LogiCORE™ IP Initiator/Target v4.13 for PCI™

Getting Started Guide

UG260 July 23, 2010

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.

© 2006–2010 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. PCI, PCI Express, PCIe, and PCI-X are trademarks of PCI-SIG.

Revision History

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, 66 MHz support
5/17/07	2.5	Fixed all references to PCI to conform to PCI-SIG trademark guidelines, advanced IUS version to 5.7.
8/08/07	3.0	Updated for IP1 Jade Minor release. Changed capacitor value to 10 uF to match XAPP653 recommendation.
10/10/07	3.5	Updated for IP2 Jade Minor release, reworded trademark references to comply with standards.
03/24/08	4.0	Updated for ISE v10.1 release.
4/25/08	4.5	Added Virtex-5 FXT support.
9/18/08	5.0	Advanced core version to 4.8, updated release date.
4/24/09	5.5	Updated to support ISE v11.1 and Spartan-6 FPGAs. Removed support for deprecated devices: Virtex-II, Virtex-II Pro, and Virtex-E.
6/24/09	6.0	Updated to support ISE v11.2.
9/16/09	7.0	Updated for v4.10 and ISE v11.3. Added additional part and package support for Spartan-6 devices.
12/02/09	7.5	Updated for v4.11 and to support ISE v11.4.
4/19/10	8.0	Updated for v4.12 and to support ISE v12.1.
5/24/10	8.1	Corrected title and footers.
7/23/10	9.0	Updated for v4.13 and to support ISE v12.2.

Table of Contents

About This Guide

The *LogiCORE™ IP Initiator/Target v4.13 for PCI Getting Started Guide* provides information about the Xilinx core interface for Peripheral Component Interconnect (PCI), which provides a fully verified, pre-implemented PCI bus interface targeting devices based on the Virtex®-5 architecture.

The guide also includes example designs in both Verilog-HDL and VHDL that lets you simulate, synthesize, and implement the interface to understand the design flow for PCI.

Guide Contents

This guide contains the following chapters:

- Chapter 1, "Introduction" describes the core and provides information about getting technical support and providing feedback to Xilinx.
- Chapter 2, "Licensing the Core" provides instructions for installing and obtaining a license for the PCI 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 that is specific to the Initiator/Target core targeting the Virtex and Spartan® families of devices.
- Chapter 5, "Functional Simulation" describes the use of supported functional simulation tools, including Cadence® IES and Mentor Graphics® ModelSim®.
- Chapter 6, "Synthesis and Implementation" describes the use of supported synthesis tools using the *Userapp* example design for step-by-step instructions.
- Chapter 7, "Timing Simulation" describes the use of supported post-route timing simulation tools, including Cadence IES and Mentor Graphics ModelSim.

Additional Resources

To find additional documentation, see the Xilinx website at:

http://www.xilinx.com/support/documentation/index.htm.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

http://www.xilinx.com/support/mysupport.htm.

Conventions

This document uses the following conventions. An example illustrates each convention.

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 that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *design_name*
	References to other manuals	See the *User 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.
Dark Shading	Items that are not supported or reserved	This feature is not supported
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}`**
Angle brackets < >	User-defined variable or in code samples	<directory name>

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

Horizontal ellipsis ...	Repetitive material that has been omitted	**`allow block`** *block_name loc1 loc2 ... locn;*

Convention	Meaning or Use	Example
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returned 45524943h.
	An '_n' means the signal is active low	**usr_teof_n** is active low.

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 the section "Additional Resources" for details. Refer to "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 LogiCORE Initiator/Target v4.12 for PCI core from Xilinx is a PCI 3.0-compliant, high-bandwidth parallel interconnect intellectual property building block for use with the Virtex®-5 FPGA. This core supports Verilog and VHDL and the example design described in this guide is provided in both languages.

This chapter introduces the Initiator/Target core for PCI 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® 12.2

Check the release notes for the required Service Pack; ISE Service Packs can be downloaded from www.xilinx.com/support/download/index.htm.

About the Core

The PCI core is an IP core available from the Xilinx CORE Generator™ software. For detailed information about the core, see the PCI/PCI-X product page.

For information about licensing options, see Chapter 2, "Licensing the Core."

Recommended Design Experience

Although the Initiator/Target core for PCI 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 PCI core, see the following documents.

