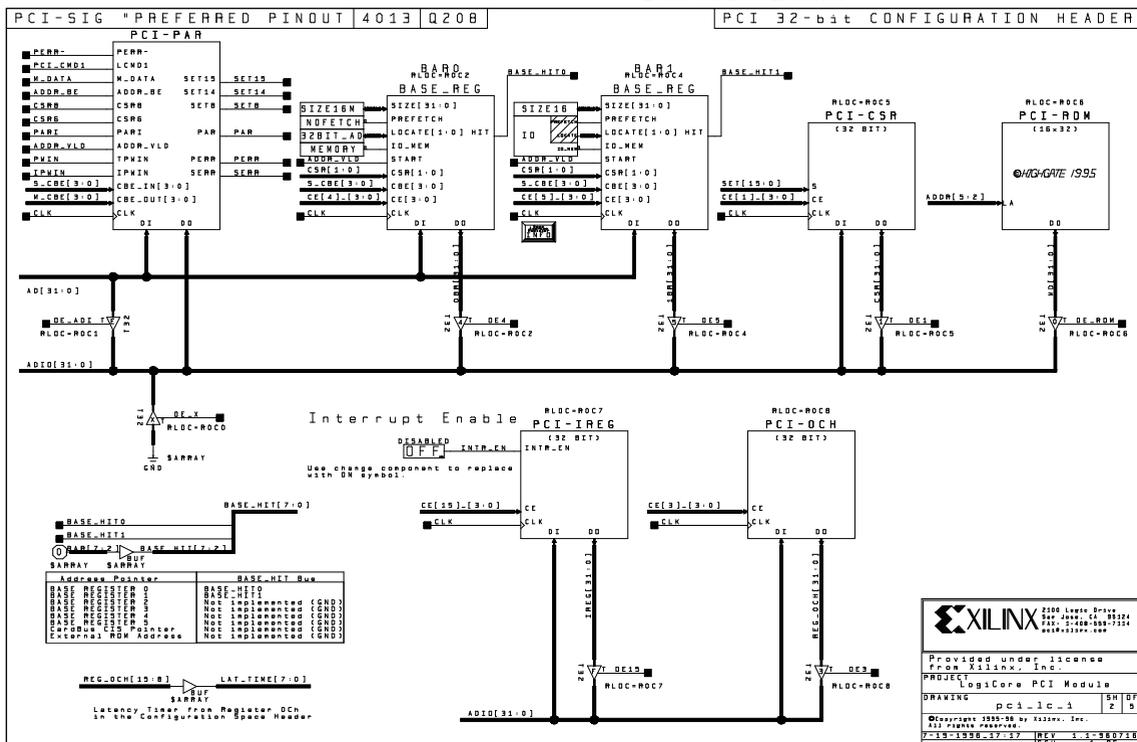


Figure 5. `PCI_LC_I.2` schematic showing the customizable Base Address Registers (`BASE_REG`) and the Configuration Space Header ROM (`PCI-ROM`) for read-only values.

3. Use the VIEWlogic 'Change Component' command to replace the ON symbol with OFF.

```
ccomp off
```

Disabling the logic permanently enables the clock-enables on the output flip-flops driving the AD, CBE, and PAR pins.

7.4 Base Address Registers

7.4.1 PCI Supports Distributed Address Decoding

There is no central address decoding resource in a PCI application. The address decoding is distributed across the various agents that make up the system. Each agent is responsible for decoding its own address. The agent requests an address range during system configuration. The system resource arbiter provides each agent with an absolute base address. This is written into a Base Address Register (BAR) contained in the agent's logic.

During the address phase of a PCI transaction, each agent monitors the address presented on the AD bus and the command presented on the CBE lines to determine whether it is the target of the transaction. If the presented address matches an address range defined in one of the agent's BARs and the command is accessing the address space (memory or I/O) defined by the BAR, then the agent will claim the transaction by asserting its DEVSEL- signal.

7.4.2 Base Address Registers in the LogiCore Interface

The LogiCore PCI design supports up to two Base Address Registers (BARs). These two registers (BAR) appear in the schematic named `pci_1c_i.2`.

A BAR supports either I/O or memory space. Memory space is preferred because it is the more flexible of the two and supports burst transfers. I/O space is supported, primarily for PC legacy support. In PCI legacy applications, I/O space is valuable. Therefore, use a memory BAR if possible.

Each BAR has several attributes. These attributes define:

- whether the address space is defined as memory or I/O. The BAR will only respond to commands that access the specified address space.
- the size of the address space block required. In the LogiCore interface, the address space can be as small as 16 bytes, or as large as 256 Mbytes.
- the ability of memory space (not I/O space) to be prefetched
- the location of memory space in total address space. This can be anywhere in 32-bit address space, anywhere in 64-bit address space, or below 1 Mbyte.

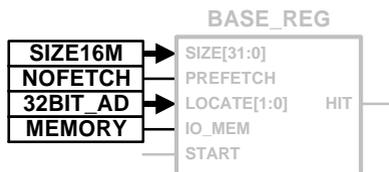


Figure 6. A Base Address Register (BAR) configured to decode a 16 Mbyte block of non-prefetchable memory, in 32-bit address space.

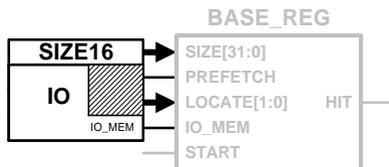


Figure 7. A Base Address Register (BAR) configured to decode a 16 byte block of I/O space.

Each of these attributes is set by replacing one or more of the symbols attached to the `BASE_REG` symbol on the `pci_1c_i.2` schematic page.

The BARs have been specially designed to match the PCI preferred pinout for the XC4013E device in the PQ208 package, so that the registers align with the internal data bus, `ADIO[31:0]`, and other logic in the FPGA. BAR0 and BAR1 are similar.

Design Note: Using a different pinout, package or device size requires modifying the relative placement constraints (RLOCs) used in this design example. Only the XC4013E device in the PQ208 package is supported.



If a design requires only a single BAR, then BAR0 must be used.

7.4.3 Customizing the Base Address Register Modes

See Section 6.2.5 in the *PCI Local Bus Specification, Revision 2.1*.

There are two different user address spaces available in a PCI application—memory and I/O. In most PC legacy systems, I/O space is rare and valuable. Consequently, memory space is preferred in most applications. However, both are supported by the PCI LogiCore macro.

Table 8. Base Address Register Modes.

Block Type	Mode Symbol
Memory	<code>memory</code>
I/O	<code>io</code>

For memory space, there are additional options available, depending on where the memory is located in address space and whether it is prefetchable or not.

Memory can be located anywhere in 32-bit address space (default), below 1 Mb (for PC legacy systems), or anywhere in 64-bit address space (for 64-bit PCI systems).

Table 9 Memory Space Location.

Description	Symbol
Located anywhere in 32-bit address space	<code>32bit_ad</code>
Located below 1MB	<code>below_1m</code>
Located anywhere in 64-bit address space	<code>64bit_ad</code>

The PCI specification defines memory as prefetchable if:

- there are no side-effects on reads (i.e.—data will not be destroyed by reading, as from a FIFO).
- byte write operations can be merged into a single double-word write, when applicable.

Table 10 Memory Type.

Description	Symbol
Prefetchable as per the PCI specification	<code>prefetch</code>
Not prefetchable	<code>nofetch</code>

Example:

Your application requires a single BAR for decoding a range of I/O addresses.

1. Look in Table 8 for the mode symbol name corresponding to I/O address space, called `io`.
2. Open the schematic named `pci_1c_i.2`.
3. Because only one BAR is required for the application, locate the `BASE_REG` symbol labeled as `BAR0`. The symbol for I/O space is larger than the symbol used to define memory space. This is done because memory space has additional options. Select, and delete the symbols attached to `LOCATE[1:0]` and `PREFETCH` ports on the `BASE_REG` symbol. Then, select the symbol labeled `MEMORY` by clicking on it with the cursor.
4. Execute the VIEWlogic 'Change Component' command to replace the `memory` symbol with `io` by typing

```
ccomp io
```

7.4.4 Customizing the Base Address Register Size

Base Address Registers (BARs) come in different sizes. This allows users to easily modify the default design for specific interface requirements. The LogiCore PCI macro supports the block sizes as indicated in Table 11.

Table 11. Default Base Address Register Sizes

Block Size	For Type	# Bits	Symbol
16	I/O	28	<code>size16</code>
32		27	<code>size32</code>
64		26	<code>size64</code>
128		25	<code>size128</code>
256		24	<code>size256</code>
512		23	<code>size512</code>
1K		22	<code>size1k</code>
2K		21	<code>size2k</code>
4K	Memory	20	<code>size4k</code>
8K		19	<code>size8k</code>
16K		18	<code>size16k</code>
32K		17	<code>size32k</code>
64K		16	<code>size64k</code>
128K		15	<code>size128k</code>
256K		14	<code>size256k</code>
512K		13	<code>size512k</code>
1M		12	<code>size1m</code>
2M		11	<code>size2m</code>
4M		10	<code>size4m</code>
8M		9	<code>size8m</code>
16M		8	<code>size16m</code>
32M		7	<code>size32m</code>
64M		6	<code>size64m</code>
128M		5	<code>size128m</code>
256M	4	<code>size256m</code>	

Generally, memory spaces below 4K in size should use a 4K block size, as recommended in the PCI specification. The maximum I/O space allowed is 256 bytes. I/O space, because it is generally so precious, must fully decode the 32-bit address.

Each block size decodes the corresponding number of bits in the address comparator. For a different base address block size, select the appropriate symbol listed and in Table 11. Then, execute a change component command with the new component.

The largest allowable I/O space is 256 bytes (not Kbytes or Mbytes). The PCI specification recommends a 4K block size for any memory space 4K or smaller.

Example:

Your application requires a 256-byte block of memory address space.

1. Look in Table 11 for the mode symbol name corresponding to a 256-byte address space. The appropriate symbol is called `size256`. However, because the application requires memory space, use a 4K block size instead (symbol `size4k`).
2. Open the schematic named `pci_1c_i.2`.

Design Note: Replace the device ID and vendor ID with your company's ID values. Do not use the values provided. A vendor ID may not be required for embedded PCI applications. Unique vendor IDs are requested through the PCI-SIG (see Appendix B: Resources)



The ROM table schematic is compilable as is. Do not move any of the sub-symbols on the schematic. The only recommended operation is a 'change component' on the HEX sub-symbols located within this table.

Each 32-bit read-only address is composed of eight nibbles. The content of each nibble is specified by changing any of the symbols to the desired value.

- 0 (hex-0) through F (hex-f) represent the 16 hexadecimal values.
- ✖ (hex-x) represents a location that is either not supported or is implemented using CLB flip-flops (read/write locations). Returns zero.

Example:

You wish to change a nibble to a hexadecimal 'F'

1. Open the schematic named `pci-rom.1`.
2. Select the desired ROM location nibble symbol.
3. Execute the VIEWlogic 'Change Component' command to change the nibble value to `hex-f`.

```
ccomp hex-f
```

The fields that should be modified are listed and described below.

7.6.1 Device ID (Location 00h, upper word)

The device ID is a unique identifier for your application. This field can be any value. In the design, as delivered, the device ID is set to 4013h to reflect that the LogiCore design is implemented in an XC4013E FPGA. Change this value for your application.

7.6.2 Vendor ID (Location 00h, lower word)

This field identifies the manufacturer of the device or application. Valid identifiers are assigned by the PCI-SIG to guarantee that each identifier is unique. As delivered, the vendor ID is set to 10EEh. This is Xilinx' vendor ID and should not be used for your application. Enter your vendor identification number here. The value FFFFh is reserved.

7.6.3 Revision ID (Location 08h, lower byte)

The revision ID indicates the revision of the device or application. It is an extension to the device ID. The value used in this design is 02h to reflect that this is the second

major revision of the LogiCore interface. Add the revision ID for your application here.

7.6.4 Class Code (Location 08h, upper 24 bits)

The class code identifies the general function of a device. It is a read-only location. The value, as provided in the default configuration, identifies the device as a generic co-processor function.

The value is divided into three byte-size fields as described in Section 6.2.1. in the PCI specification. The upper byte broadly identifies the type of function performed by the device. In this example, the class code is set to 0Bh, indicating the "Processors" class.

The middle byte defines a sub-class that more specifically identifies the device's function. The sub-classes are defined in Appendix D of the PCI specification. The example uses 40h, indicating the "Co-Processors" subclass.

The lower byte defines a specific register-level programming interface (if any). This allows device-independent software to interact with the device.

Enter the values appropriate for your application.

7.7 Command Register

The Command Register is defined as part of the Configuration Space Header but is implemented using CLB logic. Most of the Command Register values are read/write locations.

The LogiCore interface defines all 16 bits of the command register. The output of the Command Register is always provided as `CSR[15:0]` of the `CSR[39:0]` output bus.

The values in the Command Register are directly set or reset via the system software.

The Command Register bits defined in the LogiCore interface include:

- **I/O Access Enable (CSR0)** – Allows one or both of the base address registers (BARs) to access I/O space. A value of 1 enables I/O accesses. Default is 0.
- **Memory Access Enable (CSR1)** – Allows one or both of the base address registers (BARs) to access memory space. A value of 1 enables memory accesses. Default is 0.
- **Bus Master Enable (CSR2)** – Valid only for Target/Initiator applications. Allows the Initiator to generate PCI accesses. This bit must be set to 1 before any Initiator bus activity happens. Default is 0.

Design Note: The Bus Master Enable bit must be set by the system software before the LogiCore interface can perform Initiator transactions.



- **Monitor Special Cycles (CSR3)** – Special Cycle monitoring is not supported. Read-only location defined as 0.
- **Memory Write and Invalidate Enable (CSR4)** – Memory Write and Invalidate operations by the Initiator are not supported. Read-only location defined as 0.
- **VGA Palette Snoop Enable (CSR5)** – VGA palette snooping is not supported. Read-only location defined as 0.
- **Report Parity Errors (CSR6)** – Controls how the device responds to parity errors. The LogiCore macro always generates parity. Set to enable parity error reporting. Default is 0.
- **Address Stepping Supported (CSR7)** – Address stepping is not supported. Read-only location defined as 0.
- **SERR- Enable (CSR8)** – Controls how the device responds to system errors. Allows the LogiCore interface to assert the SERR- pin if an address parity error is detected. Allows the LogiCore interface to set the Signaled System Error bit (CSR30) in the Status Register. Set to enable system error reporting. Default is 0.
- **Fast Back-to-Back Enable (CSR9)** – Fast back-to-back transfers are not supported. Read-only location defined as 0.
- **Reserved (CSR[15:10])** – Reserved. Read-only locations defined as 0.
- **Fast Back-to-Back Capable (CSR23)** – The LogiCore macro, as currently supplied, does not support fast back-to-back operations. Read-only location defined as 0.
- **Data Parity Error Detected (CSR24)** – A data parity error occurred while the LogiCore interface was the initiator of a data transfer. Only set when a parity error occurs and the Report Parity Errors bit (CSR6) is set in the Command Register. Initiator-only function.
- **DEVSEL- Timing (CSR[26:25])** – Indicates how fast the LogiCore interface will decode an access to one of its Base Address Registers (BARs) or configuration accesses. The current implementation supports slow decode speed (DEVSEL- asserted on the third clock after FRAME- recognized asserted). Read-only location defined as 10b, which should not be changed.
- **Signaled Target Abort (CSR27)** – Set by the LogiCore macro when the user application asserts T_ABORT and another agent attempted a Target access.
- **Received Target Abort (CSR28)** – Set by the LogiCore Initiator when it detects that the addressed Target signaled a Target Abort condition. The user application should not retry an operation to the Target that asserted the Target Abort condition. Initiator-only function.
- **Received Master Abort (CSR29)** – Set by the LogiCore Initiator if no Target responds to a transaction that it initiated. Indicates that the addressed target is malfunctioning or non-existent. Initiator-only function.
- **Signaled System Error (CSR30)** – Set if the LogiCore interface detects a parity error during the address phase of a transaction. Only set if the SERR- Enable bit (CSR8) is set in the Command Register.
- **Detected Parity Error (CSR31)** – Set if the LogiCore interface detects a parity error and the Report Parity Errors bit (CSR6) is set in the Command Register.

7.8 Status Register

The Status Register is defined as part of the CSH but is implemented using CLB logic. Most of the values are read/write locations.

The LogiCore interface defines all 16 bits of the status register. The output of the Status Register is always provided as CSR[31:16] of the CSR[39:0] output bus.

The bits in the Status Register are set automatically by the LogiCore interface. Status bits are reset when the system software writes a '1' to a bit position in the Status Register.

The Status Register bits set or used by the LogiCore interface include:

- **Reserved (CSR[20:16])** – Reserved. Read-only locations defined as 0.
- **66 MHz Capable (CSR21)** – The LogiCore macro, as currently supplied, does not support 66 MHz applications. Read-only location defined as 0.
- **User-Defined Feature (CSR22)** – Not used in the LogiCore interface. Always reset.

7.9 Transaction Status bits (CSR[39:32])

The contents of the Command and Status registers are provided on the 40-bit bus called CSR[39:0], an output from the LogiCore PCI Interface. The lower 32 bits correspond directly to the Command/Status Register (location 04h) as defined in the Configuration Space Header. The upper 8 bits provide status on the current bus transaction.

These bits, with the exception of CSR[39:38], reflect the current status of a transaction for any bus activity—not just activity where the user application is involved. Consequently, these status bits must be qualified with other signals (*i.e.* S_DATA, M_DATA, etc.) if they are involved in the user application control logic.

- **Valid Data Cycle (CSR32)** – asserted High for the clock cycle following each valid data transfer on the

bus. Similar to `DATA_VLD`, except not specific to the user application logic.

- **End of Transaction (CSR33)** – asserted High for the clock cycle following the end of a transaction cycle (FRAME- sampled de-asserted and either TRDY- or STOP- sampled asserted).
- **Normal Termination (CSR34)** – asserted High for the clock cycle following a transaction that ended *without* a target termination condition (FRAME- and STOP- sampled de-asserted and TRDY- sampled asserted).
- **Target Termination (CSR35)** – asserted High for the clock cycle following a transaction that ended *with* a target termination condition (FRAME- sampled de-asserted and STOP- sampled asserted).
- **Target Retry or Target Disconnect without Data (CSR36)** – asserted High for the clock cycle after the target signaled a retry condition or a disconnect without data (FRAME- and TRDY- sampled de-asserted and STOP- sampled asserted). A target retry condition is only signaled on the first transfer cycle. A disconnect without data occurs on any subsequent transfer cycle.
- **Target Disconnect with Data (CSR37)** – asserted High for the clock cycle after the target signaled a disconnect with data (FRAME- sampled de-asserted and TRDY- and STOP- sampled asserted).
- **Target Signaled Abort (CSR38)** – asserted High for the clock cycle after the target signaled a Target Abort condition.
- **Master Abort (CSR39)** – asserted High on the clock cycle after the Initiator determines that the addressed target has not responded to the transaction request (DEVSEL- never asserted). Equivalent to Received Master Abort bit in the Status Register (CSR29)

8. General Design Guidelines

The following guidelines and descriptions will help simplify building the user application.

