LogiCORE™ IP Initiator/Target v6.8 for PCI-X™

Getting Started Guide

UG261 April 24, 2009

XILINX®

Xilinx is providing this product documentation, hereinafter “Information,” to you “AS IS” with no warranty of any kind, express or implied. Xilinx makes no representation that the Information, or any particular implementation thereof, is free from any claims of infringement. You are responsible for obtaining any rights you may require for any implementation based on the Information. All specifications are subject to change without notice. XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE INFORMATION OR ANY IMPLEMENTATION BASED THEREON, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF INFRINGEMENT AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Except as stated herein, none of the Information may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx.

The Design is not designed or intended for use in the development of on-line control equipment in hazardous environments requiring fail-safe controls, such as in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support, or weapons systems (“High-Risk Applications”). Xilinx specifically disclaims any express or implied warranties of fitness for such High-Risk Applications. You represent that use of the Design in such High-Risk Applications is fully at your risk.

© 2006-2009 Xilinx, Inc. Xilinx, Inc. XILINX, the Xilinx logo, Virtex, Spartan, ISE and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

Initiator/Target v6.8 for PCI-X Getting Started Guide
UG261 April 24, 2009

The following table shows the revision history for this document.

Date	Version	Revision
7/13/06	1.0	Initial Xilinx release.
2/15/07	2.0	Added Virtex-5 LXT, Synplicity, VHDL support, updated supported tools versions.
05/17/07	3.0	Changed title and text references for PCI and PCI-X to conform with PCI-SIG trademark guidelines. Advanced IUS support to v5.7.
08/08/07	3.5	Updated for IP1 Jade Minor release. Changed capacitor value to 10 uF to match XAPP653 recommendation.
10/10/07	4.0	Updated for IP2 Jade Minor release. Added information about Configuration Pins to Device Family chapter.
3/24/08	4.5	Updated for ISE v10.1 release.
4/25/08	5.0	Added support for Virtex-5 FXT devices.
9/19/08	5.5	Updated to support ISE v10.1 Service Pack 3.
4/24/09	6.0	Updated to support ISE v11.1.

Table of Contents

About This Guide

The *LogiCORE™ Initiator/Target v6.8 for PCI-X™ Getting Started Guide* provides information about the interface for Peripheral Component Interconnect Extended (PCI-X), which provides a fully verified, pre-implemented PCI-X bus interface targeting devices based on the Virtex®-5 FPGA architecture.

The guide also includes an example design in Cadence® IUS that lets you simulate, synthesize, and implement the interface to understand the PCI-X design flow.

Guide Contents

This manual contains the following chapters:

- Chapter 1, "Introduction" describes the core and provides information about getting technical support, and providing feedback to Xilinx about the core and the accompanying documentation.
- Chapter 2, "Licensing the Core" provides instructions for installing and obtaining a license for the PCI-X interface core, which you must do before using it in your designs.
- Chapter 3, "Getting Started" provides an overview of the example design and instructions for generating the core.
- Chapter 4, "Family Specific Considerations" provides design information specific to the Initiator/Target core for PCI-X targeting the Virtex-5 family of devices.
- Chapter 5, "Functional Simulation" describes the use of supported functional simulation tools, including Cadence IUS and Mentor Graphics® ModelSim®.
- Chapter 6, "Timing Simulation" describes the use of supported post-route timing simulation tools, including Cadence IUS and Mentor Graphics ModelSim.
- Chapter 7, "Synthesis and Implementation" describes the use of supported synthesis tools using the *Userapp* example design for step-by-step instructions.

Conventions

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands you enter in a syntactical statement	**`ngdbuild`** `design_name`
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *`design_name`*
	References to other manuals	See the *`Development System Reference Guide`* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *`not`* connected.
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	`ngdbuild [option_name] design_name`
Braces { }	A list of items from which you must choose one or more	**`lowpwr ={on|off}`**
Vertical bar |	Separates items in a list of choices	**`lowpwr ={on|off}`**

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis ...	Omitted repetitive material	**`allow block`** `block_name loc1 loc2... locn;`

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See "Additional Resources" for details. See "Title Formats" in Chapter 1 for details.
Blue, underlined text	Hyperlink to a website (URL)	Go to www.xilinx.com for the latest speed files.

XILINX®

Chapter 1

Introduction

The Initiator/Target core for PCI-X from Xilinx is a PCI-X 2.0 Mode 1 and PCI 3.0-compliant high-bandwidth parallel interconnect intellectual property building block for use with the Virtex-5 FPGAs. This core supports Verilog™ and VHDL. The example design described in this guide is provided in both languages, and is also described in this guide.

This chapter introduces the core and provides related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.