- *Initiator/Target for PCI Data Sheet*
- *Initiator/Target for PCI Release Notes*
- *Initiator/Target for PCI User Guide*

Further information and resources relating to the PCI technology are available from the web site: PCI 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 Initiator/Target core for PCI.

Xilinx will provide technical support for use of this product as described in the *Initiator/Target for PCI User Guide* and the *Initiator/Target for PCI 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 core 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

XILINX®

Chapter 2

Licensing the Core

This chapter provides instructions for installing and obtaining a license for the Initiator/Target core for PCI, which you must do before using it in your designs. The core is provided under the terms of the Xilinx LogiCORE IP Site License Agreement or the Xilinx LogiCORE IP Project License Agreement, which conform to the terms of the SignOnce IP License/Project 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 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, 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 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 you purchase a license, a product entitlement is added to your Product Licensing Account on the Xilinx Product Download and Licensing site. The Product Licensing Account Administrator for your site will receive an email from Xilinx with instructions on how to access a Full license and a link to access the licensing site. You can obtain a full key through your account administrator, or your administrator can give you access so that you can generate your own keys.

Further details can be found at http://www.xilinx.com/products/ipcenter/ipaccess_fee.htm.

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 available at www.xilinx.com/support/documentation/dt_ise.htm.

XILINX®

Chapter 3

Getting Started

This chapter provides an overview of the example design for PCI 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 12.2 with the following simulators:

- Cadence® Incisive Enterprise Simulator (IES) v9.2
- Mentor Graphics® ModelSim® v6.5c

Generating the Core

To generate a Initiator/Target core for PCI using the Xilinx CORE Generator software:

1. Start the CORE Generator software.

   For help, 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 "Directory Structure," page 15.
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 for PCI 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 either Verilog or VHDL.
     - For Vendor, select Synplicity (or Other for XST).

5. After creating the project, locate the core in the taxonomy tree under Standard Bus Interfaces > PCI.
6. Double-click 32-bit Interface for PCI (Virtex-5/Spartan-6 devices only) 4.13 or 64-bit Initiator/Target for PCI (Virtex-5 only) 4.13 to display the main PCI 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 15.
8. After selecting the desired features and parameters, 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 15.

Unsupported Devices

If you wish to target a device/package combination that is not officially supported (not listed in the *Initiator/Target for PCI 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 tool 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. Spartan-6 is not supported by the PCI/PCI-X UCF Generator.

66 Mhz on Unsupported Devices

Due to the stringent requirements of PCI 66 MHz, Virtex-5 family implementations provided with the product require additional directed routing (DIRT) constraints in the UCF file. These constraints provide exact locations for specific components and nets to guarantee timing. Timing closure for 66 MHz cannot be guaranteed on other devices not listed in DS206, *32-Bit Initiator/Target v3 & v4 for PCI*, and DS205, *64-Bit Initiator/Target v3 & v4 for PCI*.

For 66 MHz designs in devices not listed in the data sheets, the user can attempt to meet timing without the additional DIRT constraints by changing provided 33 MHz timing constraints in the delivered UCF files or UCFs provided by the UCF Generator. UCF Generator provides a vast number of part and package combinations, but does not provide DIRT constraints and does not guarantee timing. Depending on the size of the design relative to the device design, meeting the 66 MHz constraint requirement on devices not listed in the data sheet may be possible. It may also be possible in designs in faster speed grades or by using advanced placement techniques.

For further assistance, please contact Xilinx Design Services or Titanium Dedicated Engineering who may be able to provide assistance creating a custom UCF that will meet timing at 66 MHz.

Directory Structure

This section provides detailed information about the example design, including a description of files and the directory structure generated by the CORE Generator software, 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
Simulation files

Directory and File Contents

The PCI core directories and their associated files are defined below.

<project directory>

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

Table 3-1: **Project Directory**

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

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 3-2: **Component Name Directory**

Name	Description
<project_dir>/<component_name>	
pci_readme.txt	Core name release notes file.

Back to Top

<component name>/doc

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

Table 3-3: **Doc Directory**

Name	Description
<project_dir>/<component_name>/doc	
pci_64_ds205.pdf	64-bit Initiator/Target v3 and v4 for PCI Data Sheet
pci_64_ug262.pdf	Initiator/Target for PCI User Guide
pci_64_gsg260.pdf	Initiator/Target for PCI Getting Started Guide

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 (UCF).
pci_lc.v[hd]	Wrapper file for PCI core netlist, instantiated under <component_name>_top.v[hd].
<component_name>.v[hd]	Black-box file instantiated in pci_lc.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	Synplify project file and synthesis script.
xst.scr	XST synthesis script.
xst.prj	XST project file.