8.1 Know the Degree of Difficulty

PCI is challenging to implement in any technology and especially so in FPGA devices. Table 13 indicates the degree of difficult in implementing various types of LogiCore PCI designs.

The degree of difficulty is sharply influenced by

- the maximum system clock frequency, and
- whether the design is a Target-Only or a Target/Initiator application, and
- whether the user application supports burst transfers.

A 33 MHz application is more challenge than a 25 MHz application because there is less cycle time available for

logic levels and routing. Burst transfer support adds complexity, which typically adds layers of logic and routing. A Target/Initiator application is more challenging than a Target-Only function because of the added complexity of the Initiator state machine and control logic.

IMPORTANT!



A 33 MHz, fully-compliant Target/Initiator design requiring burst transfer support should *only* be attempted by advanced Xilinx designers or by those willing to invest the extra effort to obtain maximum bandwidth.

All PCI implementations benefit from floorplanning the user application logic. Those marked as ‘Difficult’ or ‘Advanced user only!’ *require* floorplanning. Likewise, all implementations demand careful attention to system performance requirements. Pipelining, logic mapping, floorplanning, and logic duplication are all methods that help boost system performance.

Carefully review Table 13 to determine if the LogiCore PCI interface matches your application needs.

Table 13. Degree of Difficulty to Implement Various LogiCore PCI Designs.

Function	System Clock Frequency	Burst Transfer Support	Degree of Difficulty
Target-Only	25 MHz	No	Moderate
		Yes	Moderate
	33 MHz	No	Moderate
		Yes	Difficult
Target/Initiator	25 MHz	No	Moderate
		Yes	Difficult
	33 MHz	No	Difficult
		Yes	Advanced user only!

8.2 Understanding the Signal Pipelining

In order to meet the stringent PCI performance requirements, the LogiCore interface pipelines all of the bus control signals and the data path. Consequently, some signals must be presented up to two clock cycles before they appear on the PCI bus. Likewise, arriving signals are captured and available to the user application one cycle after they appear on the PCI bus. Figure 9 provides some basic guidelines on how the LogiCore interface is pipelined.

When the user application receives a signal, it is captured in an input flip-flop to guarantee PCI’s 7 ns setup time. The signal is available to the user application one clock cycle after it appears on the PCI bus. For example, data is captured in input flip-flops and becomes available on the `ADIO[31:0]` internal bus one cycle after it appeared on the PCI bus. Signals like `ADDR_VLD` or `DATA_VLD` signal the user application to grab the value from the `ADIO[31:0]` bus.

When the user application is sending a combinatorial signal, the signal must be presented one cycle before it is to appear on the PCI bus. Most of the outputs connected to the PCI bus originate from an output flip-flop to meet PCI's 11 ns clock-to-output specification. If the signal first originates from a register in the user application, then the register inputs must be presented two cycles before the signal is to appear on the PCI bus.

8.3 Keep it Registered

The best method to simplify timing and increase system performance in an FPGA design is to keep everything registered. This means that all inputs and outputs from the user application should come from, or connect to, a flip-flop. While registering signals may not be possible for all paths, it simplifies timing analysis.

8.4 Critical Path Signals

Watch the timing and loading on the signals listed below. These signals are usually part of the critical timing path in most applications. The signals are divided into those that are in every application and those only found in Target/Initiator applications.

8.4.1 Every Application

The following list of signals are timing critical in all LogiCore PCI designs and may require special attention when connected to the user application.

- **IRDY-** — a heavily-loaded signal that connects to the Target state machine.
- **TRDY-** — a heavily-loaded signal that connects to the Initiator state machine.
- **ADDR_VLD** — a heavily-loaded signal, typically used in Target-side applications. Connects to FRAME- internally to the LogiCore macro.
- **DATA_VLD** — becomes heavily-loaded in most user applications. **DATA_VLD** is not used within the LogiCore interface. However, **DATA_VLD** uses **IRDY-**, **TRDY-**, **s_DATA**, and **m_DATA** as inputs, which are all timing-critical signals themselves. For Initiator appli-

cations, the critical path is most likely from **DATA_VLD** through the **COMPLETE** logic.

- **READY** — connects to the Target and Initiator state machine control logic through multiple layers of logic. Drive **READY** from a flip-flop, if possible. The most timing critical path is from **READY** to the **IRDY-** and **TRDY-** outputs on the PCI bus.
- **TERM** — connects to the Target state machine control logic through multiple layers of logic. Drive **TERM** from a flip-flop, if possible. The most timing critical path is from **TERM** to the **TRDY-** and **STOP-** outputs on the PCI bus.
- **KEEPOUT** — connects to the internal output-enables driving data onto the internal **ADIO[31:0]** bus. The most timing critical path is from **KEEPOUT**, through the output enable (**OE_ADI**) driving incoming data onto **ADIO[31:0]**, to any destination connected to the **ADIO[31:0]** bus.
- **s_DATA** — becomes heavily-loaded in most user applications. The LogiCore interface uses an internally-duplicated copy of **s_DATA**.

8.4.2 Target/Initiator Applications

- **FRAME-** — a heavily-loaded signal used throughout the LogiCore interface and the user application. In Target/Initiator applications, the **FRAME-** signal becomes even more heavily-loaded because it connects to the Initiator control logic. Unfortunately, **FRAME-** cannot be internally duplicated due to PCI loading restrictions.
- **COMPLETE** — is an input from the user application. It drives the **IRDY-** and **FRAME-** logic in the Initiator state machine. The critical path is usually from **DATA_VLD**, through **COMPLETE**, to setup on the **FRAME-** output on the PCI bus. Floorplanning **COMPLETE** and its associated logic helps to reduce this path.
- **M_ADDR-** — connects to the output-enables driving the Initiator's start address onto the **ADIO[31:0]** internal bus. This path has little timing margin and may require pre-placement using the Floorplanner tool. Using the XDelay timing analyzer, critical paths through **M_ADDR-** originate from the **FRAME-**, **IRDY-**, or **GNT-** signals on the PCI bus.
- **m_DATA** — becomes heavily-loaded in most user applications. The LogiCore interface uses an internally-duplicated copy of **m_DATA**.

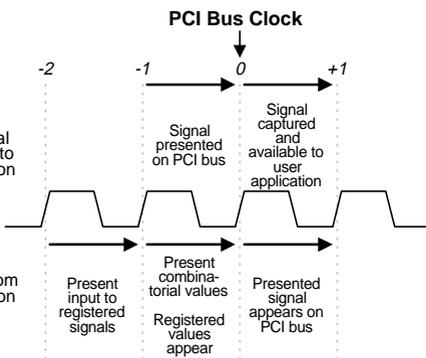


Figure 9. LogiCore PCI Pipelining.

8.5 Watch the Levels of Logic

Most of the PCI application needs to operate at 33 MHz. Some of the data paths, such as those where the interface is supplying data, are two-cycle operations executing at 16.5 MHz. In most cases, try to keep the levels of logic between flip-flops down to two layers. Three are

possible, with some care. Four or more logic layers probably will not meet 33 MHz performance.

8.6 Make Only Allowed Modifications

The LogiCore PCI interface is user-modifiable in some sections, including the number and type of Base Address Registers. The entire source schematic is provided with the product. However, do not make modifications beyond those described in this user's guide. Modifications beyond those allowed have adverse effects on system timing and PCI protocol compliance.

The design structure and the modifiable schematics are shown in Figure 10. Only the five top-level schematics and the `pci-rom.1` schematic are in the design directory. The remainder of the PCI interface design resides in a library called `LC_PCI`, which must be referenced in the `viewdraw.ini` file.

Only the first two top-level schematics (`pci_lc_i.1`, `.2`) should ever be modified in a design. The other three pages (`pci_lc_i.3`, `.4`, `.5`) are only included to make the VIEWlogic hierarchy work. They should not be modified. The other modifiable schematic is called `pci-rom.1`. It contains the ROM used to specify the read-only values in the Configuration Space Header.

The symbols for these modifiable pages point to the project directory. The remaining logic under these top-level schematics points to components in the `LC_PCI` library.

The user application, shown as `userapp` in Figure 10, contains the user application function connected to the PCI LogiCore interface. The customer user application is placed under the `userapp` hierarchy.

8.7 Place User Application Logic in USERAPP Wrapper

The LogiCore PCI interface uses a variety of advanced software features to obtain PCI system performance. These advanced features include using a "guide file" to direct the placement and routing of timing-critical logic.

The placement guide file used in the LogiCore PCI interface is a fragment of a full design. This fragment guarantees the performance of timing-critical PCI control signals in the Target and Initiator state machine logic. The guide file was carefully hand-crafted to achieve maximum performance.

Once the user application is fully integrated with the LogiCore interface, the design is final. The guide file helps the final design meet critical PCI performance requirements. The PPR placement and routing program matches logic and routing in the final PCI design to logic and routing in the hand-crafted guide file. Individual logic blocks and nets are matched by their instance name. For this reason, it is very important that the names used in the final design match those used in the guide file. If the names do not match, then PPR will not be able to guide the

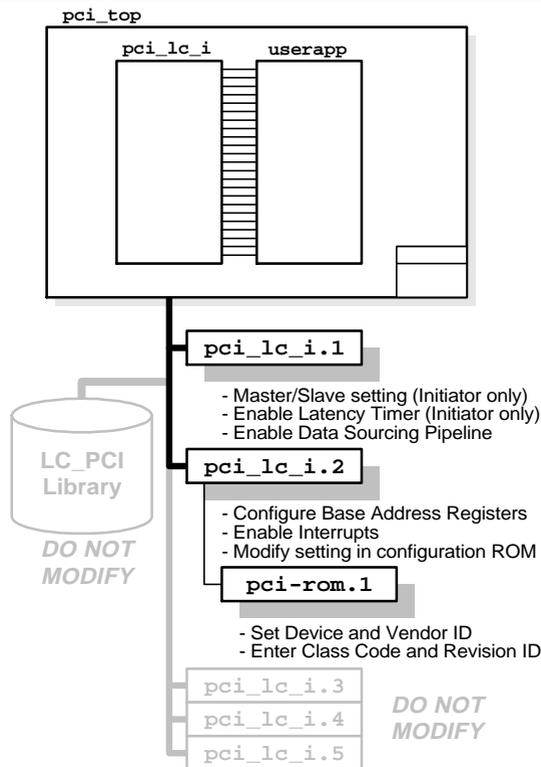


Figure 10. Design structure for LogiCore PCI interface.

full design. Consequently, PPR will not be able to achieve the required PCI performance.

To guarantee that critical instance names and net names match those used in the guide file, the custom user application should be placed in the provided "wrapper" schematic. During installation, a `/user` project directory is created. This directory contains all of the necessary, and modifiable, schematics for creating a final design.

The top-level schematic file in the `/user` directory is called `pci_top.1`. All user application logic should be placed in the design hierarchy under the `userapp.1` symbol shown in Figure 10. To add I/O pins for the user application logic, edit the right-hand side of the `userapp.1` symbol and create new I/O ports on the symbol.

All of the labels on the `pci_lc_i` symbol and all of the labels on the nets surrounding the `pci_lc_i` symbol on the `pci_top.1` schematic must remain as provided. Changing them will cause the guide file to fail.

IMPORTANT!



Do not change the symbol or net labels in the `pci_top.1` schematic. Changing these labels may cause the guide file to fail.

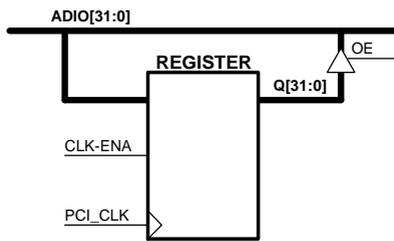


Figure 11. Example read/write register.

9. Target Data Transfers and Control

9.1 Typical Target data interface

In the majority of applications, data is transferred to and from read/write registers in the user application. These registers may also connect to internal FIFOs or to I/O pins on the user application. A typical data connection will appear something like Figure 11.

9.2 Target Write

During a Target Write operation, data is captured from the PCI bus to a data register in the user application by asserting the CLK-ENA clock enable input shown in Figure 11.

9.2.1 Interface Control Signals

The following signals control a Target Write operation.

- **BASE_HIT x =1** – indicates that one of the Base Address registers recognized that it is the target of the current operation. Use to select which set of registers are the target of the write operation.
- **S_WRDN=1** – indicates a write operation.
- **S_DATA=1** – indicates that the Target state machine is in the data transfer state.
- **DATA_VLD=1** – indicates that both the Target and the Initiator have indicated that data is available by asserting their respective TRDY- and IRDY- signals. Data is available to the user application on **ADIO[31:0]**.
- **PCI_CMD[15:0] or S_CBE[3:0]** – sometimes used to further decode a command. A memory base register responds to all memory command (Memory Write, Memory Write and Invalidate, etc.). The user application may not be able to respond to all commands.
- **ADDR_CNT[x:0]** – for burst transfers, part or all of an address counter, supplied in the user application, may be required as part of the decode logic.

9.2.2 Data Signals

- **ADIO[31:0]** – data to be written to the user application is stable and available on the internal data bus when **DATA_VLD** is asserted High.

9.2.3 Driving the CLK-ENA input to the register

The signal required to decode a Target Write operation are active and available at different times. For example, **BASE_HIT x** is active for only a single cycle, when the Base Address Register decides that it is the target of the operation. **s_WRDN** is available at the start of the address cycle and held throughout the transaction as are **PCI_CMD[15:0]** and **s_CBE[3:0]**, if required at all.

The critical gating signal is **DATA_VLD**. It is the final signal required to qualify the write operation. Consequently, the other signals can be decoded earlier and gated with **DATA_VLD**.

Part of the function would be decoded and held in a flip-flop. The equation for this decode would appear something like:

$$\begin{aligned} \text{BH}_x\text{_DEC} &:= \text{BASE_HIT}_x * \text{S_WRDN} \\ &* [\text{PCI_CMD}_x] \text{ (optional)} \\ &* [\text{ADDR_CNT}_x] \text{ (optional)} \\ &+ (!\text{S_DATA} * \text{BH}_x\text{_DEC}) \end{aligned}$$

The last term (**!S_DATA * BH x _DEC**) holds the decode asserted throughout the entire data transfer state. Again, **BASE_HIT x** is only active for a single clock cycle at the beginning of the transaction. Effectively, the above equation describes a synchronous Set/Reset flip-flop with Set dominant. The flip-flop is set when **BASE_HIT x** and associate terms are decoded and reset when **s_DATA** is Low.

Pre-decoding **BH x _DEC** allows more time for routing and reduces the number of logic levels from the critical input, **DATA_VLD**. The equation at the CLK-ENA input on the register appears as:

$$\text{CLK-ENA} := \text{BH}_x\text{_DEC} * \text{DATA_VLD}$$

9.3 Target Read

Decoding a Target Read is simpler. For this case, the application must enable the OE signal, enabling the register data onto the **ADIO[31:0]** internal bus.

9.3.1 Interface Control Signals

The following signals control a Target Read operation.

- **BASE_HIT x =1** – indicates that one of the Base Address registers recognized that it is the target of the current operation. Use to select which set of registers are the target of the read operation.
- **S_WRDN=0** – indicates a read operation.
- **S_DATA=1** – indicates that the Target state machine is in the data transfer state.
- **PCI_CMD[15:0] or S_CBE[3:0]** – sometimes used to further decode a command. A memory base register responds to all memory commands (Memory Read, Memory Read Multiple, etc.). The user application may not be able to respond to all commands.

The user application should not present data on `ADIO[31:0]` during Configuration Read operations. The `PCI_CMD10` signal is used to disable three-state buffers in the user application.

- **ADDR_CNT[x:0]** – for burst transfers, part or all of an address counter, supplied in the user application, may be required as part of the decode logic.

9.3.2 Data Signals

- **ADIO[31:0]** – data is enabled onto `ADIO[31:0]` and presented on the `AD[31:0]` pins by the LogiCore interface.

9.3.3 Driving the OE signal to present the register's data

The signals required to decode a Target Read operation are active and available at different times. For example, `BASE_HITx` is active for only a single cycle, when the Base Address Register decides that it is the target of the operation. `s_WRDN` is available at the start of the address cycle and held throughout the transaction as are `PCI_CMD[15:0]` and `s_CBE[3:0]`, if required at all.

The `DATA_VLD` is not required as part of the output enable logic. It is the responsibility of the Initiator that started the transaction to know when data transfer cycle occurred.

The output enable function would be decoded and held in a flip-flop. Note that the output enable function is active Low. Consequently, the flip-flop macro should be present on power-up and device reset. The FDPE flip-flop macro provides this capability. The equation for this function would appear something like:

```
BHx_OE := !(BASE_HITx * !S_WRDN
          * [PCI_CMDx] (optional)
          * [ADDR_CNTx] (optional)
          + (!S_DATA * !BHx_OE))
```

The last term (`!S_DATA * BHx_DEC`) holds the decode asserted throughout the entire data transfer state. Again, `BASE_HITx` is only active for a single clock cycle at the beginning of the transaction.

10. Initiator Data Transfers and Control

10.1 Typical Initiator data interface

In the majority of applications, data will be transferred to and from read/write registers in the user application. These registers may connect to internal FIFOs or may connect to I/O pins on user application. An example data connection is shown in Figure 11.

10.2 Initiator Read

During an Initiator Read operation, data is captured from the PCI bus to the data register in the user application by

asserting the `CLK-ENA` clock enable input, as shown in Figure 11.

10.2.1 Interface Control Signals

The following signals control an Initiator Read operation.

- **M_WRDN=0** – indicates a read operation.
- **M_DATA=1** – indicates that the Initiator state machine is in the data transfer state.
- **DATA_VLD=1** – indicates that both the Target and the Initiator have indicated that data is available by asserting their respective `TRDY-` or `IRDY-` signals. Data is available to the user application on `ADIO[31:0]`.