System Requirements

Windows

- Windows XP® Professional 32-bit/64-bit
- Windows Vista® Business 32-bit/64-bit

Linux

- Red Hat® Enterprise Linux WS v4.0 32-bit/64-bit
- Red Hat® Enterprise Desktop v5.0 32-bit/64-bit (with Workstation Option)
- SUSE Linux Enterprise (SLE) v10.1 32-bit/64-bit

Software

- ISE® 11.1

Check the release notes for the required service pack; ISE service packs can be downloaded from http://www.xilinx.com/support/download/index.htm.

About the Core

This core is an IP core available from the Xilinx CORE Generator. For detailed information about the core, see the PCI/PCI-X product page. For information about system requirements, installation, and licensing options, see Chapter 2, "Licensing the Core."

Recommended Design Experience

Although the Initiator/Target core for PCI-X is a fully verified solution, the challenge associated with implementing a complete design varies depending on the configuration and functionality of the application. For best results, previous experience building high

performance, pipelined FPGA designs using Xilinx implementation software and user constraints files (UCF) is recommended.

Additional Core Resources

For detailed information and updates about the core, see the following documents.

- *Initiator/Target v6.8 for PCI-X Data Sheet*
- *Initiator/Target v6.8 for PCI-X Release Notes*
- *Initiator/Target v6.8 for PCI-X User Guide*

Further information and resources relating to the PCI-X technology are available from the web site: PCI-X at PCI-SIG

Technical Support

For technical support, go to www.xilinx.com/support. Questions are routed to a team of engineers with expertise using the core.

Xilinx will provide technical support for use of this product as described in the *Initiator/Target v6.8 for PCI-X User Guide* and the *Initiator/Target v6.8 for PCI-X Getting Started Guide*. Xilinx cannot guarantee timing, functionality, or support of this product for designs that do not follow these guidelines.

Feedback

Xilinx welcomes comments and suggestions about the Initiator/Target core for PCI-X and the accompanying documentation.

Core

For comments or suggestions about the core, please submit a WebCase from www.xilinx.com/support. Be sure to include the following information:

- Product name
- Core version number
- Explanation of your comments

Document

For comments or suggestions about this document, please submit a WebCase from www.xilinx.com/support. Be sure to include the following information:

- Document title
- Document number
- Page number(s) to which your comments refer
- Explanation of your comments

Chapter 2

Licensing the Core

This chapter provides instructions for installing and obtaining a license for the Initiator/Target core for PCI-X, which you must do before using it in your designs. The core is provided under the terms of the Xilinx LogiCORE Site License Agreement, which conforms to the terms of the SignOnce IP License standard defined by the Common License Consortium. Purchase of the core entitles you to technical support and access to updates for a period of one year.

Before You Begin

This chapter assumes you have installed the core using either the CORE Generator™ IP Software Update installer or by performing a manual installation after downloading the core from the web.

License Options

The Initiator/Target core for PCI-X provides three licensing options. After installing the required Xilinx ISE software and IP Service Packs, choose a license option.

Simulation Only

The Simulation Only Evaluation license key is provided with the Xilinx CORE Generator tool. This key lets you assess core functionality with either the example design provided with the Initiator/Target core for PCI-X, or alongside your own design and demonstrates the various interfaces to the core in simulation. (Functional simulation is supported by a dynamically generated HDL structural model.)

Full System Hardware Evaluation

The Full System Hardware Evaluation license is available at no cost and lets you fully integrate the core into an FPGA design, place-and-route the design, evaluate timing, and perform functional simulation of the Initiator/Target core for PCI-X using the example design and demonstration test bench provided with the core.

In addition, the license key lets you generate a bitstream from the placed and routed design, which can then be downloaded to a supported device and tested in hardware. The core can be tested in the target device for a limited time before timing out (ceasing to function), at which time it can be reactivated by reconfiguring the device.

Full

The Full license key is available when you purchase the core and provides full access to all core functionality both in simulation and in hardware, including:

- Functional simulation support
- Full implementation support including place and route and bitstream generation
- Full functionality in the programmed device with no time outs

Obtaining Your License Key

This section contains information about obtaining a simulation, full system hardware, and full license keys.

Simulation License

No action is required to obtain the Simulation Only Evaluation license key; it is provided by default with the Xilinx CORE Generator software.

Full System Hardware Evaluation License

To obtain a Full System Hardware Evaluation license, do the following:

1. Navigate to the product page for this core: www.xilinx.com/pci
2. Click **Evaluate**.
3. Follow the instructions to install the required Xilinx ISE software and IP Service Packs.

Full License

To obtain a Full license key, you must purchase a license for the core. After doing so, click the "Access Core" link on the Xilinx.com IP core product page for further instructions.