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. Implementation script results are placed in this 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.
`stim_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 PCI 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 IES simulation script.
`wave.sv`	Cadence IES waveform, invoked by **`simulate_ncsim.{bat|sh}`**.
`simulate_mti.do`	Mentor Graphics ModelSim simulation script.
`wave.do`	Mentor Graphics ModelSim waveform display script; invoked by **`simulate_mti.do`**.

Back to Top

simulation/timing

The timing directory contains the 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 IES simulation script.
`wave.sv`	Cadence IES waveform, invoked by `simulate_ncsim.{bat|sh}`.
`simulate_mti.do`	Mentor Graphics ModelSim simulation script.
`wave.do`	Mentor Graphics ModelSim waveform display script; invoked by `simulate_mti.do`.
`simulate_ncsim.{bat | sh}`	Cadence IES simulation script.

Back to Top

XILINX®

Chapter 4

Family Specific Considerations

This chapter provides important design information specific to the core interface for PCI that targets the Virtex®-5 family of devices.

Device Initialization

Immediately after FPGA configuration, both the Initiator/Target core for PCI 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 bus reinitializes the core and three-states all PCI bus signals. This behavior is fully compliant with the *PCI Local Bus Specification*. The core is designed to correctly handle asynchronous resets.

Typically, the user application must be initialized each time the Initiator/Target core for PCI interface is initialized. In this case, use the RST output of the core as the asynchronous reset signal for the user application. If part of the user application requires an initialization capability that is asynchronous to PCI 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, because the global resource is not used. The use of the global reset resource is not recommended.

Configuration Pins

Designers should be aware that PCI 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 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.

Input Delay Buffers

Input delay buffers are used to provide guaranteed hold time on all bus inputs. Virtex-5 implementations make use of the IDELAY input delay buffer primitives. Spartan-6 devices use a combination of IODELAY2 primitive and legacy input buffers to guarantee hold time. 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 which use the IODELAY2 primitive (Spartan-6 FPGAs) do not require an input reference clock for delay calibration. Designs that use IDELAY primitives (Virtex-5 devices) 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, RCLK, 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.

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 have fixed frequency. In this case, you must verify the following:
  - The jitter of the source clock, to determine if it is appropriate for use as an input to a DCM.
  - The DCM configuration, to generate a 200 MHz clock on any appropriate DCM output (CLKFX, CLKDV, and so forth).
  - The jitter of the derived 200 MHz reference clock, to determine if it is appropriate for use as an input to an IDELAYCTRL.
  - 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 documentation. As with the other implementation options, the derived 200 MHz reference clock must be distributed by a global clock buffer to the IDELAYCTRL instances.

Important: 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," page 23 for additional information about the allowed behavior of the PCI bus clock in compliant systems.

Regional Clock Usage

Some Virtex-5 implementations use a regional clock buffer (BUFR) for the PCI bus clock instead of a global clock buffer (BUFG). Use of a regional clock resource greatly improves the pin-to-pin clock to out of the interface while preserving full compliance. (Pin-to-pin clock to out is a silicon (chip) performance parameter important for PCI.)

Designers must be aware of additional constraints imposed by the use of regional clocks. Virtex-5 devices are divided into clock regions. Regional clock signals enter at the center of a given region, and span the region of entry in addition to the region above and the region below. The reach of a regional clock is physically limited to three clock regions. Figure 4-1 illustrates BUFR driving three clock regions. See the Virtex-5 FPGA Data Sheet and Virtex-5 FPGA User Guide for more information about regional clocks.