10.2.2 Data Signals

- **ADIO[31:0]** – data to be written to the user application is stable and available on the internal data bus when `DATA_VLD` is asserted High.

10.2.3 Driving the CLK-ENA input to the register

Most of the signals required to decode an Initiator Read operation are active at the start of the transaction. For example, `M_WRDN` should be provided by the user application for the duration of the transaction.

The critical gating signal is `DATA_VLD`. It is the final signal required to qualify the write operation.

```
CLK-ENA = !M_WRDN * M_DATA * DATA_VLD
```

10.3 Initiator Write

Decoding an Initiator Write is simpler. For this case, the application must enable the OE signal, enabling the register data onto the `ADIO[31:0]` internal bus.

10.3.1 Interface Control Signals

The following signals control an Initiator Write operation.

- **M_WRDN=1** – indicates a write operation
- **M_DATA=1** – indicates that the Initiator state machine is in the data transfer state

10.3.2 Data Signals

- **ADIO[31:0]** – data is enabled onto `ADIO[31:0]` and presented on the `AD[31:0]` pins by the LogiCore interface.

10.3.3 Driving the OE signal to present the register's data

The equation for an Initiator Read from an internal data source appears something like:

```
OE = !(M_WRDN * M_DATA)
```

Note, again, that the output enables for internal tri-state buffers (`BUFTs`) are active-Low.

10.4 Data Steering Control

Table 14 lists the signals required to steer data on the ADIO bus during different transactions. Note that the table refers to the flow of data to or from the PCI bus instead of using just Target Read or Initiator Write.

The various signals can be grouped together. For example, the logic to enable data onto the ADIO[31:0] bus in all cases where the LogiCore macro is providing data (PCI ← LogiCore) would appear as:

$$OE = !(!M_ADDR- + M_DATA * M_WRDN + S_DATA * !S_WRDN * !PCI_CMD10)$$

Note that the entire term is inverted because it feeds an active-Low output enable. The tri-state buffers driving the ADIO[31:0] bus (BUFTs) have active-Low output enables.

Table 14. Data Steering Control.

Direction	LogiCore as Target	LogiCore as Initiator
<i>PCI → LogiCore</i>	<i>Target Write</i>	<i>Initiator Read</i>
Address Phase	ADDR_VLD	
Data Phase	S_DATA * S_WRDN	M_DATA * !M_WRDN
<i>PCI ← LogiCore</i>	<i>Target Read</i>	<i>Initiator Write</i>
Address Phase		!M_ADDR-
Data Phase	S_DATA * !S_WRDN	M_DATA * M_WRDN

* = Logical AND operation, ! = logical inversion. Note: The user application should never drive the ADIO[31:0] bus during a Configuration Read operation.

11. Data Flow Control Signals

There are two general data flow control signals: **READY** and **DATA_VLD**. **READY** indicates that the user application is ready to perform a transaction. **DATA_VLD** indicates that a data transfer has taken place.

11.1 READY: Ready to Perform a Transaction

Asserting **READY** High tells the LogiCore PCI Interface that the user application is ready to send or receive data.

The ideal PCI agent would always be ready to accept any transaction. However, this is more difficult in practice. Some general guidelines for the **READY** signal are as follows:

- If possible, have the user application ready to respond to any transaction. This is possible in some applications, but requires careful design in the user application and requires FIFOs for burst applications.
- If the user application cannot always be ready, then delay asserting **READY** until the user application can continue without de-asserting **READY** again in the middle of the transaction. Delaying **READY** for up to eight clock cycles is allowed by the PCI specification.

Beyond that, a Target application should issue a Target Retry condition by asserting **TERM**.

- If possible, the user application should avoid asserting and de-asserting **READY** during the course of a transaction. **READY** is a timing-critical path. Toggling it during the course of a transaction makes the control logic more complicated and the system timing more difficult.
- In Initiator operations, the user application should be ready to perform the transaction before it requests the bus. However, the user application may delay asserting **READY** to arbitrate between an incoming Target access and a pending Initiator transaction. For example, the Initiator may have requested the bus but receives a Target access before the system arbiter grants the bus to the Initiator. The user application must respond appropriately.
- If possible, drive **READY** with the output of a flip-flop. This simplifies system timing.

READY is not required to be asserted during configuration transactions because the configuration logic is integrated in the macro and always available.

11.2 DATA_VLD: Valid Data Transfer

The **DATA_VLD** signal indicates that data was transferred on the PCI bus whenever:

- **IRDY-** is asserted Low, and
- **TRDY-** is asserted Low, and
- The Target state machine is in the **s_DATA** state or the Initiator state machine is in the **m_DATA** state. The Initiator state machine is not used in Target-Only applications.

During an operation where the LogiCore macro is receiving data from the PCI bus (PCI → LogiCore), **DATA_VLD** indicates that valid data is available on the ADIO[31:0] internal bus.

During an operation where the LogiCore macro is providing data to the PCI bus (PCI ← LogiCore), **DATA_VLD** indicates that data was received by the agent on the other end of the transaction.

Because **DATA_VLD** is widely used and generally involved in timing-critical paths, it is sometimes useful to duplicate the function of **DATA_VLD** in the user application. The equation for **DATA_VLD** is

$$DATA_VLD = !IRDY- * !TRDY- * (M_DATA + S_DATA)$$

12. Target-Initiated Terminations

The user application can force various Target-initiated termination conditions using the **TERM** and **READY** signals as shown in Table 15.

A Target Retry condition must be signaled by the user application before the data cycle begins (**READY** de-asserted, **TERM** asserted). This informs the Initiator that it must retry the transaction again later. The Initiator is always obliged to retry a transaction terminated with a Target Retry.

A Target Disconnect informs the Initiator that the Target is no longer able to continue the transfer (e.g., the user application FIFO is full and cannot accept any more data). The initiating agent is not required to continue the operation later. The **READY** signal is used to indicate whether the Disconnect occurs with or without data. **READY** is asserted High, if data is transferred on the Disconnect cycle.

A Disconnect without data on the first cycle is equivalent to a Retry.

Table 15. Forcing Target Termination Conditions.

Condition	Bus Signals	From User Application	
		READY	TERM
Normal	TRDY- = 0 DEVSEL- = 0 STOP- = 1	High	Low
Retry	TRDY- = 1 DEVSEL- = 0 STOP- = 0	Low	High before first cycle
Disconnect	TRDY- = X DEVSEL- = 0 STOP- = 0	High to disconnect with data	High

There is a special type of Target Terminations called Target Abort. This condition is used to signal a serious error condition from the user application. It informs the Initiator that it cannot perform the requested transaction due to various potential problems such as the Initiator attempting to burst beyond a Target's address block.

When the Target is accessed with the **T_ABORT** signal asserted from the user application, the LogiCore macro automatically signals the Target Abort condition on the bus (**DEVSEL-** asserted claiming the cycle, then **DEVSEL-** de-asserted with **STOP-** asserted). It also sets the Signaled Target Abort bit (**CSR27**) in the Status Register.

13. Automatic Wait-State Insertion

The LogiCore PCI Interface automatically inserts wait states (de-asserts **IRDY-** or **TRDY-**) under two conditions:

1. The LogiCore interface is the source of the data presented on the PCI bus during a burst transfer (i.e.—Target Read, Initiator Write).
2. The LogiCore interface is the Initiator of the transaction and is completing a burst transfer (**IRDY-** only).

13.1 Transfers where the LogiCore Interface is providing data

The LogiCore PCI Interface automatically inserts wait states (de-asserts **IRDY-** or **TRDY-**) when it is the source of data during the transaction. Consequently, the LogiCore interface can accept data at 100% burst transfer rate, but can only supply data at 50%.

Table 16. XC4000E-3 LogiCore Transfer Rates

Direction	LogiCore Operation	
	Target	Initiator
PCI → LogiCore Target Write Initiator Read	100%	100%
PCI ← LogiCore Target Read Initiator Write	50%	50%

13.1.1 Why the limitation exists

This limitation is due to the dynamics of a PCI burst transfer. The first example, Figure 12, shows the LogiCore PCI interface acting as a Target. It can accept data at 100% transfer rate. The LogiCore interface captures the data in its input flip-flops and can respond with its **TRDY-** signal at the proper time (**TRDY-** should always be asserted because, ideally, **READY** should always be asserted in the user application once the burst transfer begins).



Figure 12. Full Data Rate on PCI Write to LogiCore PCI Interface.

Contrast this with the example shown in Figure 13, a Target Read operation. Here, the LogiCore interface is the source of the data. The macro does not know if a transaction has completed until 7 ns before the next clock edge (worst-case). The Initiator may insert its own wait states (**IRDY-** de-asserted).

Assume a 100% burst transfer rate design. Once the Initiator has accepted the transaction, the LogiCore interface would need to recognize **IRDY-** asserted, decode this signal with other control logic, route the resulting signal to 36 I/O locations on three edges of the device, enable the output flip-flops.

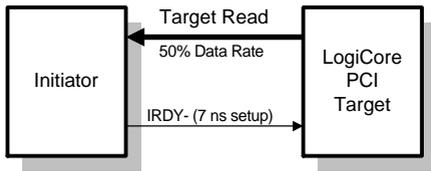


Figure 13. 50% Data Rate on PCI Read from LogiCore PCI Interface.

Meeting these requirements is difficult in the XC4013E-2 FPGA. Instead, the LogiCore interface behaves slightly differently. Once the Initiator has accepted the transaction, the LogiCore interface provides the current data (which was available on the AD bus pins) and automatically inserted a TRDY- wait state. This extra wait state allows the LogiCore interface and the user application to reliably present the next data on the AD bus pins.

13.1.2 When the wait state occurs

The automatically-inserted wait state on IRDY- and TRDY- only happens during burst transfers where the LogiCore interface is supplying data to the PCI bus (PCI ← LogiCore). It never happens on single transfers. The wait state only lasts for a single clock cycle, assuming **READY**- is asserted. The wait state is inserted on the cycle immediately following a successful data transfer.

- The IRDY- wait state only happens on Initiator Write burst transfers. See Figure 23 for an example waveform.
- The TRDY- wait state only happens on Target Read burst transfers. See the second transaction in Figure 32 for an example waveform.

13.2 Wait state when completing an Initiator transaction

The Initiator state machine automatically inserts a single IRDY- wait state at the completion of a burst data transfer. This was done to provide reliable de-assertion of the FRAME- signal. FRAME- cannot be de-asserted until the next-to-last data transfer has completed. See Figure 24 for an example waveform.

The extra IRDY- wait state is really only evident during Initiator Read burst operations. During an Initiator Write, this wait state is merged with the IRDY- wait state automatically inserted because the LogiCore macro is the source of data.

The extra IRDY- wait is not inserted on single transfers, only on bursts.

14. Handling Burst Transfers

Performing a single data transfer across the PCI bus is the simplest type of transaction. However, because of the overhead of distributed address decoding, this wastes valuable bus bandwidth. PCI's performance advantage is

in burst transactions, where two or more data words are transferred during the transaction.

Building a user application that supports single transfers is the easiest to design. Building a user application that supports burst transfers is significantly more complex, but worth the effort, if maximum bandwidth is the goal.

14.1 Keeping Track of the Address Pointer

In a PCI transaction, only the starting address is broadcast over the bus. For single transfers, this is sufficient. For burst transfers, however, the user application must keep track of the current address.

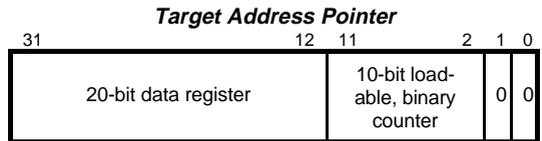
14.1.1 Target

If the Target application performs burst transfers, then it must keep a local copy of the current address pointer and increment it after every successful data transfer cycle. Luckily, this counter can be small, depending on the Target's requested address block size (set in the BAR).

The counter must be able to support bursts throughout its enter address range. For example, if the Target only decodes a 4K block of memory space, then the address counter need only keep track of addresses within the 4K block. A 10-bit loadable binary counter will suffice (4K requires 12-bits to cover the address space but bits 0 and 1 will always equal zero for 32-bit transfers). If an Initiator attempts to keep bursting past the 4K block boundary, then the Target should issue a Target Disconnect, indicating that it is not able to perform the requested operation.

The upper 20 bits of the address pointer are a simple register, loaded during the address cycle of the transaction (**ADDR_VLD** signal). Likewise, the starting address within the address block is loaded into the 10-bit binary counter during the address cycle.

The upper 20 bits, if not required in the user application, can be eliminated.



If the Target supports multiple BARs, a single address pointer should suffice, but the counter must support the largest block of address space. If one BAR supports a 4K block while the other supports a 16M block, then the counter must support the 16M block (24-bit loadable binary counter).

14.1.2 Initiator

The Initiator must also keep track of the current address pointer. During the course of a transaction, the Initiator might receive a target termination condition.

If it receives a Target Retry, the Initiator simply restarts the transaction from the starting address. However, if it receives a Target Disconnect, it may have to start the transaction somewhere in the middle of a longer burst transfer. To complicate matters, the Target can disconnect with, or without data. The Initiator must increment or hold the address pointer accordingly. The status bits that drive the decision are contained in the upper eight bits of the `CSR[39:0]` status bus (see Table 6).

The Initiator must always present and track the full 32 bits of address when starting the transaction (the lowest 2 bits are always zero, reflecting Linear Burst ordering). In most applications, however, a smaller address counter and a larger data register are sufficient to track the current address. This approach is similar to that discussed above for the Target pointer. The actual size of the counter depends on the Initiator's function. What is the largest-sized burst that the Initiator will ever perform? Will the transaction cause the address counter to roll over a major address boundary or will all of the transactions be confined to a smaller address block?

Worst-case, the Initiator may require a full 30-bit, loadable binary counter. Again, the lowest 2 bits are always zero, because the Initiator only supports Linear Burst ordering.

When implemented, the Initiator address counter/register combination should be located in CLB column 4 in the device. Likewise, it should be added to the timing constraints file provided with the LogiCore design. See the 'testbnch' schematic test design for an example.

14.2 Supplying Data in Burst Transfers

One of the difficulties in building a PCI interface in an FPGA is supplying data at full transfer rate. Receiving data at full rate is not a problem. When the interface is supplying data, however, it does not know if a transaction has completed until 7 ns before the next clock edge (worst-case). The other end of the transfer may have inserted wait states.

14.2.1 Data Pipeline Source Enable

The LogiCore interface provides burst data using a pipelined data path. The pipelined data source enable signal, `SRC_EN`, is used to advance any data pointers in the user application logic or to enable new data onto the internal

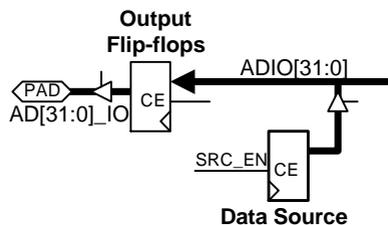


Figure 14. Data path when user application is sourcing data.

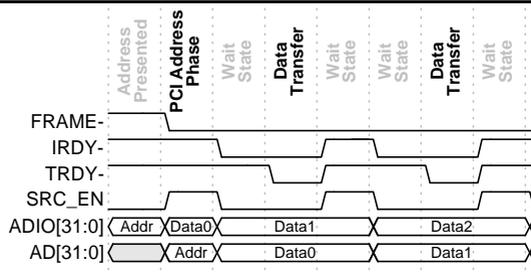


Figure 15. Burst data transfer showing `SRC_EN` signal.

`ADIO[31:0]` data bus. The data pointers may provide address to an internal FIFO, or `SRC_EN` may merely load the next data value.

Internally, the LogiCore macro captures the data value provided by the user application on the `ADIO[31:0]` bus and holds this value on the output flip-flops driving the `AD[31:0]_IO` bus pins, as shown in Figure 14. This allows the user application to present the next data value on `ADIO[31:0]`, instead of holding the previous value until the current transaction completes.

The output flip-flops driving `CBE[3:0]_IO` and `PAR_IO` are similarly controlled by the clock-enable. `PAR_IO` follows the `AD[31:0]_IO` and `CBE[3:0]_IO` busses by one clock cycle.

Figure 15 shows a burst data transfer, where the LogiCore macro is performing an Initiator Write operation (`PCI ← LogiCore`). A Target Read operation is similar.

After presenting the address value on the internal `ADIO[31:0]` during the `M_ADDR-` state, the LogiCore macro asserts the `SRC_EN` signal causing the user application to advance to the next data. The `Data0` data that was on `ADIO[31:0]` and now held in the `AD[31:0]` output flip-flops. The user application presents the next data, `Data1`, on the `ADIO[31:0]` bus. Both values are held until the next data transfer.

Immediately after the transfer, the LogiCore macro again asserts `SRC_EN`, causing the user application to provide the next data value (`Data2`) and captures the current value on the `ADIO[31:0]` bus (`Data1`) in the `AD[31:0]` output flip-flops.

This approach allows the user application to provide new data, without needing to wait for the current data transfer to complete, resulting in higher data throughput.

14.2.2 Handling Termination Conditions

Using `SRC_EN` to present the next data value on `ADIO[31:0]` or to advance any source data points does require some additional control logic. The state machine providing data from the user application must consider various termination conditions. These conditions may

require decrementing a counter or keeping a shadow copy of the previous data values.

If the source pointer is advanced to the next data location, and the data is never transferred, then the control logic must decide what to do with the non-transferred data. In some cases, the data can be discarded. For example, if the user application is providing data from an external RAM, the non-transferred data can be discarded. The original data remains in the external RAM for future use.

In other applications, reads may be destructive, such as reading from a single-port FIFO (i.e.—the data “disappears” once it is read from the FIFO). For these cases, the unused data must be restored in the data source so it is available for future use.

These termination conditions include:

- **The last data value in a Target Read burst transfer.** Because the Target does not know the size of a burst transfer, it receives the termination condition (the Initiator de-asserts FRAME-) after it has already advanced the source pointer to the next data location. The user application will need to decrement the source pointer.
- **A Target Termination condition during an Initiator Write.** The Initiator may have started a transaction that the Target cannot complete. The Target may signal some form of Target termination condition, such as Target Retry or Target Disconnect. The Initiator will have already advanced its source data pointer to the next location when it receives the termination conditions. The user application will need to decrement the source pointer, taking into account a Disconnect with, or without data.

14.3 LogiCore PCI transfer rates

The PCI bus derives its performance from its ability to support burst transfers. The performance of any PCI application depends largely on the burst transfer capability of the interface chip—not how fast it responds to or initiates a single data transfer. This is why the XC4000E’s unique on-chip, synchronous RAM capability is essential for high-performance PCI applications.