Installing Your License File

The Simulation Only Evaluation license key is provided with the ISE CORE Generator system and does not require installation of an additional license file. For the Full System Hardware Evaluation license and the Full license, an email will be sent to you containing instructions for installing your license file. Additional details about IP license key installation can be found in the ISE Design Suite Installation, Licensing and Release Notes document.

XILINX®

Chapter 3

Getting Started

This chapter provides an overview of the example design and instructions for generating the core. Subsequent chapters describe how to simulate and implement the example design using the provided demonstration test bench and compilation scripts.

Overview

The example design consists of the following:

- Core netlists
- Core simulation models
- Example HDL wrapper (which instantiates the cores and example design)
- A demonstration test bench to simulate the example design

The example design has been tested with Xilinx ISE 11.1 and the following simulators:

- Cadence IUS v8.1 -s006
- Mentor Graphics ModelSim v6.4a

Generating the Core

To generate an Initiator/Target core for PCI-X using the Xilinx CORE Generator tool:

1. Start the CORE Generator tool.

   For help starting and using the CORE Generator tool, see the Xilinx CORE Generator Guide, available from the ISE documentation web page.
2. Choose File > New Project.
3. Type a directory name.

   Note: The name `<project_dir>` is used in the Back to Top section on page 13.
4. Set the following project options:
   - Part Options

     From Target Architecture, select the desired family. For a list of supported families, see the *Initiator/Target v6.8 for PCI-X Data Sheet*.

     Note: If an unsupported silicon family is selected, the core will not appear in the taxonomy tree.
   - Generation Options

     For Design Entry, select VHDL.

     For Vendor, select Synplicity® or Other (for XST).

5. After creating the project, locate the core interface in the taxonomy tree under Standard Bus Interfaces > PCI.
6. Double-click LogiCORE Initiator/Target for PCI-X (Virtex-5 devices only) v6.8 to display the main PCI-X screen.
7. In the Component Name field, enter a name for the core instance.

   Note: The name `<component_name>` is used in the Back to Top section on page 13.
8. After selecting the desired features and parameters from the GUI screens, click Finish.

   The core and its supporting files, including the example design, are generated in the project directory. For detailed information about the example design files and directories see "Directory Structure," page 13.

Unsupported Devices

If you wish to target a device/package combination that is not officially supported (not listed in the *Initiator/Target for PCI-X Data Sheet*), you may use the UCF Generator for PCI/PCI-X to create a user constraints file that implements a suitable pinout for your target device. This tool is available in the Xilinx CORE Generator under **UCF Generator for PCI/PCI-X**. For more information on this tool, consult the *UCF Generator for PCI/PCI-X Data Sheet*.

Note: It is important to verify the UCF files generated by this tool to confirm that the timing requirements of your application are met. Xilinx cannot guarantee that every UCF file generated by the UCF Generator tool will work for every application.

Directory Structure

This section provides a description of files and the directory structure generated by the Xilinx CORE Generator, the purpose and contents of the provided scripts, the contents of the example HDL wrappers, and the operation of the demonstration test bench.

<project directory>
Top-level project directory; name is user-defined

<project directory>/<component name>
Core release notes file

<component name>/doc
Product documentation

<component name>example design
Verilog and VHDL (or whichever, if it's only one) design files

<component name>/implement
Implementation script files

implement/results
Results directory, created after implementation scripts are run, and contains implement script results

<component name>/simulation
Simulation scripts

simulation/functional
Functional simulation files

simulation/timing
Timing simulation files

<project directory>

The <project directory> contains all the CORE Generator project files.

Table 3-1: **Project Directory**

Name	Description
`<component_name>.ngc`	Top-level netlist.
`<component_name>.v[hd]`	Verilog or VHDL simulation model.
`<component_name>.xco`	CORE Generator project-specific option file; can be used as an input to the CORE Generator.
`<component_name>_flist.txt`	List of files delivered with core.
`<component_name>_readme.txt`	Readme file.

<component name>/doc

The doc directory contains the PDF documentation provided with the core.

Table 3-2: **Doc Directory**

Name	Description
<project_dir>/<component_name>/doc	
pcix_64_ds208.pdf	*Initiator/Target v6.8 for PCI-X Data Sheet*
pcix_64_gsg261.pdf	*Initiator/Target v6.8 for PCI-X Getting Started Guide*
pcix_64_ug263.pdf	*Initiator/Target v6.8 for PCI-XL User Guide*

Back to Top

<project directory>/<component name>

The <component name> directory contains the release notes file provided with the core, which may include last-minute changes and updates.

Table 3-3: **Component Name Directory**

Name	Description
<project_dir>/<component_name>	
pcix_64_readme.txt	Release notes text file.

Back to Top

<component name>example design

The example design directory contains the example design files provided with the core.