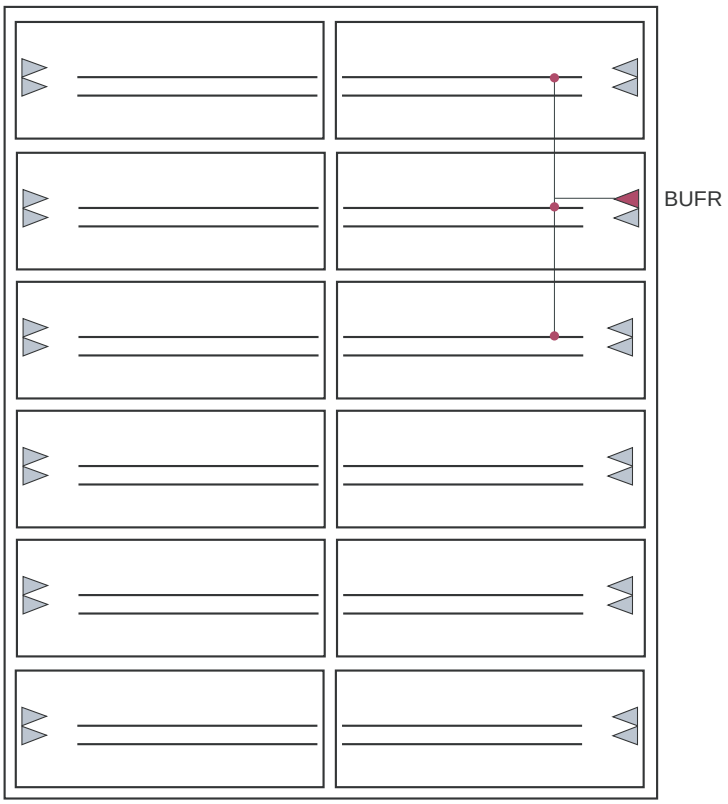

Figure 4-1: **Regional Clocking Illustration**

For designs using regional clocking, the core and those portions of the user application clocked from the PCI bus clock must completely fit inside the three clock regions accessible to the regional clock signal. This restriction limits the number of FPGA resources that may be synchronous with the PCI bus clock. Access to additional logic is available by crossing to another clock domain.

Clock regions are 20 CLBs /40 IOBs tall and one-half the width of the device. With a regional clock span limited to three regions, this yields a maximum of 120 IOB that may be used for a PCI interface. A 64-bit Initiator/Target core for PCI requires 90 IOB, and a 32-bit Initiator/Target core for PCI requires 50 IOB. In some device and package combinations (typically, physically large devices in a relatively low pin-count packages) not all IOB sites are bonded to package pins. This renders some clock regions unusable. This is generally not an issue for 32-bit core interfaces; however, for 64-bit core interfaces, it is a concern.

Bus Clock Usage

The bus clock output provided by the core interface is derived from the bus clock input, and is distributed using a global clock buffer or regional (66 MHz) 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.

Consistent frequency of this clock is not guaranteed. In fact, in a compliant system, the clock may be any frequency, up to and including the maximum allowed frequency, and may change on a cycle-by-cycle basis. Under certain conditions, the 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 it use this clock in the design of interval timers (for example, DRAM refresh counters).

Datapath Output Clock Enable

PCI solutions targeting Spartan-6 devices use a specialized resource called PCILOGICSE to distribute the datapath output clock enable signal. For this reason, IRDY and TRDY must be placed on specific IOB's which have direct access to PCILOGICSE block. Do not change the pin locations of IRDY and TRDY provided in the UCF file.

Electrical Compliance

The Initiator/Target core for PCI targeting Virtex-5 or Spartan-6 devices uses one of two Virtex-5 I/O buffer types, depending on the signaling environment (this selection is made using the wrapper file).

Note: Virtex-5 and Spartan-6 devices are not 5.0 volt tolerant. Do not use these devices in a 5.0 volt signaling environment.

Wrapper files for the 3.3 volt signaling environment use either the PCI33_3 or the PCI66_3 I/O buffers available on Virtex-5 devices. For 3.3 volt signaling in Virtex-5 devices, the VCCO 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 4-2 shows one possible low-cost solution to generate the required 3.0 volt output driver supply for the Virtex-5 devices. Xilinx recommends the use of the circuits shown in Figure 4-2, although other approaches using other regulators are also possible.