Achieving the most effective use of the PCI bus requires a careful understanding of the interaction between various system designs and the user application. PCI performance is controlled by three key factors:

1. Aggregate bandwidth,
2. Data throughput, and
3. Access latency.

The maximum theoretical bandwidth for a 33 MHz, 32-bit PCI system is 132 Mbytes per second, assuming an infinite burst transfer size. However, there is additional overhead due to the way that PCI handles transactions plus any additional overhead contributed by the actual

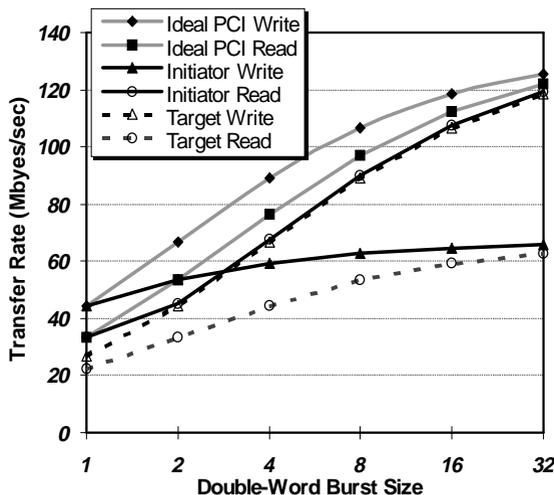


Figure 16. Effects of Read Burst Size and Latency on PCI Bandwidth.

logic implementation. Therefore, the actual system bandwidth varies tremendously from one platform to another. Modeling data throughput (bytes transferred per transfer time) as a function of access latency (delay to first data) and sustained data transfer rate (delay to subsequent data) provides insight into the various factors that limit PCI performance.

Figure 16 compares maximum data transfer between the ideal PCI read and write operations and LogiCore PCI Interface Initiator and Target transactions.

The ideal transaction assumes that both the target and the initiator operate with zero wait-states and that the target responds with fast decode speed. The ideal PCI Write transaction requires three clock cycles to first data (Idle, Address, Data) and one clock cycle for each consecutive double-word transfer (3-1-1-1). The ideal PCI Read transaction requires four clock cycles to first data (Idle, Address, turn-around, and Data) and one clock cycle for each consecutive double-word transfer (4-1-1-1).

14.3.1 LogiCore Initiator transfer rates

When acting as an Initiator, the LogiCore PCI interface performs write operations with three clocks to first data (assuming a Target with fast decode speed), but two cycles between subsequent transfers, resulting in a (3-2-2-2) transfer rate. This is shown both in Figure 16 and in Table 17. The Initiator Write transfer rate is limited by the automatically-inserted Initiator wait-states required when the LogiCore interface is providing data (see Section 13 above). During Initiator Read operations, the LogiCore interface requires four clock cycles to first data, one clock cycle between transfers, and two clock cycles for the last transfer resulting in a (4-1-1-2) transfer rate.

The LogiCore PCI interface uses slow address decoding during Target operations and asserts the TRDY- signal one clock cycle after IRDY-. These three additional clock cycles give the LogiCore PCI Interface a maximum Target Write transfer rate of (5-1-1-1) and a maximum Target Read transfer rate of (6-2-2-2) in zero wait-state environments.

Table 17. PCI Bus Transfer Rates.

Operation	Transfer Rate
Ideal PCI Write	3-1-1-1
Ideal PCI Read	4-1-1-1
Initiator Write (PCI ← LogiCore)	3-2-2-2
Initiator Read (PCI → LogiCore)	4-1-1-2
Target Write (PCI → LogiCore)	5-1-1-1
Target Read (PCI ← LogiCore)	6-2-2-2

Note that Initiator Read and Target Write operations have effectively the same bandwidth for burst transfers.

14.4 FIFOs Increase PCI Burst Bandwidth

Although the LogiCore PCI Interface requires additional clock cycles to access the PCI bus, the actual bandwidth is primarily determined by the size of the burst transfer.

FIFOs to support PCI burst transfers are efficiently implemented using the XC4000E on-chip RAM feature. Each XC4000E CLB supports two 16 x 1 RAM blocks. This corresponds to 32 bits of single-ported RAM or 16 bits of dual-ported RAM, with simultaneous read/write capability.

There are several methods to build FIFOs optimized for PCI (see “Implementing FIFOs in XC4000E RAM” application note described in Appendix B). The best structure for most applications is a dual-FIFO design with separate read and write FIFOs. Adding registers to the input and output FIFO arrays increases flexibility by supporting asynchronous user functions.

The flexible architecture of the XC4000E RAM provides for many specialized, high-performance buffering schemes, tailored to the user application.

The burst size is set by the FIFO control logic. FIFOs that are 16 double-words deep fit best into the XC4000E logic block. There is no performance or density gained by making the FIFO less than 16 locations deep. FIFOs deeper than 16 double-words but less than 33 consume more logic blocks but do not add delay. FIFOs deeper than 32 double-words consume more logic and add delay.

15. Tips for Building an Initiator Controller

This section contains a few brief statements on building the Initiator control logic in the user application.

15.1 Start with a “Mission” Statement

An effective method of building the Initiator control state machine is to write a “mission” statement for the user application. The next few questions help define some of the various design issues.

15.2 What Data Transfers are Required

Consider what data must be moved around by the Initiator.

- How big is each transfer? This affects the transfer counter size and the address counter size.
- What is the source/destination of the transfer? Is it to memory or I/O?
- How fast can the Target accept or send data?
- How fast can the Initiator provide or receive data?

15.3 What Happens if the Transaction Terminates?

Invariably, a Target will signal some form of termination condition. How will the Initiator respond? What should it do?

- The Initiator is obliged to retry an operation over again if the Target signals a Target Retry condition. Restart operation from the beginning.
- The Initiator should not retry operation if it detects a Target Abort or Master Abort condition. This indicates that the operation is either illegal for the address Target or that no target exists at the starting address.
- Handling a Disconnect condition is at the discretion of the Initiator. The Initiator is not required to retry an operation terminated with a disconnect. Also, handling a Disconnect with data is different than handling a Disconnect without data.
- The Initiator can also be terminated if the Latency Timer expires while GNT- is de-asserted. The **TIME_OUT** signal indicates when the timer expires.

15.4 Who Goes First?: Arbitrating Between an Incoming Target Access and a Pending Initiator Transaction

When the user application requests the bus for an Initiator operation, it may not actually be granted the bus for quite some time. In the meantime, another agent may initiate a target access to the user application. How should the user application respond?

Does the user application accept the target access? It may contain important information relevant to the user application’s pending Initiator transaction.

Denying the other agent access by forcing a Target Retry will be disastrous in a system with priority-based arbitration. This initiating agent may keep retrying the transaction because it may have higher priority, and the user application will never access the bus. However, in a

round-robin system, forcing a Target Retry is a good way for the Initiator to perform its pending transaction first. It can then respond to the target access when it is later retried by the other agent.

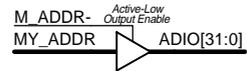
16. Controlling Initiator Transactions

Requesting the PCI bus as an Initiator is a multi-step process. Some of the timing depends on system arbitration, when the system grants access to the bus, and how fast the selected Target decodes its address. However, the setup procedure for an Initiator transfer is simple. Figure 17 demonstrates a four-word read transfer initiated by the LogiCore PCI Interface. See the example design in the `testbnch` schematics.

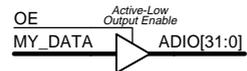
16.1 Requesting the Bus

Before actually requesting the bus by asserting the **REQUEST** signal, the user application should be set up and ready to perform the transfer.

- Provide the start address for the transaction on the input to BUFTs driving **ADIO[31:0]**



- Drive the **M_CBE[3:0]** signals with the appropriate PCI bus command (listed in Table 4). For example, assert **M_CBE[3:0]=0111b** to perform a memory write operation.
- Drive **M_WRDN** with the write or read direction control. In many applications, the **M_CBE0** signal can be captured in a flip-flop enabled while **M_DATA** is deasserted to provide **M_WRDN**.
- If performing an Initiator Write, have the data available and ready when the transaction starts. The OE logic is described in section 10.3.



- Assert **REQUEST** High to indicate that the user application requests to become a bus master (Initiator). Note: The Bus Master Enable flag must be set as bit 2 in the Command Register (CSR2) before the Initiator is able to request the bus. **REQUEST** should remain asserted until the Initiator receives **GNT-** and has asserted **FRAME-**, which is indicated by the **M_ADDR-** signal.

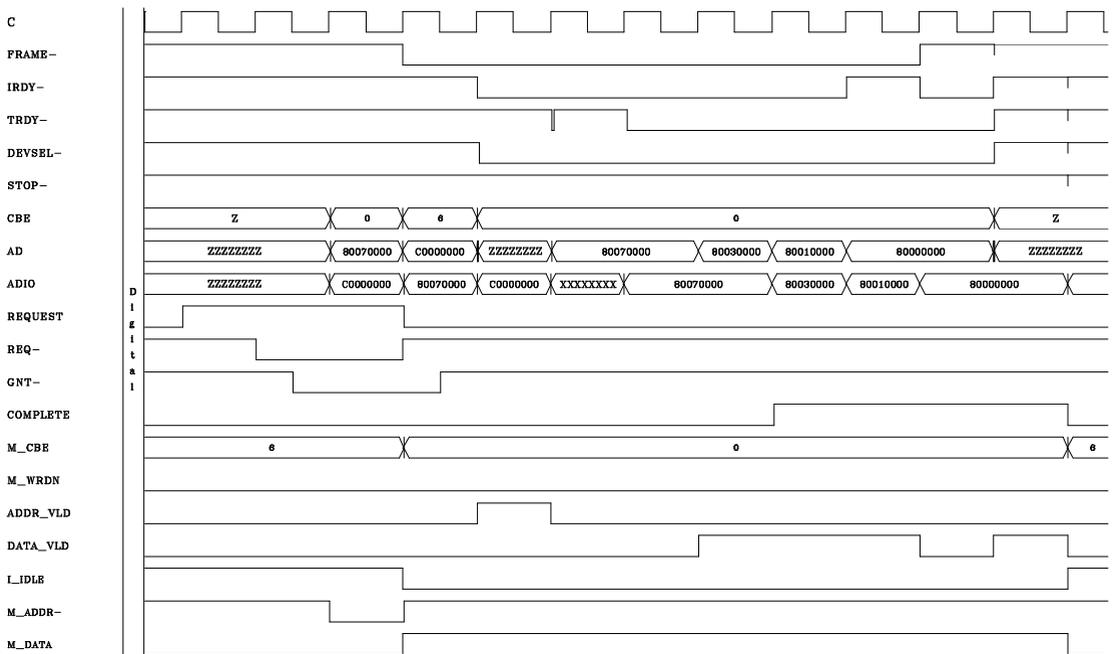


Figure 17. LogiCore Initiator Four-Word Burst Memory Read Transfer (note that the LogiCore interface automatically inserts an **IRDY-** wait-state before the last transfer in a burst transaction).

- Assert **READY** High to indicate that data is available for transfer. For many applications, **READY** is tied High because the user application is always ready to send or receive data. In some other applications, **READY** may be delayed until the bus is granted so that the user application can arbitrate between a pending Initiator transaction and an incoming Target access.

On the clock cycle after **REQUEST** is asserted, the Interface asserts its **REQ-** output Low indicating that it requests the bus. The bus will be granted after an unspecified time when the system asserts the **GNT-** input Low. Other agents are probably active on the bus.

16.2 Starting the Transaction (Address Phase)

After an unspecified time, the system arbiter asserts **GNT-** Low. The LogiCore Initiator cannot start the transaction until after it receives **GNT-**, the bus is in the idle state (**IRDY-** and **FRAME-** both de-asserted), and the Initiator state machine asserts the **M_ADDR-** signal. This signal indicates the address phase for the user application. The actual address phase on the PCI bus happens one cycle later, due to internal pipelining. Note that **M_ADDR-** overlaps with **I_IDLE** by a single clock cycle.

When **M_ADDR-** is asserted,

- Provide address information on **ADIO[31:0]** via an internal 3-state buffer (**BUFT**). **M_ADDR-**, and possibly other local select logic, is used to enable address information onto the **ADIO[31:0]** bus.

16.3 Transferring Data (Data Phase)

On the next cycle, the LogiCore Initiator asserts **FRAME-** Low, indicating the beginning of a transaction. The PCI bus command—generated by the user application on the previous cycle—appears on the **CBE[3:0]** pins. The address information presented on the **ADIO[31:0]** internal data bus now appears on the **AD[31:0]** PCI bus pins. This now starts the data phase.

- Drive the **M_CBE[3:0]** signals with the appropriate byte enables for the application. For 32-bit data bus applications, **M_CBE[3:0]=0000b**.
- Setup to send or receive data via the internal **ADIO[31:0]** bi-directional bus.
- Each valid Initiator transfer is indicated by the **DATA_VLD** signal which is asserted when the Initiator state machine is in the **M_DATA** state and both **IRDY-** and **TRDY-** are asserted Low.

16.4 Completing the Transfer (**COMPLETE**)

The user application signals the end of an Initiator transaction with the **COMPLETE** signal. Once asserted, **COMPLETE** must be held asserted through the end of the **M_DATA** state.

The time to assert **COMPLETE** depends on whether the transaction is a single or burst data transfer. See the design example shown in the **testbnch.3** schematic.

16.4.1 Single Transfers

If the Initiator is only sending or receiving a single data word, assert **COMPLETE** coincident with asserting **REQUEST**. Again, hold **COMPLETE** through the end of the **M_DATA** state.

16.4.2 Burst Transfers

- Assert **COMPLETE** High coincident with the next-to-last data transfer and hold it through the end of the **M_DATA** state. See the **testbnch.3** schematic for an example.

The logic generating **COMPLETE** can be fairly complex. As described above, single transfers can assert **COMPLETE** immediately, if there is a **REQUEST** pending. For burst transfers, **COMPLETE** is asserted during the Next-to-Last transfer. Typically, a transfer counter tracks these values. The last state is called **n**, the next-to-last is called **n-1**, etc. The actual decoding method depends on the type of transfer counter used.

The second-to-last transfer is only decoded during burst reads because Initiator Read operations occur at full burst rate. **COMPLETE** would be asserted as soon as the second-to-last transfer completes, indicated by **DATA_VLD**. Watch the timing on the **DATA_VLD** path.

The next-to-last transfer is asserted one clock cycle after entering the **M_DATA** state. This is indicated in Table 18 as **MDATAQ** (**MDATAQ := M_DATA**).

If the transfer counter is ever in the last cycle, **COMPLETE** is asserted immediately and unconditionally.

Table 18. Asserting **COMPLETE.**

Transfer Cycle	Decode	Read (<i>M_WRDN=0</i>)	Write (<i>M_WRDN=1</i>)
Last	n	REQUEST	REQUEST
Next-to-Last	n-1	MDATAQ	MDATAQ
Second-to-Last	n-2	DATA_VLD	

17. Design Validation Process

The LogiCore PCI Interface serves as a foundation for PCI designs. The interface is validated via extensive simulation. Further user testing of the LogiCore PCI Interface is accomplished using simulation or actual testing of a programmed device.

The following design example (`testinit.1`) demonstrates how to properly install and validate the LogiCore PCI Interface design environment. The design example includes a specially-built user application to help in PCI protocol testing. Each step in this four-step process introduces specific aspects of the design and assures proper operation of the complete design environment.

1. Setup and validate the design environment
2. Functional simulation
3. FPGA processing
4. Timing simulation

Completing each of the above steps with “known good” results simplifies the learning curve and validates the design environment.

Follow this step-by-step example, at least once. This helps to develop a good understanding of the tools and the overall design process. This also forms a development path that can be referenced anytime in the design process to validate PCI compliant protocol or timing. After successfully completing the design process, iterative design techniques help integrate the LogiCore PCI Interface with custom user applications.

A typical design begins by customizing the LogiCore PCI Interface (*i.e.*—selecting Target-Only or Target/Initiator, configuring the Base Address Registers, etc.) and re-running the functional simulation. Once complete, the user application is integrated with the LogiCore PCI Interface. Verifying PCI protocol compliance on an iterative basis simplifies the debug process and accelerates FPGA development time.

IMPORTANT! Due to the complexity of the PCI interface, Xilinx can only guarantee PCI compliance of the LogiCore PCI Interface as provided, and cannot provide any guarantees for user designs.



17.1 Step 1: Validate VIEWlogic Design Environment

The Release Notes provide instructions on how to install the LogiCore PCI Interface. To verify proper installation, invoke `VIEWdraw` and create a new project pointed at the example files in the `testbnch` directory. Make sure that all of the libraries are setup correctly in `viewdraw.ini` file. Then open the top-level schematic named `testinit.1`. Verify that everything is visible and appears similar to the schematic shown in Figure 3 (it will not appear exactly the same because of the special user appli-

cation). The schematic window can be left open to display logic levels during simulation.

The `testinit.1` design demonstrates the overall design structure and how to compile and test your own application. The overall design structure appears in Figure 10. Only the five top-level schematics and the PCI-ROM schematic are in the design directory. The remainder of the PCI interface design resides in a library called `LC_PCI`, which must be referenced in the `viewdraw.ini` file.

In this example design, the user application is a small function required to test PCI Initiator protocol compliance. This example application is called `testbnch.1`, `.2`, `.3`, and `.4`.

The detailed steps to validate the design environment are as follows:

1. Create and set the `VIEWlogic` project directory to point at the LogiCore PCI Interface design in the `testbnch` directory.
2. Edit the `viewdraw.ini` file to point to the XC4000E design libraries and to the `lc_pci` library containing the LogiCore PCI Interface design. An example file, `example.ini`, is included for reference.
3. Open the top-level design (`testinit.1`) and push into the first level of hierarchy.
4. Visually check the schematic. If the design appears strange or if symbols are missing (empty white boxes), recheck the `viewdraw.ini` file so that it points to the correct XC4000E and `LC_PCI` libraries.
5. Create the lower-level `.wir` files, by ‘checking’ the project.

```
check -p
```

17.2 Step 2: Validate Functional Simulation Environment

The `testinit.1` design is a PCI design for verifying functional simulation only. The `testinit.1` design is not actually compilable through the Xilinx design software, though the `pci_test.1` design is compilable. The `testinit.1` design contains a Target functional model, implemented in schematic form, to which the LogiCore Initiator performs read and write operations.

17.2.1 The PCI protocol testbench

Figure 18 shows the major components in the PCI protocol testbench design (`testinit.1`). These include the LogiCore PCI interface (`lc_pci_i`), and the Initiator testbench user application (`testbnch`). These two components are also in the compilable `pci_test` design, shown in the shaded box in Figure 10.