Table 3-4: **Example Design Directory**

Name	Description
<project_dir>/<component_name>/example_design	
<component_name>_top.v[hd]	Verilog or VHDL top-level example design.
<component_name>_top.ucf	Example design user constraints file.
pcix_lc_64.v[hd]	Wrapper file for core netlist, instantiated under <component_name>_top.v[hd].
<component_name>.v[hd]	Black-box file instantiated in pcix_lc_64.v[hd].

Back to Top

<component name>/implement

The implement directory contains the core implementation script files.

Table 3-5: **Implement Directory**

Name	Description
<project_dir>/<component_name>/implement	
implement.{bat | sh}	DOS or UNIX/Linux synthesis and implementation script.
synplify.prj	Snyplify project file and synthesis script.
xst.prj	XST project file.
xst.scr	XST synthesis script.

Back to Top

implement/results

The results directory is created by the implement script, after which the implement script results are placed in the results directory.

Table 3-6: **Results Directory**

Name	Description
<project_dir>/<component_name>/implement/results	
Created by the implementation script; implement script results are placed in directory.	

Back to Top

<component name>/simulation

The simulation directory contains the simulation scripts provided with the core.

Table 3-7: **Simulation Directory**

Name	Description
<project_dir>/<component_name>/simulation	
test_tb.v[hd]	Top-level simulation test bench.
sim_tasks.v	Definitions for common simulation tasks.
stimulus.v[hd]	Verilog or VHDL stimulus file.
busrec.v[hd]	Bus recorder module: logs the state of the core interface to a file.

Back to Top

simulation/functional

The functional directory contains functional simulation scripts provided with the core.

Table 3-8: **Functional Directory**

Name	Description
<project_dir>/<component_name>/simulation/functional	
simulate_ncsim.{bat | sh}	Cadence IUS simulation script.
wave.sv	Cadence IUS waveform, invoked by simulate_ncsim.{bat|sh}.
simulate_mti.do	ModelSim simulation script.
wave.do	ModelSim waveform display script; invoked by simulate_mti.do.

Back to Top

simulation/timing

The timing directory contains timing simulation scripts provided with the core.

Table 3-9: **Timing Directory**

Name	Description
<project_dir>/<component_name>/simulation/timing	
simulate_ncsim.{bat | sh}	Cadence IUS simulation script.
wave.sv	Cadence IUS waveform, invoked by simulate_ncsim.{bat|sh}.
simulate_mti.do	ModelSim simulation script.
wave.do	ModelSim waveform display script; invoked by simulate_mti.do.

Back to Top

XILINX®

Chapter 4

Family Specific Considerations

This chapter provides important design information specific to the PCI-X interface targeting Virtex-5 devices.

Device Initialization

Immediately after FPGA configuration, both the core interface and the user application are initialized by the startup mechanism present in all Virtex-5 devices.

During normal operation, the assertion of RST# on the PCI-X bus reinitializes the core interface and three-state all PCI-X bus signals. This behavior is fully compliant with the *PCI Local Bus Specification*. The core interface is designed to correctly handle asynchronous resets.

Typically, the user application must be initialized each time the core interfacee is initialized. In this case, use the **RST** output of the core interface as the asynchronous reset signal for the user application.

If part of the user application requires an initialization capability that is asynchronous to PCI-X bus resets, simply design the user application with a separate reset signal.

Note that these reset schemes require the use of routing resources to distribute reset signals, since the global resource is not used. The use of the global reset resource is not recommended.

Configuration Pins

Designers should be aware that PCI-X bus interface pins should not be placed on the dual purpose I/O pins used for configuration. Please verify the selected UCF to ensure that the pins do not conflict with the pins used for the chosen configuration mode. It is fine for PCI-X pins to be located on dual purpose I/O configuration pins that are NOT also used for configuration. Please refer to the appropriate device pin-out guide for locations of configuration pins.

Bus Width Detection

A core interface that provides a 64-bit datapath needs to know if it is connected to a 64-bit bus or a 32-bit bus. The core interface is capable of sensing and adjusting to the bus width automatically. However, this behavior can be manually forced by using the bus width disable (`BW_DETECT_DIS`) and bus width manual (`BW_MANUAL_32B`) inputs to the core. See the *Initiator/Target for PCI-X User Guide* for details and read the following section about bus mode detection.

Bus Mode Detection

A core interface that provides backward compatibility with PCI mode must determine whether it is in PCI-X bus mode or PCI bus mode. The core interface is capable of sensing and adjusting to the bus mode automatically. However, this behavior can be manually forced using the bus mode detect disable (`BM_DETECT_DIS`) and bus mode manual (`BM_MANUAL_PCI`) inputs to the core. See the *Initiator/Target for PCI-X User Guide* for details.