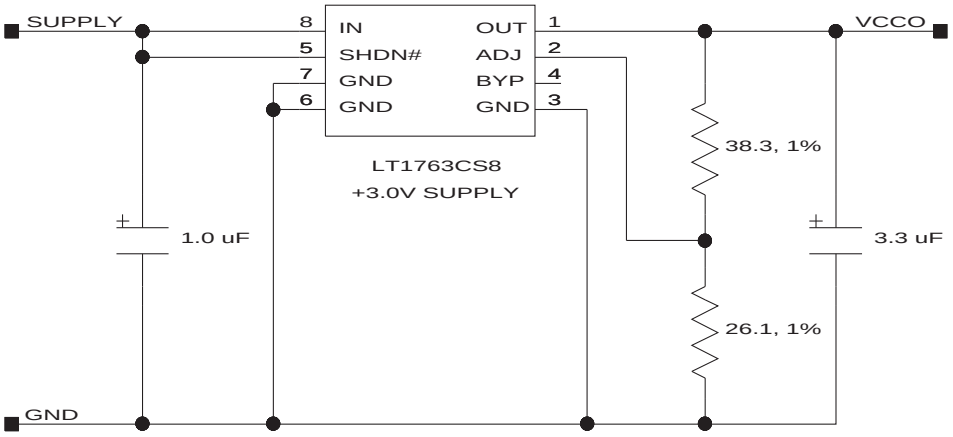

Figure 4-2: **PCI Output Driver VCCO Generation**

The Virtex-5 devices exhibit a 10 pF pin capacity. 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.

Although the Initiator/Target core for PCI does not directly provide the PME# signal for power management event reporting, it may be implemented by the user application. A typical implementation would involve the implementation of the power management capability item in user configuration space, along with a dedicated PME# output on a general purpose I/O pin.

On all device families, if the FPGA power is removed, the general purpose I/O pin appears as a low impedance to ground, and appears to the system as an assertion of PME#. For this reason, implementations that use the PME# signal should employ an external buffering scheme to prevent false assertions of PME# when power is removed from the FPGA device.

For Spartan-6 devices, the *Spartan-6 FPGA Data Sheet* lists the VCCO requirement as 3.3V with a range of -10% to +5% (3.0V to 3.45V). It is required to regulate the VCCO voltage because the PCI Specification specifies the PCI power rail to be 3.3V +/- 10% (3.0V to 3.6V). The regulator must regulate to 3.3 volts to maintain compliance.

Generating Bitstreams

The bitstream generation program, `bitgen`, may issue DRC warnings when generating bitstreams for use in designs for PCI. The number of these warnings varies depending on the configuration options used for the 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 UCF.

Be aware that the bitgen options provided with the example design are for reference only. The actual options used will depend on the desired FPGA configuration method and clock rate of your complete design, as implemented on a board. Carefully consider the following configuration time requirements when selecting a configuration method and clock rate.

- Any designs that do not automatically sense the bus width must be configured within (100 ms + 2^{25} bus clocks) after the bus power rails become valid.
- Any designs that must sense the bus width must be configured within 100 ms after the bus power rails become valid.
- 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 IES
- Mentor Graphics ModelSim

Cadence IES

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

Verilog

1. Navigate to the functional simulation directory:

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

2. View the `simulate_ncsim.sh` file.

   This file lists commands that invoke IES 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 PCI Core Simulation Model'
   ncvlog -work work .. /../../<component_name>.v

   echo 'Compiling PCI Example Design'
   ncvlog -work work ../../example_design/userapp.v
   ncvlog -work work ../../example_design/pci_lc.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. View the `simulate_ncsim.sh` file.

   This file lists commands that invoke IES 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 PCI Core Simulation Model'
ncvhdl -v93 -work work .. /../../<component_name>.vhd

echo 'Compiling PCI Example Design'
ncvhdl -v93 -work work ../../example_design/userapp.vhd
ncvhdl -v93 -work work ../../example_design/pci_lc.vhd
ncvhdl -v93 -work work ../../example_design/<component_name>_top.vhd

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

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/pci_lc.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
```

   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. 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, the user can invoke ModelSim and run the script directly from the system prompt:

   `vsim -do simulate_mti.do`

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/pci_lc.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, 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. 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, the user can invoke ModelSim and run the script directly from the system prompt:

   `vsim -do simulate_mti.do`

Chapter 6

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:

- Xilinx XST™

Userapp consists of the design files listed in Table 6-1.

Table 6-1: ***Userapp*** **Example Design Files**

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

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

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.

XILINX®

Chapter 7

Timing Simulation

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

- Cadence Incisive Enterprise Simulator (IES) v9.2 and above
- Mentor Graphics ModelSim v6.5c and above

Cadence IES

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

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 the following:

   `<component_name>/simulation/timing`

4. To run the simulation, type the following:

   `simulate_ncsim.sh`

   (Invoke simulate_ncsim.bat on Windows/NT platforms.)

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

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, the user can invoke ModelSim and run the script directly from the system prompt:

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