The Target functional model (`faketarg`) has been specially designed to respond according to the scenarios described in the **PCI-SIG Compliance Checklist**. During

Table 19. The LogiCore PCI Interface Design Process

Process Stage	Purpose or Goal	Required Inputs:
Functional Simulation	Validate design environment Validate PCI functional compliance	Reference Design: 'testinit' - \sch PCI schematic files - \sym PCI symbol files - Sample *.cmd simulation command files - *.ini files
FPGA Processing	Process reference design into an FPGA - Xdelay report validates PCI timing	Functionally-checked reference design Use 'pci_test' design - *.cst constraints file - *.lca guide file
Timing Simulation	PCI compliance timing verification	xnfb*.xnf

protocol testing, the checklist requires the Target to respond in ways that a real Target design would not—as such as forcing invalid parity, responding with various DEVSEL- decode speeds, etc.

The simple arbiter (**fakearb**) merely asserts GNT- after the LogiCore interface asserts its REQ- pin. For most of the tests, GNT- is asserted for two clock cycles, then de-asserted until the next request. When testing Bus Parking (Scenario 1.13), GNT- is over-driven by the VIEWsim command file. There is also a test in Scenario 1.14 where GNT- is only asserted for one cycle.

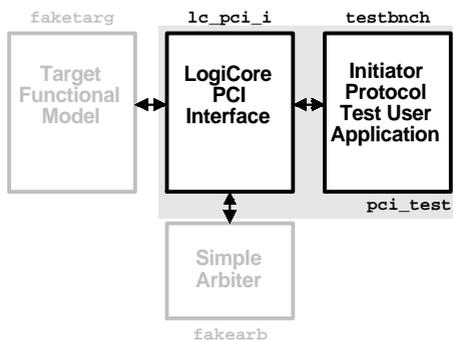


Figure 18. PCI Protocol Testbench.

Design Note: For proper simulation results, FRAME-, DEVSEL-, STOP-, IRDY-, and TRDY- are connected to PULLUP resistors model those found on a PCI motherboard. The SCENARIO and TEST buses on the schematic indicates the protocol scenario and sub-test number during simulation.



17.2.2 Create the VSM file

After successfully executing Check, invoke the VSM utility on the entire design. This creates a single .vsm file representing the entire design.

IMPORTANT!



Do not use XSimMake to create the design. When using ProFlow, do not check the “Design contains X-BLOX, RAM, ROM or X-ABEL” box.

The VSM utility should also finish without any warning or error messages. With this complete, invoke the VIEWsim simulator on the testinit design.

17.2.3 Invoke VIEWsim and execute setup command file

At the SIM> prompt, type the name of the top-level VIEWlogic simulation command file called testbnch.cmd. For design validation purposes, this command file initializes the VIEWsim interface and invokes the VIEWtrace window. This command file sets up the signals that appear in the trace window, initializes the design, and then calls other PCI cycle command files to initialize the LogiCore interface's base registers and command register, as shown in Figure 19. Table 20 summarizes the various setup and top-level command files.

The setup procedures perform the following steps:

- Setup the LogiCore PCI Interface's Command Register to enable parity error and system error responses, to enable bus mastering (Initiator functionality) and to enable the Target interface to respond to memory and I/O commands.
- Reset the LogiCore interface's status register.
- Write all ones to both Base Address Register 0 and to Base Address Register 1.
- Loads the Latency Timer Register with 255, the maximum value. This prevents the latency counter from expiring during a transaction.

For example, to run the Initiator test scenario 1.8, execute the sub-file called `ts_1_8.cmd` as shown in Table 21.

To list the Initiator test scenario command files, execute `initlist.cmd`. To list the Target test scenario command files, execute `targlist.cmd`.

17.2.6 Initiator Protocol Tests

Table 21 shows the various sub-files used in the full Initiator protocol compliance test. Test scenario 1.8 (`ts_1_8.cmd`) is a good general test of basic Initiator functionality. It performs a series of four double-word transfers while the addressed Target inserts various TRDY- wait-state patterns.

Note that the shaded table entry indicates a function that is not supported.

17.2.7 Target Protocol Tests

Table 22 shows the various supported sub-files used in the full Target protocol compliance test. Test scenario 2.13 (`ts_2_13.cmd`) is a good general test of basic Target functionality. It performs a series of three double-word transfers while the bus master (Initiator) inserts various IRDY- wait-state patterns.

The **PCI-SIG Compliance Checklist** does not contain a 2.14 scenario. This is a specially-added Xilinx test to evaluate various target termination conditions, including the `KEEPOUT` function.

Note that the shaded table entries indicate functions are not supported or tests that are not implemented.

Table 22. Supported Target Protocol Tests.

Test No.	Filename *.cmd	Test Description
2.1	<code>ts_2_1</code>	Interrupt cycles.
2.2	<code>ts_2_2</code>	Special cycles.
2.3	<code>ts_2_3</code>	Address and data parity errors during special cycles.
2.4	<code>ts_2_4</code>	Illegal byte-enables during I/O cycles.
2.5	<code>ts_2_5</code>	Target ignores reserved commands.
2.6	<code>ts_2_6</code>	Configuration cycles.
2.7	<code>ts_2_7</code>	Address and data parity errors during I/O cycles.
2.8	<code>ts_2_8</code>	Address and data parity errors during configuration cycles.
2.9	<code>ts_2_9</code>	Memory cycles.
2.10	<code>ts_2_10</code>	Address and data parity errors during memory cycles.
2.11	<code>ts_2_11</code>	Fast back-to-back transfers.
2.12	<code>ts_2_12</code>	Target Lock.
2.13	<code>ts_2_13</code>	Target responds to data transfers while Initiator inserts IRDY- wait-states.
2.14	<code>ts_2_14</code>	Xilinx-only tests to verify correct functionality of various Target termination conditions including Target Retry, Disconnect, Abort, and Keepout.

17.2.8 Command Summary

1. Create a functional simulation model for the design.


```
vsm testinit
```
2. Invoke the VIEWsim simulator


```
viewsim testinit
```
3. Execute the testbench setup command file


```
sim testbnch.cmd
```
4. Execute a PCI test vector command file


```
fulltest
```

Design Note: A waveform and text log file for a complete VIEWsim testbench compliance run is stored in the `test_out` directory on the CD-ROM. Use VIEW-Trace to view the `comply.wfm` waveform file.



17.3 Step 3: Validate FPGA Design Environment

Process the compilable design, `pci_test.1`, through the Xilinx software. XACT version 6.0.1/5.2.1 is required, along with the XC4013E speeds file (`4013e.spd`).

The protocol test design, `testinit.1`, is not compilable because it contains extra functional models (shown on `testinit.2`) that are not part of the actual FPGA design. The compilable design, `pci_test.1`, is nearly identical to `testinit.1` except that does not contain the extra functional models.

The design must be compiled with specific options to guarantee PCI performance. These options include specifying the placement and routing guide file, the constraints file, the router effort, and the placer effort. Also, the bitstream compiler (MakeBits) options must be set. The LogiCore PCI interface should use the XC4000E's fast configuration capability.

17.3.1 Compiling the `pci_test.1` design.

IMPORTANT!



PC-based users must manually set the PPR constraints file and guide file options. This is due to certain incompatibilities between the Windows-based and DOS-based design manager programs.

1. Invoke the Xilinx Design Manager (XDM).


```
xdm
```
2. Set the various compiler control options by reading the XDM profile settings. These settings instruct the XNFPREP and PPR programs to use the proper constraints file (`/cst_file/i13p208.cst` for a Target/Initiator design, `/cst_file/t13p208.cst` for a Target-only design). It also instructs PPR to use the proper guide file (`/guide/i13p208.lca` for a Target/Initiator design, `/guide/t13p208.lca` for a

Target-only design). It sets PPR's routing and placer effort controls. Furthermore, it instructs the MakeBits bitstream compiler to use the fast configuration mode available on the XC4000E and to tie unused logic and nets during the compilation process.

Profile → Readprofile

The settings defined in the `xdm.pro` profile may require modification in other designs. To view the current setting, select the following options from the XDM menus

Profile → Settings

- Run XMAKE on the `pci_test.1` design. This process will require an estimated one to four hours of computer run time, depending on the platform and the processor speed.

XMake → Done → pci_test.1 →

Then, select the desired compilation level. Selecting 'Make bistream' runs through the entire compilation process. After executing XMake the first time, you can select the `pci_test.mak` 'make' file. The 'make' file will only re-execute any necessary steps and not re-compile every file.

Design Note: During processing, any unused logic is trimmed from the design. The trimmed logic is reported in the `*.prp` report file generated by `xnfprep` during the `xmake` process. The trim report can be huge. The Perl utility called `trim` clarifies and reduces the trim report. The Perl program is required to execute the Perl utility (see Appendix B). Execute the Perl script by typing

```
perl perl/trim <in>.prp <out>.prp (Unix)
```

```
perl perl\trim <in>.prp <out>.prp (DOS)
```

17.3.2 Check for timing violations

After the design has been placed and routed, verify that all of the timing specifications have been met using the XDelay timing calculator. Check for any failed specifications.

- Create a timing report of any failed XACT-Performance timing constraints using Xdelay. Invoke the XDelay program from XDM.

Verify → Xdelay → *

- Read in the design.

Design → Design pci_test

- Create a report file of any missed performance parameters.

Misc → Report pci_top.xrp

Xdelay-TimeSpec → -FailedSpec

- Check the report file for any failed timing constraints. Occasionally, some of the timing paths may fail to

meet their specifications. Inspect the failed path to see that it is a valid path for the design. PPR and XDelay perform static timing analysis and may report "false" paths that are not actually part of the critical path. If the failed path appears to be valid, and if the path fails by a small margin, simply rerun PPR. PPR uses a pseudo-random placing algorithm and may produce better results on the next iteration.

17.4 Step 4: Validate Timing Simulation Environment

Validating timing can be accomplished using two methods. The first is using the XDelay static timing calculator. XDelay is easier and faster than validating timing through simulation.

However, timing simulation is required in some development environments. It is important to maintain separate directories for design and for timing verification. The design directory is where all changes are made using the `VIEWdraw`, `VIEWsim`, and Xilinx software.

PCI timing verification requires a separate directory structure to ensure that the tools automatically process the correct `.wir` files. Schematic files should only be temporarily placed in the timing verification directory for debug purposes.

The following list describes the correct installation and checkout procedures for the LogiCore PCI Interfaces.

Due to some I/O modeling issues, XSimMake should not be used to create the back-annotated simulation file. The following steps create the correct set of files.

- Process the `.lca` file with `lca2xnf` using the `-g` option to create an `.xnf` file.

```
lca2xnf -g pci_test.lca pci_test.xnf
```

It is important that the `.lca` file has embedded timing information, which is generated during the XMake process by default. Alternatively, you may use

```
xdelay -d -w pci_test.lca
```

which adds timing data and over-writes the `.lca` file.

- Run the `model_io` Perl script, (`model_io` version 2.01 or higher should be used). This corrects known conservative modeling values for the setup/hold and clock-to-output timing of the I/O flip-flops. The Perl script provides the guaranteed data book values in the final `.xnf` file (see the **XC4000E Technical Data** data sheet). The Perl program version 5.00 or higher is required to execute the `model_io` script, (see Appendix B: Resources).

(Unix)

```
perl perl/model_io pci_test.xnf model_io.xnf
```

(DOS)

```
perl perl\model_io pci_test.xnf model_io.xnf
```

- Run `xnfbba`. This updates the `.xnf` file generated by `model_io` with timing information.

```
xnfbba pci_test.xff model_io.xnf (cont'd)
-o pci_time.xnf
```

- Generate the required design wire files using `xnf2wir`. The output filename from this process must match your top-level design name. The resulting wire files contain the fully back-annotated timing information. This is required by `VIEWsim` to create the timing model.

```
xnf2wir pci_time wir/pci_time (Unix)
xnfbba pci_time wir\pci_time (DOS)
```

- Create the timing simulation model.

```
vsm pci_time
```

- Invoke the `VIEWsim` simulator.

```
viewsim pci_time
```

- Execute the PCI test vector command file.

```
sim testbnch.cmd
```

Note that the `testbnch.cmd` file was originally created for PCI protocol functional timing. Some of the signals that it references are not visible in a compiled, back-annotated design. `VIEWsim` will issue messages indicated that it cannot find these signals. Also, the Initiator tests will not function correctly in the compiled design because the tests require the `faketarg` Target functional model.

18. Compliance Process

PCI protocol compliance of the Xilinx Target LogiCore PCI Interface was tested according to the **PCI Compliance Checklist, Revision 2.0b** published by the PCI Special Interest Group (PCI-SIG). Additional tests were added to test functions not covered by the PCI-SIG checklist, including how the Target responds in various target termination conditions and how the Target responds to the `KEEPOUT` signal. The results are summarized in **LogiCore PCI Interface Compliance Checklist (v2.1)** available from Xilinx (see Appendix B: Resources). The LogiCore PCI Interface has been tested using the workstation-based Synopsys VSS VHDL simulator and Powerview 5.3 `VIEWsim` simulator. Equivalent PC tools are available with Workview Office 7.2 (and later). The PCI bus simulation model testbench is available from VirtualChips, (see Appendix B: Resources).

The design files for the interface between the VHDL test bench and the PCI LogiCore Interface design are available from Xilinx upon request. Send E-mail to pci@xilinx.com, with 'Request ViewLogic VHDL Interface' in the subject header.

The outputs of these compliance tests are stored in the `/test_out` sub-directory on the CD-ROM.

Table 23. Compliance test output files.

File Name	Description
<code>comply.log</code>	Log file created by the Xilinx <code>VIEWsim</code> test bench.
<code>comply.wfm</code>	<code>VIEWtrace</code> waveform file created by the <code>VIEWsim</code> test bench. View waveform using <code>VIEWtrace</code> .
<code>init_1_1.log</code>	Log file created by VirtualChips Initiator tests.
<code>targ_1_1.log</code>	Log file created by VirtualChips Target tests.

19. Appendix A: Pinout, Configuration

19.1 Layout Considerations

Xilinx devices support the PCI-SIG suggested pinout for add-in cards as well as an FPGA optimized pinout for embedded applications. The pinout shown in Table 24 follows the PCI-SIG suggested pinout and aligns the PCI data path (**ADIO[31:0]**) along the horizontal long lines in the FPGA. The user applications data bus (**D[31:0]**) also aligns with the horizontal Longlines.

The horizontal Longlines support internal 3-state busses. Various registers, such as the Base Address Registers, are aligned vertically, in columns. The schematic contains topographical representations of the XC4013E indicating how functions are placed on the device.

This pinout is developed specifically for the XC4013E device in the 208-pin PQFP package.

19.2 Pinout Table

Table 24 lists the PCI pin assignment for the 208-pin PQFP package. For each pin, both the PCI function and the fundamental device pin function are listed. Those shown in **bold italics** are dedicated pins for configuring the FPGA device using one of the serial configuration

modes. Pins without a PCI function listed are available as additional user I/O. **Note:** If there are conflicts between Table 24 and the constraints file, the constraints file has precedence.

19.3 Configuration Mode

The LogiCore PCI Interface is designed to use Serial Master Mode or Slave Mode for configuring the device. An external serial configuration PROM is required for Serial Slave Mode.

Using the XC4000E's fast configuration mode is recommended to minimize the FPGA power-up configuration time. The fast mode is set as part of the MakeBits options in the XDM profile read in during the design compilation phase.

Please refer to the **XC4000E Technical Data** data sheet for additional information.

Slave mode is useful during debugging. However, the PCI system needs to be held reset while the FPGA's bitstream is loaded.

Table 24. Example FPGA Pinout for the XC4013EPQ208 Package (pinout defined in the *.cst constraints file)

Pin No.	PCI Function	Pin Function	Pin No.	PCI Function	Pin Function	Pin No.	PCI Function	Pin Function
P1	N.C.	N.C.	P57		I/O, PGCK2	P113	USER_D2	I/O (D6)
P2	GND	GND	P58	HDC	I/O (HDC)	P114	USER_D3	I/O
P3	N.C.	N.C.	P59	CBE0	I/O	P115	USER_D4	I/O
P4	CLK	I/O, PGCK1 (A16)	P60	AD7	I/O	P116	USER_D5	I/O
P5		I/O (A17)	P61	AD6	I/O	P117	USER_D6	I/O
P6	AD23	I/O	P62	LDC-	I/O (LDC-)	P118	USER_D7	I/O
P7	AD22	I/O	P63	AD5	I/O	P119	GND	GND
P8	TDI	I/O, TDI	P64	AD4	I/O	P120	USER_D8	I/O
P9	TCK	I/O, TCK	P65	AD3	I/O	P121	USER_D9	I/O
P10	AD21	I/O	P66	AD2	I/O	P122	USER_D10	I/O (D5)
P11	AD20	I/O	P67	GND	GND	P123	USER_D11	I/O (CS0)
P12	AD19	I/O	P68	AD1	I/O	P124	USER_D12	I/O
P13	AD18	I/O	P69	AD0	I/O	P125	USER_D13	I/O
P14	GND	GND	P70		I/O	P126	USER_D14	I/O
P15	AD17	I/O	P71		I/O	P127	USER_D15	I/O
P16	AD16	I/O	P72		I/O	P128		I/O (D4)
P17	TMS	I/O, TMS	P73		I/O	P129		I/O
P18	CBE2	I/O	P74		I/O	P130	VCC	VCC
P19	GNT-	I/O	P75		I/O	P131	GND	GND
P20	FRAME-	I/O	P76		I/O	P132		I/O (D3)
P21	IRDY-	I/O	P77	INIT-	I/O (INIT-)	P133		I/O (/RS)
P22	TRDY-	I/O	P78	VCC	VCC	P134	USER_D16	I/O
P23	DEVSEL-	I/O	P79	GND	GND	P135	USER_D17	I/O
P24	STOP-	I/O	P80		I/O	P136	USER_D18	I/O
P25	GND	GND	P81		I/O	P137	USER_D19	I/O
P26	VCC	VCC	P82		I/O	P138	USER_D20	I/O (D2)
P27	LOCK-	I/O	P83		I/O	P139	USER_D21	I/O
P28	PERR-	I/O	P84		I/O	P140	USER_D22	I/O
P29	SERR-	I/O	P85		I/O	P141	USER_D23	I/O
P30	PAR	I/O	P86		I/O	P142	GND	GND
P31	REQ-	I/O	P87		I/O	P143	USER_D24	I/O
P32	CBE1	I/O	P88		I/O	P144	USER_D25	I/O
P33	AD15	I/O	P89		I/O	P145	USER_D26	I/O
P34	AD14	I/O	P90	GND	GND	P146	USER_D27	I/O
P35	AD13	I/O	P91		I/O	P147	USER_D28	I/O (D1)
P36	AD12	I/O	P92		I/O	P148	USER_D29	I/O (RCLK, RDY/BUSY)
P37	GND	GND	P93		I/O	P149	USER_D30	I/O
P38	AD11	I/O	P94		I/O	P150	USER_D31	I/O
P39	AD10	I/O	P95		I/O	P151	DIN	I/O (D0, DIN)
P40	AD9	I/O	P96		I/O	P152	DOUT	I/O, SGCK4 (DOUT)
P41	AD8	I/O	P97		I/O	P153	CCLK	CCLK
P42		I/O	P98		I/O	P154	VCC	VCC
P43		I/O	P99		I/O	P155	N.C.	N.C.
P44		I/O	P100		I/O, SGCK3	P156	N.C.	N.C.
P45		I/O	P101	GND	GND	P157	N.C.	N.C.
P46		I/O	P102	N.C.	N.C.	P158	N.C.	N.C.
P47		I/O, SCGK2	P103	DONE	DONE	P159	TDO	O, TDO
P48	M1	O (M1)	P104	N.C.	N.C.	P160	GND	GND
P49	GND	GND	P105	N.C.	N.C.	P161		I/O (A0, WS)
P50	M0	I (M0)	P106	VCC	VCC	P162		I/O, PGCK4 (A1)
P51	N.C.	N.C.	P107	N.C.	N.C.	P163		I/O
P52	N.C.	N.C.	P108	PROGRAM-	PROGRAM-	P164		I/O
P53	N.C.	N.C.	P109	RST-	I/O (D7)	P165		I/O (CS1, A2)
P54	N.C.	N.C.	P110		I/O, PGCK3	P166		I/O (A3)
P55	VCC	VCC	P111	USER_D0	I/O	P167		I/O
P56	M2	I (M2)	P112	USER_D1	I/O			