The core interface targeting Virtex-5 devices *cannot* support PCI-X bus mode and PCI bus mode with a single bitstream. For this reason, a fully compliant design requires two bitstreams and the ability to reconfigure the FPGA after the bus mode is detected.

For designs that use multiple bitstreams, the **RTR** output of the core interface will assert following the deassertion of the bus reset signal if the interface recognizes that the incorrect bitstream is in use. When this occurs, external circuitry is responsible for re-initializing the FPGA and loading an alternate bitstream. This requires storage for two complete bitstreams and another device, such as a CPLD, for managing the reconfiguration process. The reconfiguration process cannot be controlled by the FPGA because the FPGA becomes inactive during configuration.

The bitstream loaded in response to RTR will become active after the bus reset and the design will not be present to observe the busmode and buswidth broadcast. Missing the busmode broadcast is not an issue, as the newly loaded bitstream will be correct for the busmode in use. However, the newly loaded bitstream will not know if the bus is 32-bit or 64-bit. Upon the assertion of **RTR**, the FPGA must save the buswidth state in the CPLD so that the CPLD can restore it later. For more information about saving these values and designing a mechanism to reconfigure the FPGA, see XAPP 938.

Bus Clock Usage

The bus clock output provided by the interface is derived from the bus clock input, and is distributed using a global clock buffer. The interface itself is fully synchronous to this clock. In general, the portion of the user application that communicates with the interface must also be synchronous to this clock.

It is important to note that the frequency of this clock is not guaranteed to be constant. In fact, in a compliant system, the clock may be any frequency, up to and including the maximum allowed frequency, and the frequency may change on a cycle-by-cycle basis. Under certain conditions, the PCI-X core may also apply phase shifts to this clock.

For these reasons, the user application should not use this clock as an input to a DLL or PLL, nor should the user application use this clock in the design of interval timers (e.g. DRAM refresh counters).

Electrical Compliance

Virtex-5 devices, as specified in the relevant device data sheet, exhibit a 10 pF pin capacitance. This is compliant with the *PCI Local Bus Specification*, with one exception. The specification requires an 8 pF pin capacitance for the IDSEL pin, to allow for non-resistive coupling to an AD[xx] pin. In practice, this coupling may be resistive or non-resistive, and is performed on the system board or backplane. For system board or backplane designs, use resistive coupling to avoid non-compliance. For add-in cards, this is not under the control of the designer.

The *PCI-X Addendum* requires an 8 pF pin capacitance for all pins. Virtex-5 devices do not comply with this requirement.

Although the PCI-X interface provides a direct PME# output from a general purpose I/O pin, this output signal has certain limitations. If the FPGA power is removed, the general purpose I/O pin will appear as a low impedance to ground. This appears to the system as an assertion of PME#. For this reason, implementations that use the PME# signal should employ an external buffering scheme that will prevent false assertions of PME# when power is removed from the FPGA device.

For 3.3 volt signaling in Virtex-5 devices, the V_{CCO}. supply must be reduced to 3.0 volts and derived from a precision regulator. This reduction of the output driver supply provides robust device protection without sacrificing PCI electrical compliance, even in the extreme case where the 3.3 volt system supply climbs as high as 3.6 volts as allowed by the *PCI Local Bus Specification*.

Figure 4-1 illustrates one possible low-cost solution to generate the required 3.0 volt output driver supply. Xilinx recommends the use of this circuit; however, other approaches using other regulators are possible.

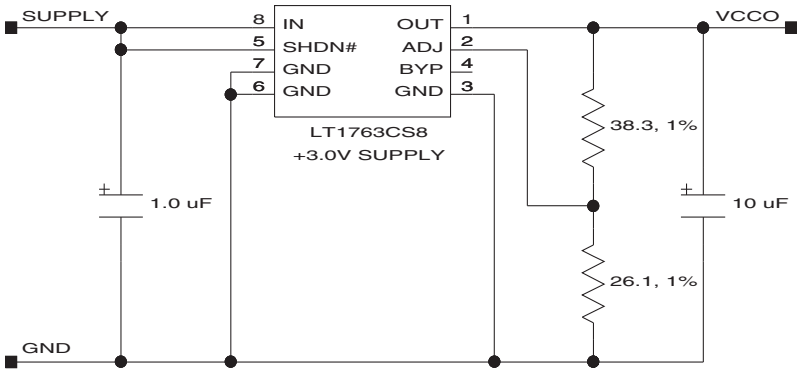

Figure 4-1: **PCI/PCI-X Pro Output Driver VCCO Generation**

Input Delay Buffers

Input delay buffers are used to provide guaranteed hold time on all bus inputs when in PCI bus mode. Where possible, the PCI-X interface targeting Virtex-5 devices uses input delay elements present in the IOBs of the FPGA device. The use of these delay buffers is selected through the implementation specific constraints file.