Pin No.	PCI Function	Pin Function
P168		I/O
P169		I/O
P170		I/O
P171	GND	GND
P172		I/O
P173		I/O
P174		I/O (A4)
P175		I/O (A5)
P176		I/O
P177		I/O
P178		I/O
P179		I/O
P180		I/O (A6)
P181		I/O (A7)
P182	GND	GND
P183	VCC	VCC
P184		I/O (A8)
P185		I/O (A9)
P186		I/O
P187		I/O
P188		I/O
P189		I/O
P190		I/O (A10)
P191		I/O (A11)
P192	AD31	I/O
P193	AD30	I/O
P194	GND	GND
P195	AD29	I/O
P196	AD28	I/O
P197	AD27	I/O
P198	AD26	I/O
P199	AD25	I/O (A12)
P200	AD24	I/O (A13)
P201	CBE3	I/O
P202		I/O
P203	IDSEL	I/O (A14)
P204		I/O, SGCK1 (A15)
P205	VCC	VCC
P206	N.C.	N.C.
P207	N.C.	N.C.
P208	N.C.	N.C.

20. Appendix B: Resources

Design Note: Load the file called `/web/pci.htm` on the CD-ROM into your Internet browser for easy access to the latest PCI information referenced below.



20.1 PCI Special Interest Group (PCI-SIG) Publications

The PCI-SIG publishes various PCI specifications and related documents. Most publications cost US\$25 plus applicable shipping charges.

- **PCI Local Bus Specification**, Rev. 2.1
- **PCI Compliance Checklist** v2.0b and unratified v2.1 draft (available via the World-Wide Web)
- **PCI System Design Guide** v1.0

Contact:

PCI Special Interest Group
 2575 NE Kathryn St. #17
 Hillsboro, OR 97214
 TEL: 1-800-433-5177 (within USA)
 TEL: 1-503-693-6232 (worldwide)
 FAX: 1-503-693-8344
 E-Mail: info@pcisig.com
 WEB: <http://www.pcisig.com>

20.2 PCI and FPGA Design Consultants

HighGate Design implemented a large portion of the LogiCore PCI Target interface. Their extensive Xilinx FPGA design experience helped the LogiCore PCI Interface meet the stringent PCI timing requirements.

Contact:

HighGate Design, Inc.
 12380 Saratoga/Sunnyvale Road
 Suite 8
 Saratoga, CA 95070
 TEL: 1-408-255-7160
 FAX: 1-408-255-7162
 E-mail: highgate@highgatedesign.com

20.3 PCI Bus Simulation Model

The VirtualChips PCI bus test environment used for compliance testing is provided by Phoenix Technologies.

Contact:

VirtualChips (Phoenix Technologies)
 2107 N. First St., Suite 100
 San Jose, California 95131
 TEL: 1-888-4V CHIPS (1-888-482-4477)
 FAX: 1-408-452-0952
 E-mail: sales@vchips.com
 WEB: <http://www.vchips.com>

20.4 PCI Reference Books

There are many reference books available on PCI. The following are a few that the product development team found useful.

- **PCI System Architecture** by Tom Shanley and Don Anderson. ISBN 1-881609-08-1. An excellent general reference book on PCI. **This book is included with the LogiCore PCI product.**

Contact:

Mindshare Press
 2202 Buttercup Dr.
 Richardson, TX 75082
 TEL: 1-214-231-2216
 FAX: 1-214-783-4715

Distributed by:

Computer Literacy Bookshops, Inc.
 P.O. Box 641897
 San Jose, CA 95164
 TEL: 1-408-435-0744
 FAX: 1-408-435-1823
 E-mail: info@clbooks.com
 WEB: <http://www.clbooks.com>

- **PCI Hardware and Software Architecture & Design** by Edward Solari & George Willse. ISBN 0-929392-19-1. Everything that you ever wanted to know about PCI systems design, and more.

Contact:

Annabooks
 11848 Bernardo Center Drive
 Suite 110
 San Diego, CA 92128
 TEL: 1-619-673-0870
 1-800-462-1042
 FAX: 1-619-673-1432

20.5 Xilinx Documents

The following documents are available in Adobe Acrobat format in the `/docs` sub-directory on the CD-ROM.

- **LogiCore PCI Interface Protocol Checklist (PCI-SIG, Rev. 2.1)**. Complete protocol checklist following the same style as the PCI-SIG checklist. See the `/docs/pcicomp1.pdf` Acrobat file on the CD-ROM.
- **XC4000 Series Technical Data**. Data sheet for XC4000E and XC4000EX FPGA devices. See the `/docs/xc4000e.pdf` Acrobat file on the CD-ROM.
- **Implementing FIFOs in XC4000E**. Application note. P/N: 0010273-01. See the `/docs/xapp053.pdf` Acrobat file on the CD-ROM.

- **Synchronous and Asynchronous FIFO Designs.** Application note. See the `/docs/xapp051.pdf` Acrobat file on the CD-ROM.

Contact:

Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124
TEL: 1-408-559-7778
FAX: 1-408-879-4442
E-mail: pci@xilinx.com
WEB: <http://www.xilinx.com>

IMPORTANT!



Be sure to visit the Xilinx WebLINX web site for the latest information and application notes using the LogiCore PCI interface.

20.6 LogiCore User's VIP Web Site

The LogiCore User's VIP web site provides a quick and convenient way to obtain the latest updates, documentation, design tips, application notes, and utilities. The VIP web site is open to registered LogiCore users. To register, point your Internet browser software to:

www.xilinx.com/products/logicore/logicore.htm

20.7 Perl Software

The LogiCore PCI Interface uses two Perl software utilities. These utilities require Perl version 5.00 or later.

- `trim` removes any known warnings from the trimming report for clarity.
- `model_io`, version 2.01 or later, modifies the overly conservative timing data on device input and output flip-flops. The modified value match the guaranteed values specified in the XC4000E data sheet.

Perl source is provided for these utilities. The Perl executable required to execute these utilities is not provided on the CD-ROM. Perl is widely available, at no cost, via the World Wide Web. Perl is also available on a wide range of platforms.

The easiest method to download the Perl source is to visit the Perl home page at

<http://www.perl.com>

From there, the CPAN (Comprehensive Perl Archive Network) contains links to various Perl FTP sites around the world. The Perl program ported to various computer platforms can be found in a `ports` sub-directory.

Example:

<ftp://ftp.sedl.org/pub/mirrors/CPAN/ports>

Another easy method to find the Perl program for your platform is to point your Internet browser at the LogiCore VIP Web Site (see section 20.6).

20.8 Low-level PCI Software Drivers

The LogiCore PCI Interface does not include low-level software drivers. However, such software is available from Codesign.

The Codesign PCI OCX is targeted at custom PCI board developers. The OCX allows easy access to a PCI device through Visual Basic, Delphi, Visual C++, or any other environment that uses OCXs. The OCX is fully Plug-n-Play compatible and can handle interrupts, port I/O, and memory I/O under Windows 95. A control panel applet allows the user to specify vendor and device IDs for their custom PCI device. The OCX automatically detects what resources are allocated to the device by the operating system. The PCI OCX is great for prototyping and demonstrating new PCI hardware devices and for developing end-user applications."

Part Number: 2A91

Price: US\$199.00 each

For fastest response, please send orders and inquiries to info@cdcorp.com.

Contact:

Codesign
3712 N Broadway
Suite 120
Chicago, IL 60613
E-mail: info@cdcorp.com

21. Appendix C: Waveforms

21.1 Initiator Interface

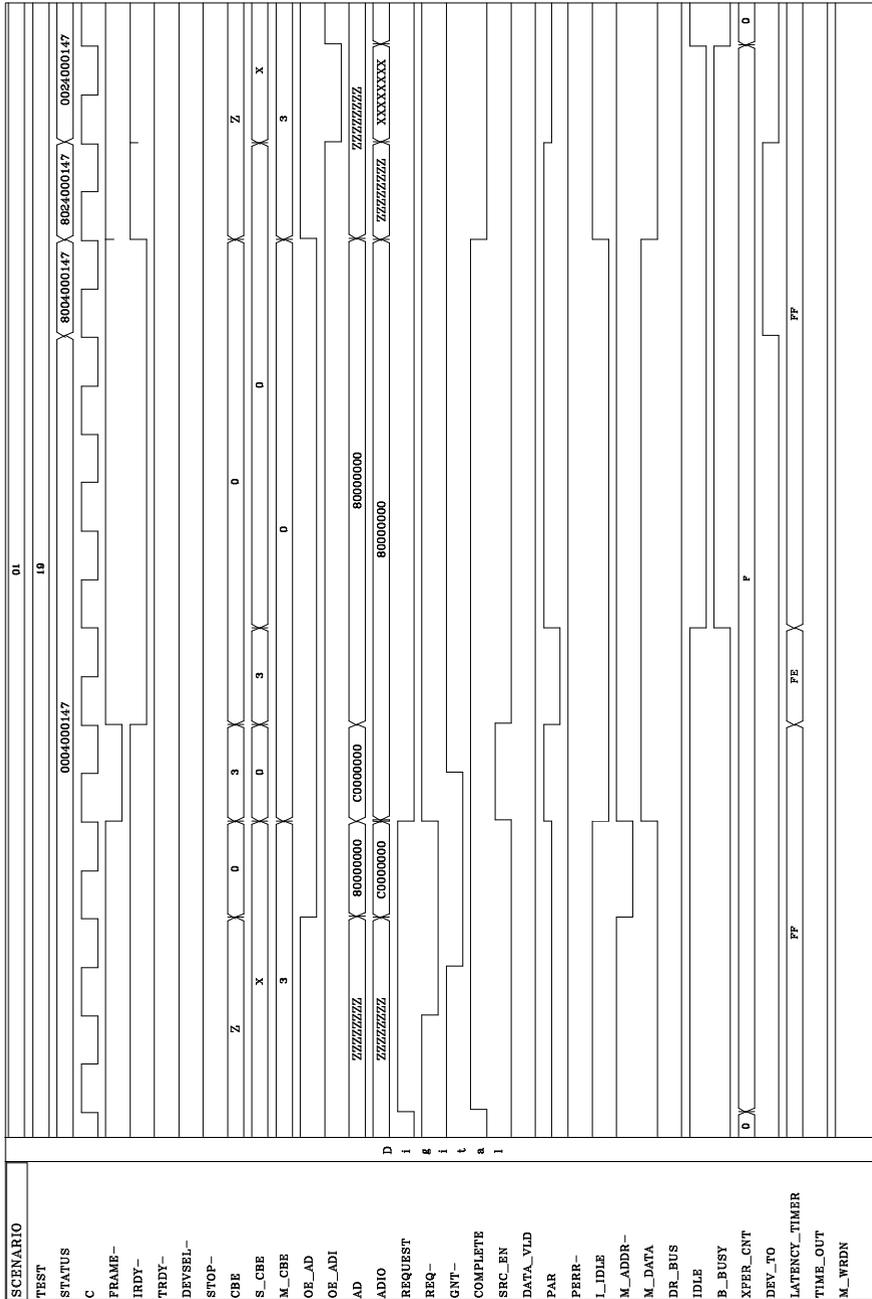


Figure 20. Master Abort during an Initiator Memory Write (Scenario 1.1.19). Note that Received Master Abort bit (CSR29) set in the Status Register. Note the DEV_TO internal signal indicates when the Initiator's Device Timeout counter expires.

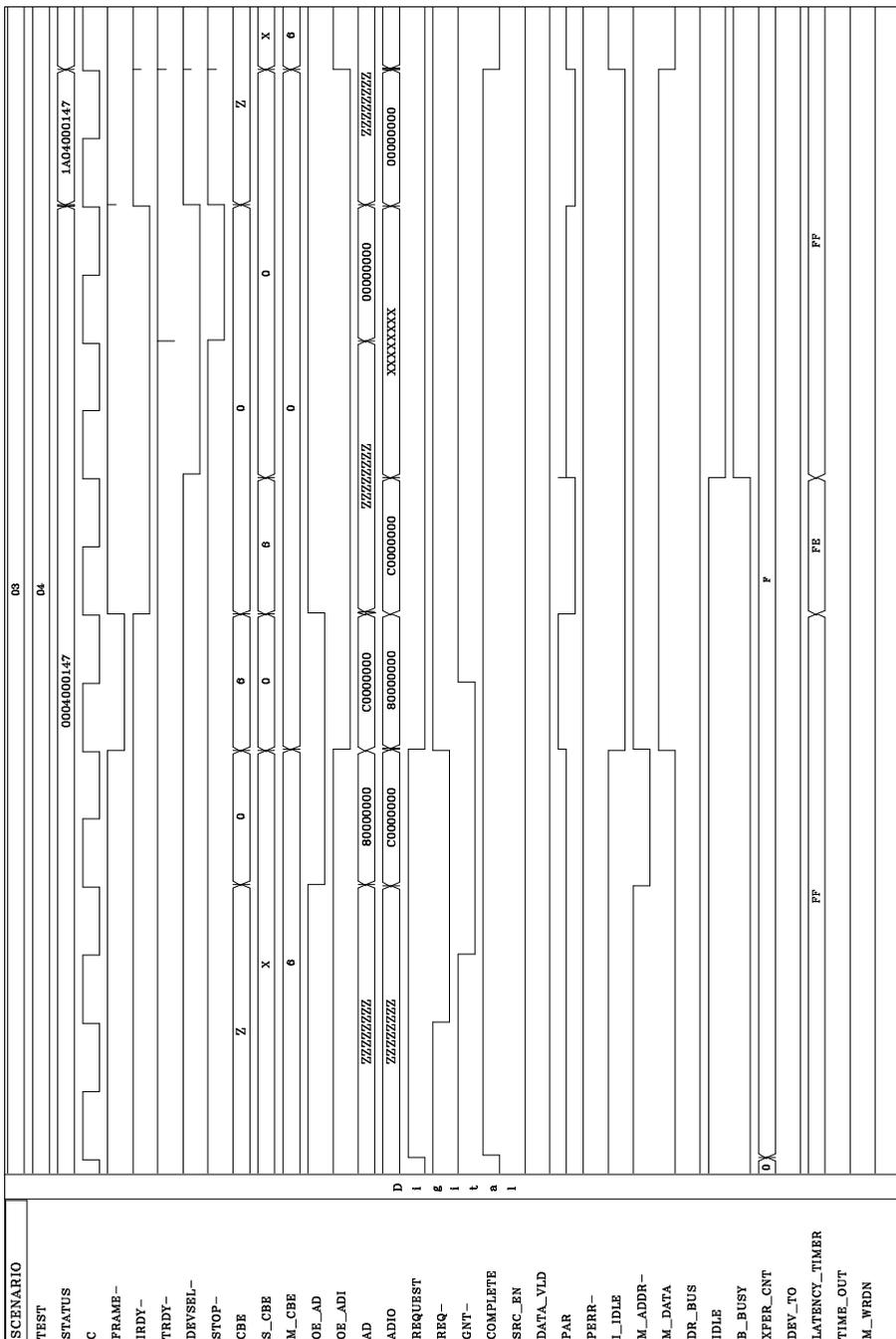


Figure 21. Initiator single data-phase Memory Write to a medium-speed slave that signals a Target Retry condition (Scenario 1.3.4).