Virtex-5 FPGA implementations that support PCI bus mode make use of the IDELAY input delay buffer primitives. An IDELAY input delay buffer is a calibrated and adjustable delay line. This delay mechanism provides superior performance over the legacy input delay buffers.

Designs that use IDELAY primitives also require the use of the IDELAYCTRL primitive. The function of the IDELAYCTRL primitive is to calibrate the IDELAY delay lines. To perform this calibration, the IDELAYCTRL primitive requires a 200 MHz input clock. The design and wrapper files for use with reference clocks contain IDELAY instances, IDELAYCTRL instances, and an additional input, REF_I, for a 200 MHz reference clock from an I/O pin. This reference clock is distributed to all applicable IDELAYCTRL primitives using a global clock buffer.

It is important to note that there is some flexibility in the origin, generation, and use of this 200 MHz reference clock. The provided design and wrapper files represent a trivial case that can may be modified to suit specific design requirements:

- For designs requiring IDELAY and IDELAYCTRL for other IP cores, or custom user logic, the 200 MHz reference clock can be shared. It is possible to tap the reference clock in the wrapper file, after it is driven by the global buffer. This signal may be used by other IDELAY and IDELAYCTRL instances.
- For designs that already have a 200 MHz reference clock distributed on a global clock buffer, this clock can be shared. The wrapper file can be modified to remove the external I/O pin and the global clock buffer instance. Simply tap the existing 200 MHz clock signal and bring it into the wrapper file for the interface to use.
- For designs that do not have a 200 MHz reference clock, it may be possible to generate a 200 MHz reference clock using a Digital Clock Manager (DCM) and another clock. The other clock may be available internally or externally, but must be fixed frequency. In this case, verify the following:
  1. The jitter of the source clock, to determine if it is appropriate for use as an input to a DCM.
  2. The DCM configuration, to generate a 200 MHz clock on any appropriate DCM output (CLKFX, CLKDV, and so forth).
  3. The jitter of the derived 200 MHz reference clock, to determine if it is appropriate for use as an input to an IDELAYCTRL.
  4. The IDELAYCTRL reset must be tied to the DCM lock output so that the IDELAYCTRL remains in reset until the DCM is locked.

  For more information about the relevant timing parameters, see the *Virtex-5 FPGA Data Sheet* and *Virtex-5 FPGA User Guide*. As with the other implementation options, the derived 200 MHz reference clock must be distributed by a global clock buffer to the IDELAYCTRL instances.

Warning: The fixed frequency requirement of the source clock precludes the use of the PCI bus clock, unless the design is used in an embedded/closed system where the PCI bus clock is known to be a fixed frequency. See "Bus Clock Usage" for additional information about the allowed behavior of the PCI bus clock in compliant systems.

Generating Bitstreams

The bitstream generation program, bitgen, may issue DRC warnings when generating bitstreams for PCI-X designs. The number of these warnings varies depending on the configuration options used for the PCI-X core. Typically, these warnings are related to nets with no loads generated during trimming by the map program. Some of these nets are intentionally preserved by statements in the user constraints file.

Please be aware that the bitgen options provided with the example design are only for reference. The actual options used will depend on the desired FPGA configuration method and clock rate of your complete design, as implemented on a board. Please carefully

consider the following configuration time requirements when selecting a configuration method and clock rate.

1. Any designs that do not automatically sense both the bus width and bus mode must be configured within (100 ms + 2^{25} bus clocks) after the bus power rails become valid.
2. Any designs that must sense either the bus width or the bus mode must be configured within 100 ms after the bus power rails become valid.
3. Cardbus designs must be configured as quickly as possible after the bus power rails become valid.

XILINX®

Chapter 5

Functional Simulation

This chapter describes how to simulate the *Userapp* example design using the supported functional simulation tools, which include:

- Cadence IUS v6.1
- Mentor Graphics ModelSim v6.3c

Cadence IUS

Before attempting functional simulation, ensure that the IUS environment is properly configured and that the Xilinx simulation libraries have been compiled for use.

Verilog

1. Navigate to the functional simulation directory:

   ```
   cd <component_name>/simulation/functional
   ```

2. Open the `simulate_ncsim.sh` file.

   This file lists commands that invoke IUS on the example design. It includes compile commands for the example design and each of the test bench components, plus commands to run and view the simulation:

   ```
   echo 'Compiling PCIX Core Simulation Model'
   ncvlog -work work .. /../../<component_name>.v

   echo 'Compiling PCIX Example Design'
   ncvlog -work work ../../example_design/userapp.v
   ncvlog -work work ../../example_design/pcix_lc_64.v
   ncvlog -work work ../../example_design/<component_name>_top.v

   echo 'Compiling Test Bench'
   ncvlog -work work -incdir ../ ../stimulus.v
   ncvlog -work work ../busrec.v
   ncvlog -work work ../test_tb.v

   ncelab -access +r work.TEST_TB glbl

   ncsim -gui work.TEST_TB -input @"simvision -input wave.sv"
   ```

   Most of the files listed are related to the example design and its test bench. For other test benches, compile the core simulation model and your own test bench files.