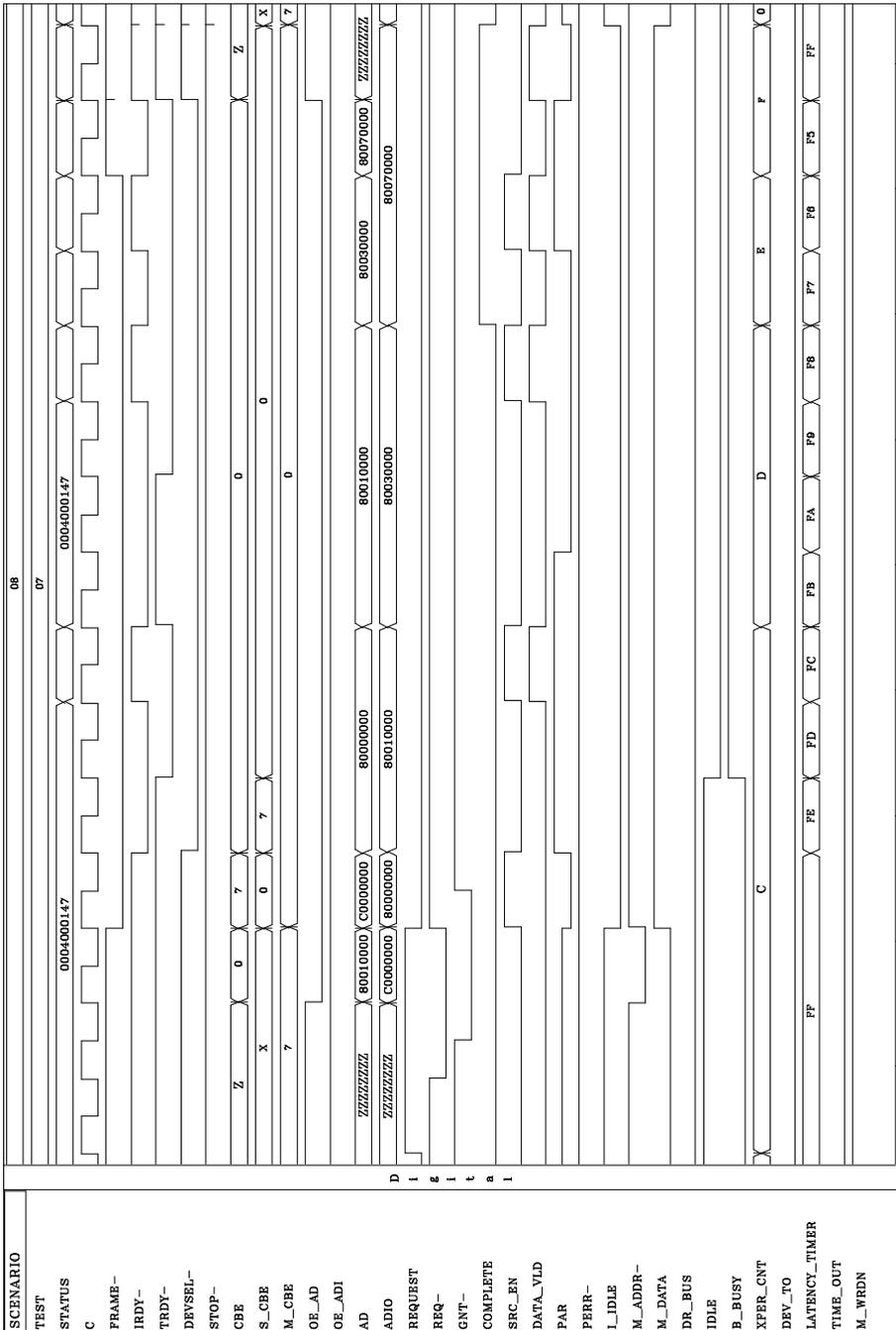


Figure 23. Initiator multi data-phase Memory Write to a fast-speed slave that inserts TRDY- wait-states between 4th and 6th rising clock edge after FRAME- (Scenario 1.8.7). Note that the LogiCore Initiator automatically inserts IRDY- wait-states after a successful data transfer.

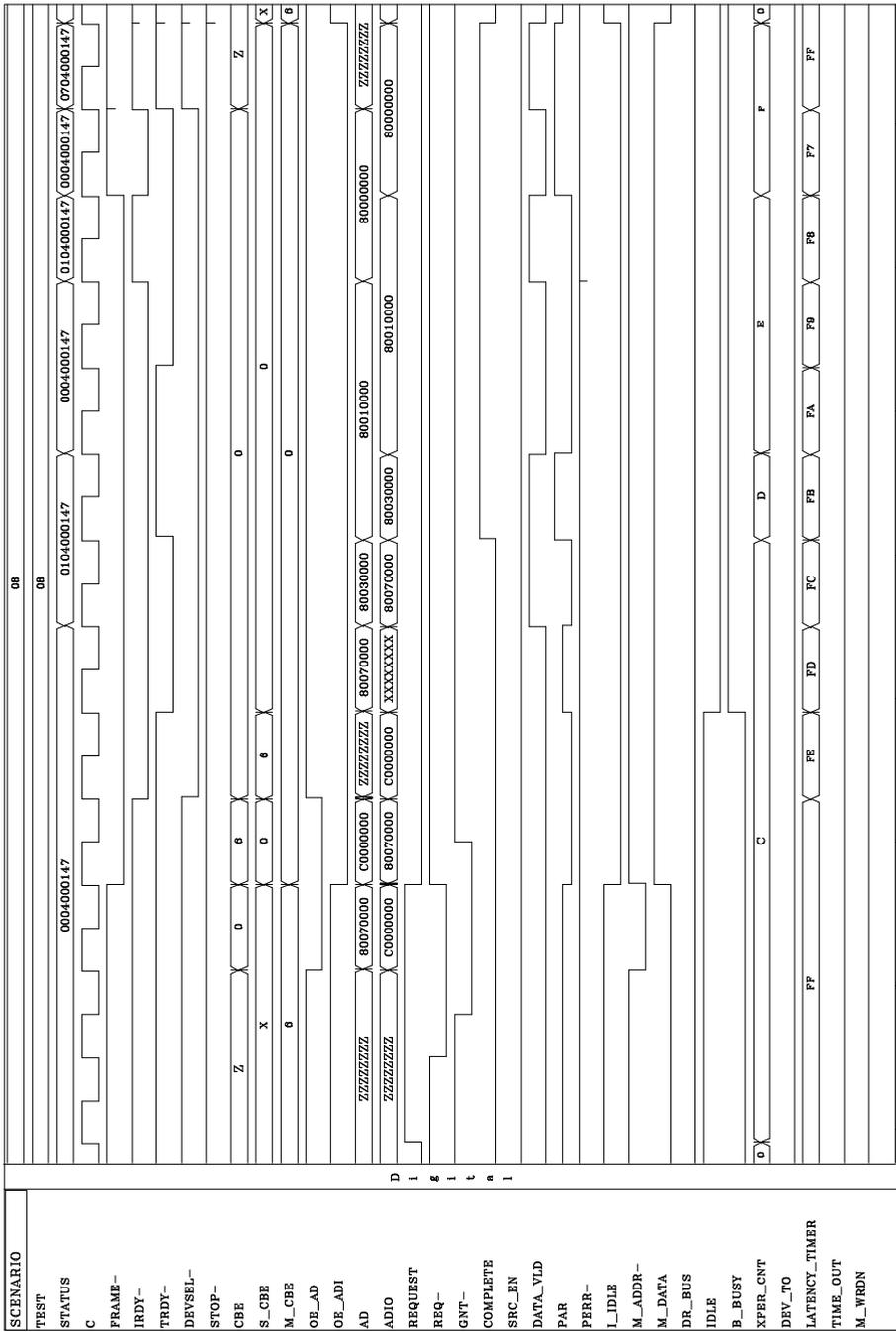


Figure 24. Initiator multi data-phase Memory Read to a fast-speed slave that inserts TRDY- wait-states between 4th and 6th rising clock edge after FRAME- (Scenario 1.8.8). Note timing of COMPLETE signal. Also note that the LogiCore Initiator automatically inserts IRDY- wait-states before the last data transfer, before it de-asserts FRAME-.

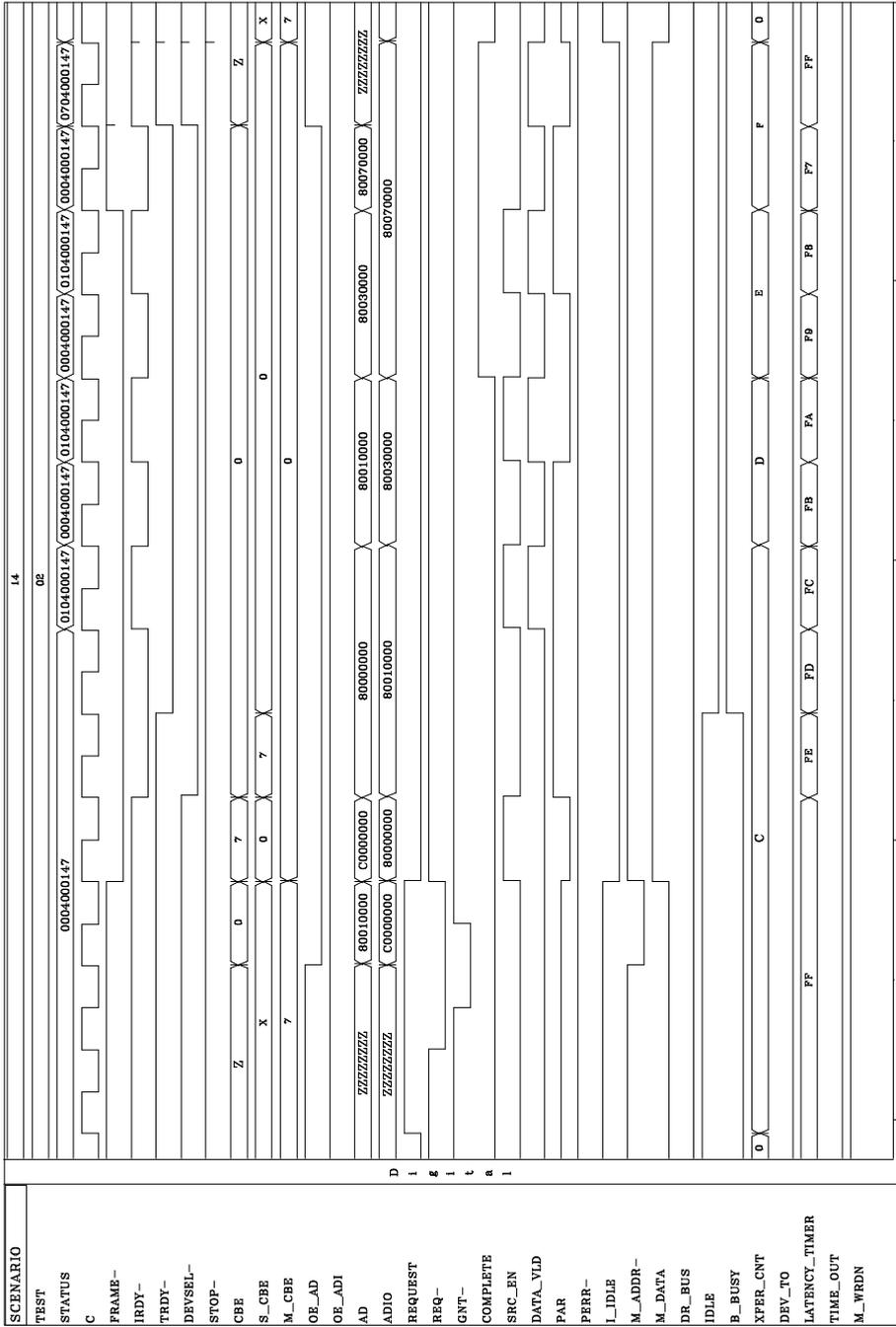


Figure 28. Initiator performing Memory Write and Memory Read transactions after receiving a single-cycle GNT- signal (Scenario 1.14.2).

21.2 Target Interface

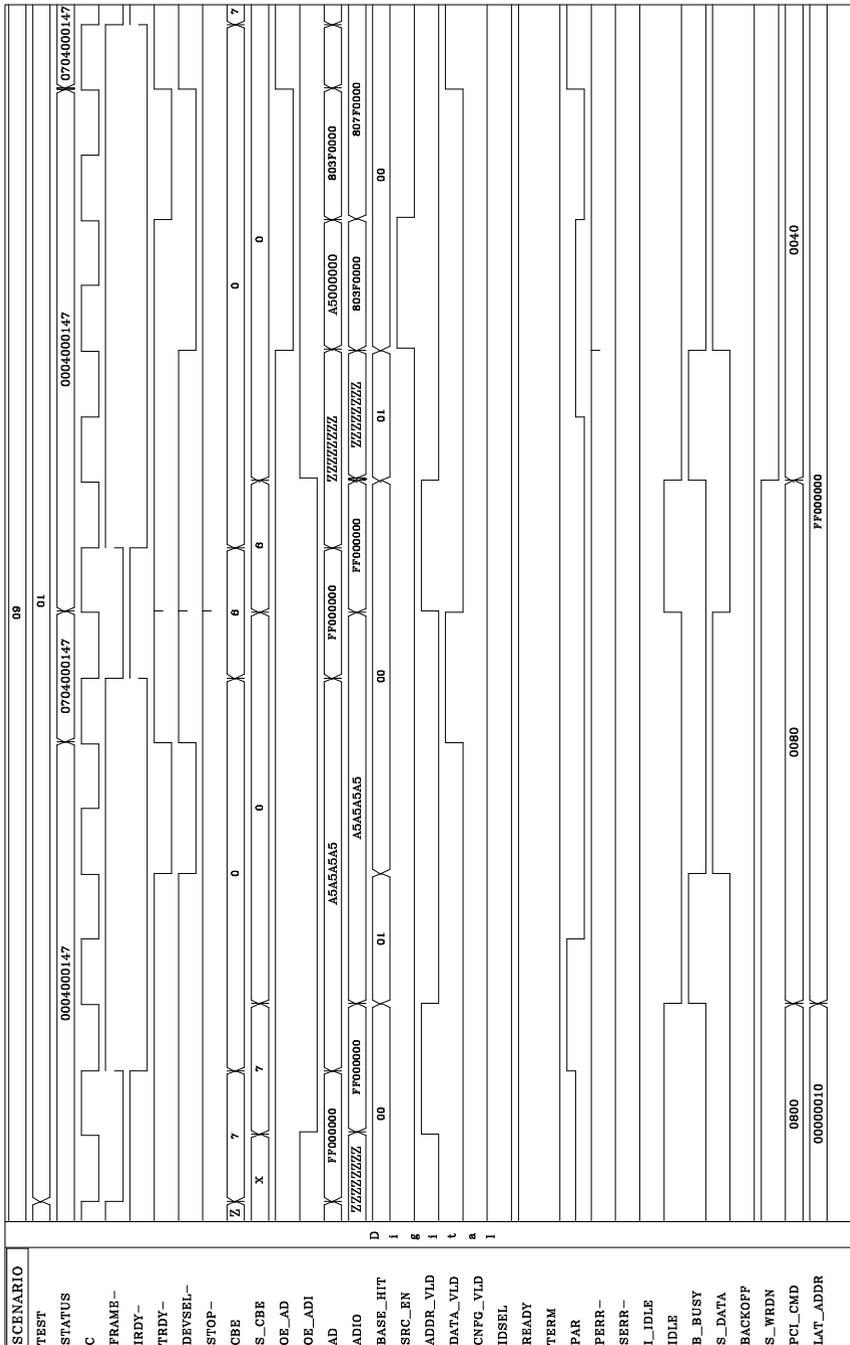


Figure 29. Target Memory Write followed by Memory Read, single double-word transfer (Scenario 2.9.1).

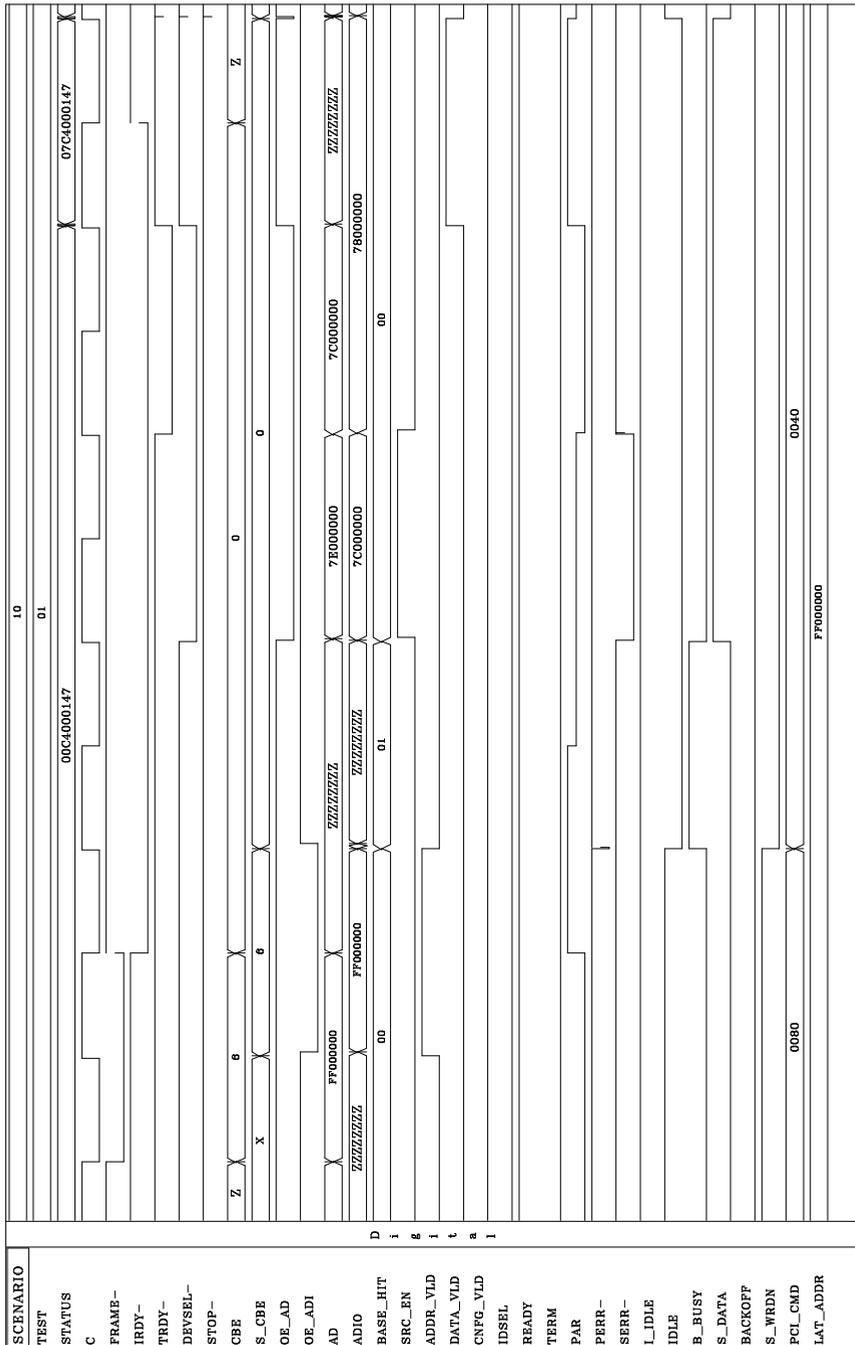


Figure 30. Address parity error during a Target Memory Read, single double-word transfer (Scenario 2.10.1a). Note that PAR is asserted High by the test scenario on the clock cycle following the address phase, which is incorrect parity. The LogiCore macro asserts SERR- Low indicating an address parity error. The System Error Signaled bit, CSR30, is also set in the Status Register.