This list does not include any configuration file, user application, top-level wrapper, or test bench. These additional files are required for a meaningful simulation.

3. Run the simulation by typing the following:

   `simulate_ncsim.sh`

   (Invoke simulate_ncsim.bat on Windows platforms.)

   This compiles all modules, runs the simulator and displays the waveform viewer.

VHDL

1. Navigate to the functional simulation directory:

   `cd <component_name>/simulation/functional`

2. Open the `simulate_ncsim.sh` file.

   This file lists commands that invoke IUS on the example design. It includes compile commands for the example design and each of the test bench components, plus commands to run and view the simulation:

```
echo 'Compiling PCIX Core Simulation Model'
ncvlog -v93 -work work .. /../../<component_name>.vhd

echo 'Compiling PCIX Example Design'
ncvhdl -v93 -work work ../../example_design/userapp.vhd
ncvhdl -v93 -work work ../../example_design/pcix_lc_64.vhd
ncvhdl -v93 -work work \
  ../../example_design/<component_name>_top.vhd

echo 'Compiling Test Bench'
ncvhdl -v93 -work work ../stimulus.v
ncvhdl -v93 -work work ../busrec.v
ncvhdl -v93 -work work ../test_tb.v

ncelab -access +r work.TEST_TB

ncsim -gui work.TEST_TB -input @"simvision -input wave.sv"
```

   Most of the files listed are related to the example design and its test bench. For other test benches, compile the core simulation model and your own test bench files.

   This list does not include any configuration file, user application, top-level wrapper, or test bench. These additional files are required for a meaningful simulation.

3. Run the simulation by typing the following:

   `simulate_ncsim.sh`

   (Invoke simulate_ncsim.bat on Windows platforms.)

   This compiles all modules, runs the simulator and displays the waveform viewer.

Mentor Graphics ModelSim

Before attempting functional simulation, ensure that the ModelSim environment is properly configured and that the Xilinx simulation libraries have been compiled for use with ModelSim.

Verilog

1. Navigate to the functional simulation directory:

   ```
   cd <component_name>/simulation/functional
   ```

2. Open the `simulate_mti.do` file.

   This file lists macro commands to be run in ModelSim. It includes compile commands for the example design and each of the test bench components, plus commands to run the simulation:

   ```
   # Compiling the core structural model
   vlog -work work .. /../../<component_name>.v

   # Compiling the example design
   vlog -work work ../../example_design/userapp.v
   vlog -work work ../../example_design/pcix_lc_64.v
   vlog -work work ../../example_design/<component_name>_top.v

   # Compiling the demonstration testbench
   vlog -work work +incdir+../ ../stimulus.v
   vlog -work work ../busrec.v
   vlog -work work ../test_tb.v

   vsim -L unisims_ver -t ps work.TEST_TB work.glbl

   do wave.do

   run 50us
   ```

3. Close the file. (It does not need to be modified.)

   Most of the files listed are related to the example design and its test bench. For other test benches, compile the core simulation model and your own test bench files.

   This list does not include any configuration file, user application, top level wrapper, or test bench. These additional files are required for a meaningful simulation.

4. Invoke ModelSim and ensure that the current directory is set to the following:

   ```
   <component_name>/simulation/functional
   ```

5. To run the simulation, type the following:

   ```
   do simulate_mti.do
   ```

   This compiles all modules, loads them into the simulator, displays the waveform viewer and runs the simulation.

VHDL

1. Navigate to the functional simulation directory:

   ```
   cd <component_name>/simulation/functional
   ```

2. Open the `simulate_mti.do` file.

   This file lists macro commands to be run in ModelSim. It includes compile commands for the example design and each of the test bench components, plus commands to run the simulation:

   ```
   # Compiling the core structural model
   vcom -work work .. /../../<component_name>.vhd
   ```

```
# Compiling the example design
vcom -work work ../../example_design/userapp.vhd
vcom -work work ../../example_design/pcix_lc_64.vhd
vcom -work work ../../example_design/<component_name>_top.vhd

# Compiling the demonstration testbench
vcom -work work ../stimulus.vhd
vcom -work work ../busrec.vhd
vcom -work work ../test_tb.vhd

vsim -L unisim -t ps work.TEST_TB

do wave.do

run 50us
```

Most of the files listed are related to the example design and its test bench. For other test benches, the following subset must be used for proper simulation of the PCI-X interface:

This list does not include any configuration file, user application, top level wrapper, or test bench. These additional files are required for a meaningful simulation.