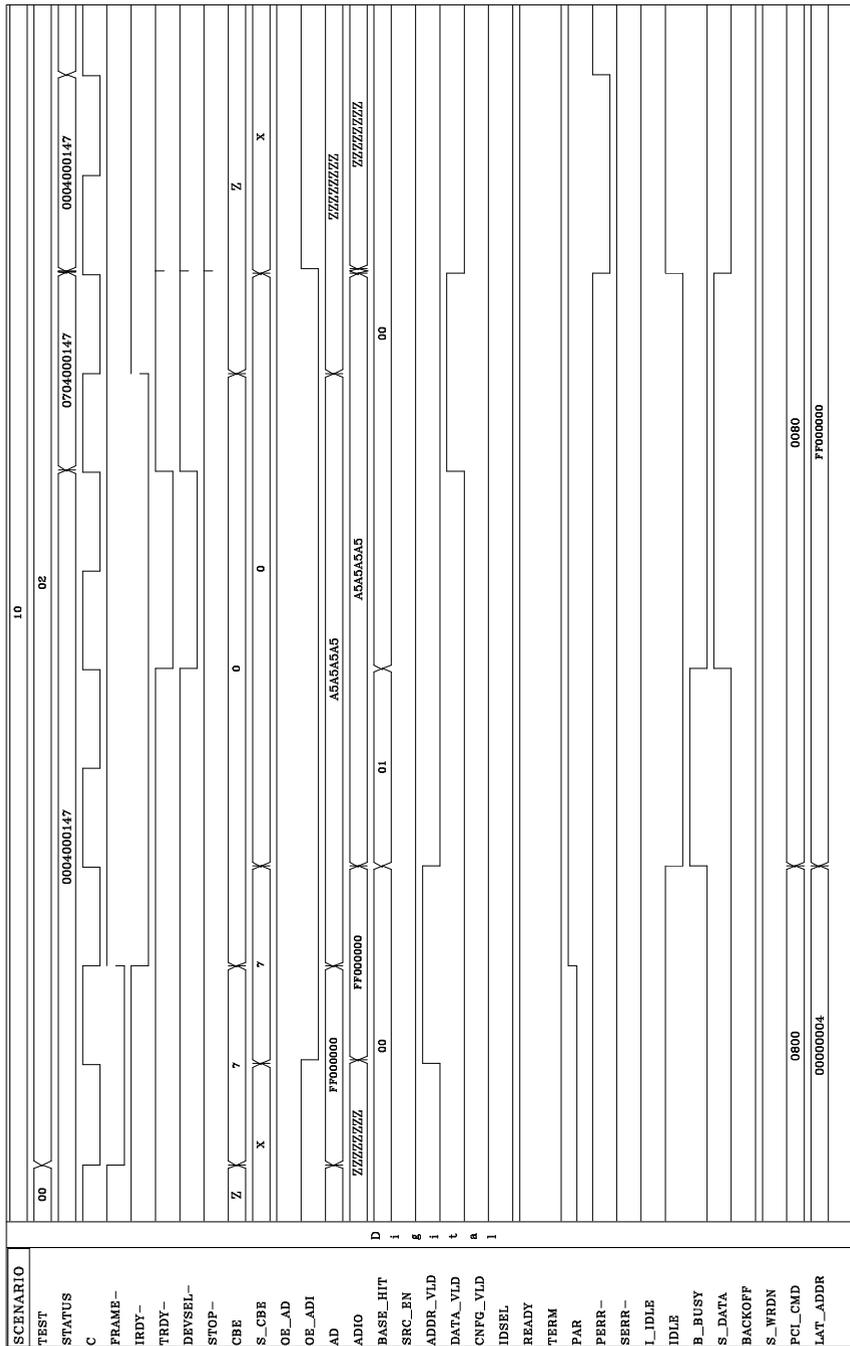


Figure 31. Data parity error during a Target Memory Write, single double-word transfer (Scenario 2.10.2). Note that PAR is asserted High by the test scenario on the clock cycle following the data phase, which is incorrect parity. The LogiCore macro asserts PERR- Low two cycles later indicating a data parity error. The Parity Error Detected bit, CSR31, is also set in the Status Register.

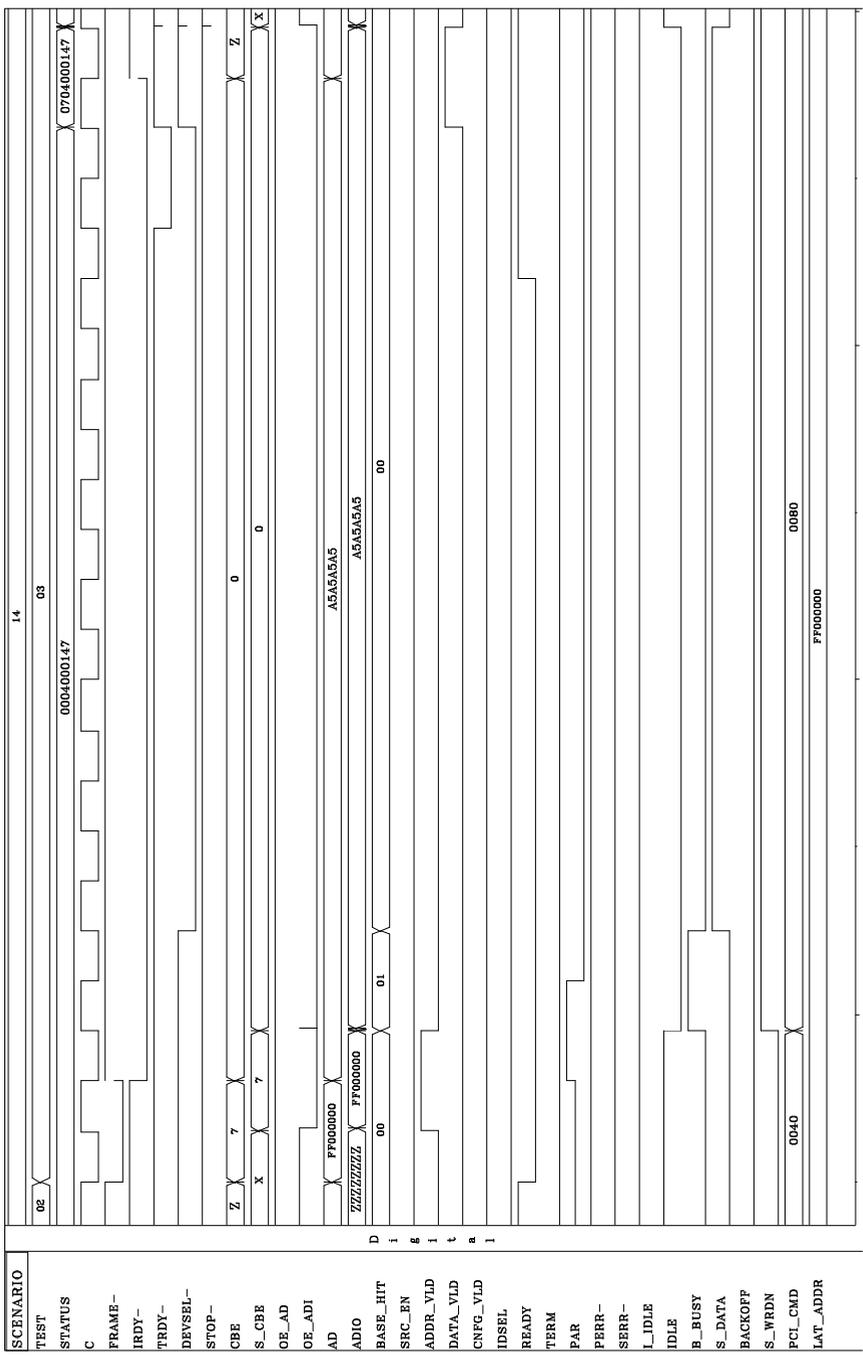


Figure 33. Target Memory Write, single double-word transfer. User application delays asserting READY for seven clock cycles (Scenario 2.14.3)

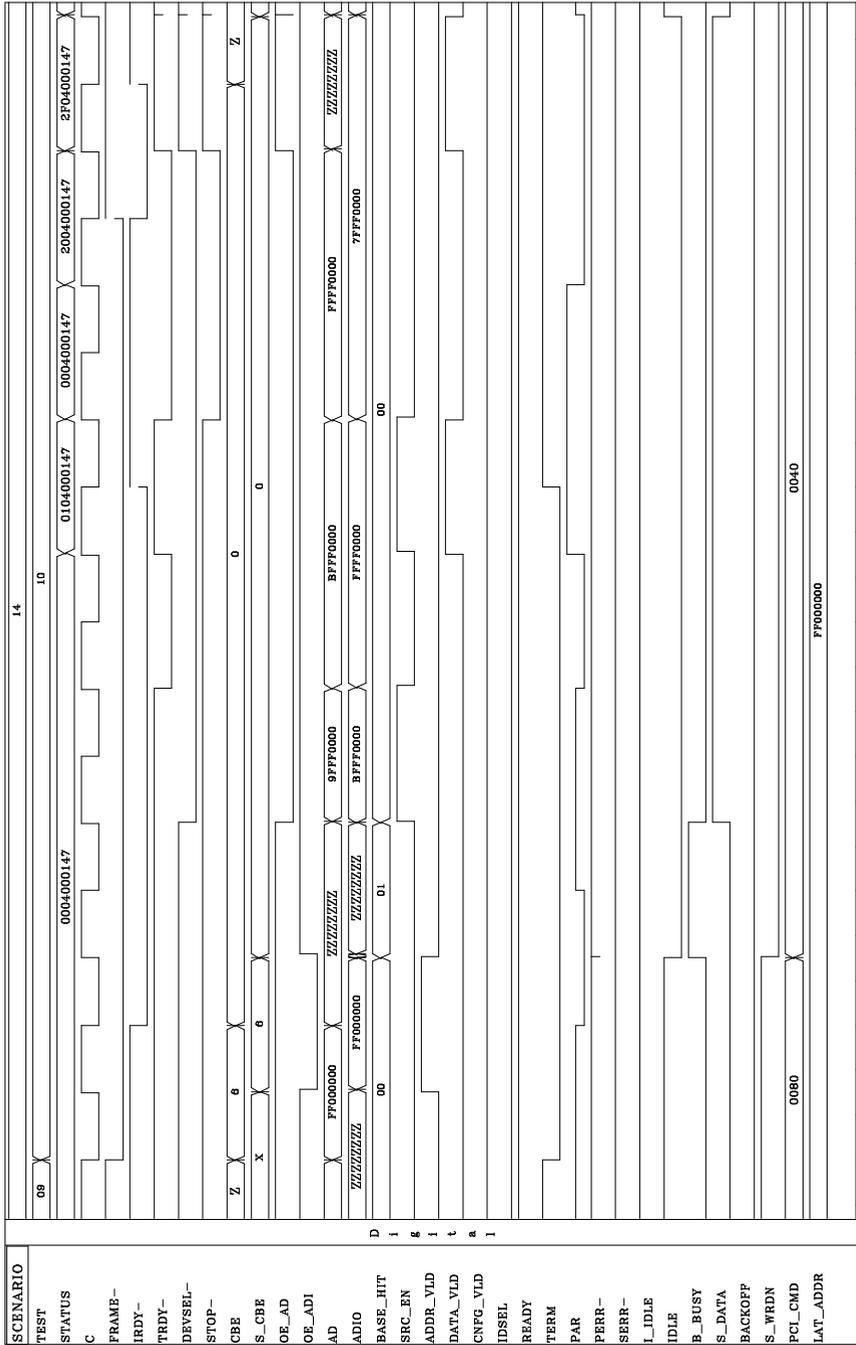


Figure 34. Target Memory Read, two double-word transfer. User application forces Target Disconnect by asserting **TERM** before second transfer (Scenario 2.14.10).

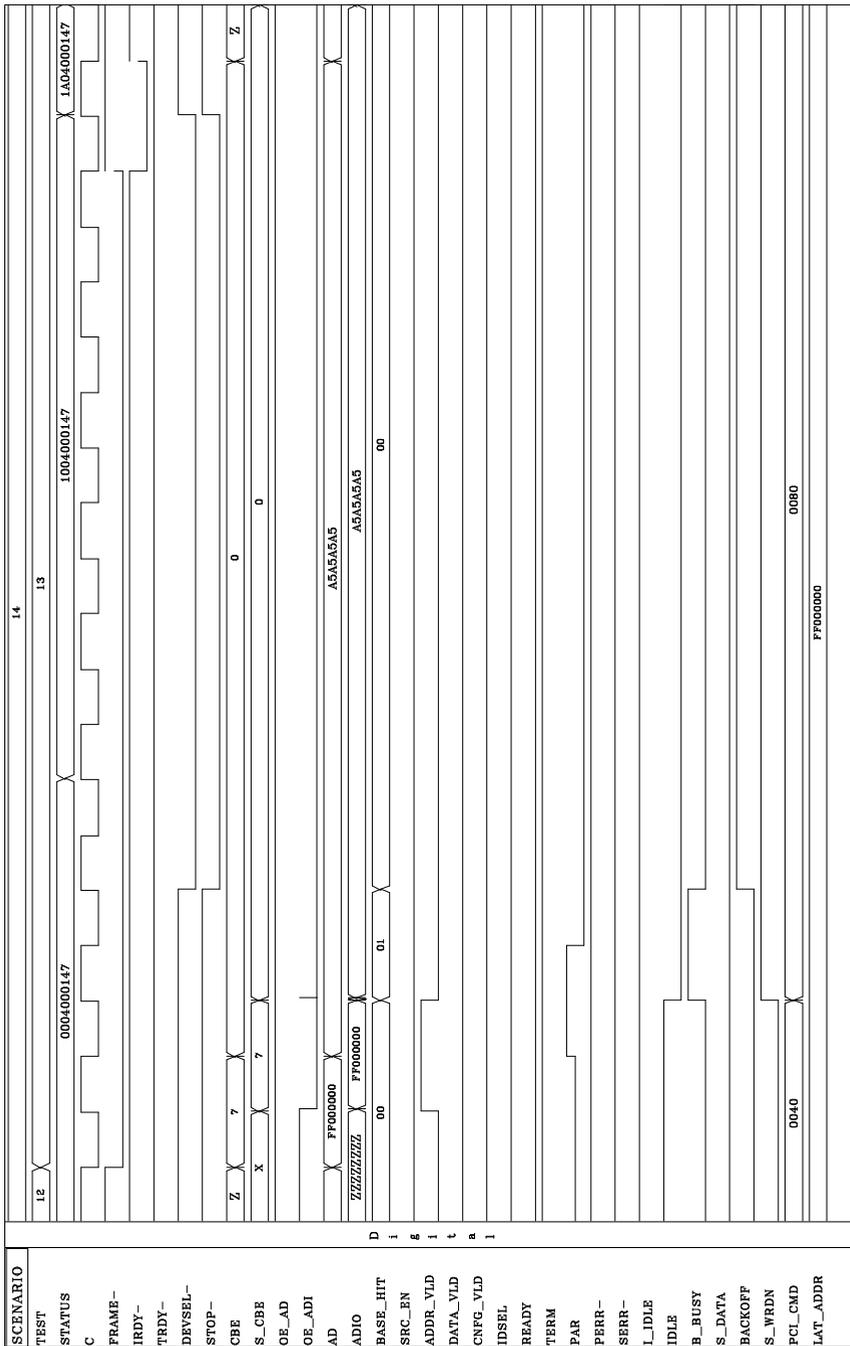


Figure 35. Target Memory Write, single double-word transfer. Initiator asserts FRAME- and de-asserts IRDY- for seven clock cycles (Scenario 2.14.13).

Notes:

North America**Corporate Headquarters**

Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124
U.S.A.

Tel: 1 (408) 559-7778
FAX: 1 (408) 559-7114
EMAIL: hotline@xilinx.com
WEB: <http://www.xilinx.com>

Northern California

Xilinx, Inc.
1281 Oakmead Parkway
Suite 202
Sunnyvale, CA 94086

Tel: (408) 245-9850
FAX: (408) 245-9865

Southern California

Xilinx, Inc.
15615 Alton Parkway
Suite 280
Irvine, CA 92718

Tel: (714) 727-0780
FAX: (714) 727-3128

New Hampshire

Xilinx, Inc.
61 Spit Brook Road
Nashua, NH 03060

Tel: (603) 891-1096
FAX: (603) 891-0890

Pennsylvania

Xilinx, Inc.
905 Airport Rd.
Suite 200
West Chester, PA 19380

Tel: (610) 430-3300
FAX: (610) 430-0470

Texas

Xilinx, Inc.
4100 McEwen
Suite 237
Dallas, TX 75244

Tel: (214) 960-1043
FAX: (214) 960-0927

Illinois

Xilinx, Inc.
939 N. Plum Grove Road
Suite H
Schaumburg, IL 60173

Tel: (708) 605-1972
FAX: (708) 605-1976

North Carolina

Xilinx, Inc.
6080-C Six Forks Rd.
Raleigh, NC 27609

Tel: (919) 846-3922
FAX: (919) 846-8316

Europe**United Kingdom**

Xilinx, Ltd.
Suite 1B, Cobb House
Oyster Lane, Byfleet
Surry KT14 7DU
UNITED KINGDOM

Tel: (44) 932-349401
FAX: (44) 932-349499
EMAIL: ukhelp@xilinx.com

France

Xilinx SARL
Espace Jouy Technology
21, rue Albert Calmette, Bt. C
78353 Jouy en Josas, Cedex
FRANCE

Tel: (33) 1-34-63-01-01
FAX: (33) 1-34-63-01-09
EMAIL: frhelp@xilinx.com

Germany

Xilinx, GmbH
Dorfstr. 1
85609 Aschheim
München
GERMANY

Tel: (49) 89-99-1549-0
FAX: (49) 89-99-904-4748
EMAIL: dlhelp@xilinx.com

Japan

Xilinx, K.K.
Daini-Nagaoka Bldg. 2F
2-8-5, Hatchobori Chuo-ku
Tokyo 104
JAPAN

Tel: (03) 3297-9191
FAX: (03) 3297-9189

Asia Pacific**Hong Kong**

Xilinx Asia Pacific
Unit No. 4312
Tower 2, Metroplaza
Hing Fong Road
Kwai Fong, N.T.
HONG KONG

Tel: (852) 2424-5200
FAX: (852) 2494-7159
EMAIL: hongkong@xilinx.com



The Programmable Logic CompanySM