3. Invoke ModelSim and ensure that the current directory is set to the following:

   `<component_name>/simulation/functional`

   To run the simulation, type the following:

   `do simulate_mti.do`

   This compiles all modules, loads them into the simulator, displays the waveform viewer and runs the simulation.

4. Alternatively, you can invoke ModelSim and run the script directly from the system prompt:

   `vsim -do simulate_mti.do`

   This compiles all modules, loads them into the simulator, displays the waveform viewer and runs the simulation.

XILINX®

Chapter 6

Timing Simulation

This chapter describes how simulate the *Userapp* example design using the supported timing simulation tools, Cadence IUS and Mentor Graphics ModelSim.

Cadence IUS

Before attempting timing simulation, ensure that the IUS environment is properly configured and that the Xilinx simulation libraries have been compiled for use.

1. Navigate to the timing simulation directory:

   `cd <component_name>/simulation/timing`

2. Open the `simulate_ncsim.sh` file.

   This script is similar to the one used for functional simulation (see "Cadence IUS" in Chapter 5) but instead of the example design RTL, it compiles the structural model for the implemented design. In addition, this script imports the delay values in the SDF file generated by the Xilinx tools.

3. Run the simulation by typing the following:

   `simulate_ncsim.sh`

   (Invoke simulate_ncsim.bat on Windows platforms.)

   This compiles all modules, runs the simulation and displays the waveform viewer.

Mentor Graphics ModelSim

Before attempting timing simulation, ensure that the ModelSim environment is properly configured and that the Xilinx simulation libraries have been compiled for use with ModelSim.

1. Navigate to the timing simulation directory:

   `cd <component_name>/simulation/timing`

2. Open the `simulate_mti.do` file.

   This script is similar to the one used for functional simulation (see "Mentor Graphics ModelSim" in Chapter 5). However, instead of the example design RTL, it compiles the structural model for the implemented design. In addition, this script imports the delay values in the SDF file generated by the Xilinx tools.

3. Invoke ModelSim and ensure that the current directory is set to:

   `<component_name>/simulation/timing`

To run the simulation, type:

```
do simulate_mti.do
```

This compiles all modules, loads them into the simulator, displays the waveform viewer and runs the simulation.

4. Alternatively, you can invoke ModelSim and run the script directly from the system prompt:

```
vsim -do simulate_mti.do
```

Chapter 7

Synthesis and Implementation

This chapter describes the use of supported synthesis tools using the *Userapp* example design for step-by-step instructions and illustrations.

Supported synthesis tools include:

- Synplicity Synplify
- Xilinx ISE software v11.1

Userapp consists of the design files listed in Table 7-1.

Table 7-1: ***Userapp*** **Example Design Files**

Name	Description
`pcix_lc_64.v[hd]`	Wrapper for `<component_name>.v[hd]` that interfaces to the user application and includes various I/O settings
`userapp.v[hd]`	User application that interfaces to the core interface
`<component_name>.v[hd]`	Black-box shell used for synthesis
`<component_name>_top.v[hd]`	Top-level design, instantiates `pcix_lc_64` and `userapp`

Synplicity Synplify

Before synthesizing a design, ensure that the Synplicity environment is properly configured.

1. Navigate to the implementation directory:

   ```
   cd <component_name>/implement
   ```

   The synthesis directory contains a script for use with Synplify; this script is called `implement.bat` for PC platforms and `implement.sh` for Unix and Linux platforms.

   The `synplify.prj` called in this script lists all the design files in Table 7-1. At the top of the list is also a `unisim.v[hd]` file, located in `$SYNPLICITY/lib/xilinx`. This file contains black boxes for all Xilinx simulation primitives and must be included as the first source file in all Xilinx designs.

2. If required, modify the files as required to suit your application.
3. Synthesize and implement the design by running the script.

The tool may issue warnings about unused signals; these warnings are expected as unused portions of the design are automatically trimmed.

Xilinx XST

Before attempting to synthesize a design, ensure that the Xilinx XST environment is properly configured. Synthesis is supported only from the XST command line.

1. Navigate to the implementation directory:

   cd <component_name>**/implement**

   The synthesis directory contains a script for use with Xilinx XST; this script is called `implement.bat` for PC platforms and `implement.sh` for Unix and Linux platforms. Note that the `xst.scr` and `run_xst.prj` files are common and used by both scripts.

2. If required, modify the files as required to suit your application.
3. Synthesize and implement the design by running the script.

The tool may issue warnings about unused signals; these warnings are expected as unused portions of the design are automatically trimmed.