LogiCORE™ IP Initiator/Target v3.167 for PCI™

Getting Started Guide

UG157 June 24, 2009

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
6/01/00	1.0	Initial Xilinx release
4/11/01	2.0	Revised book formatting to use FrameMaker 6.0 features
6/24/02	3.0	Initial Xilinx release of corporate-wide common template set.
11/11/04	3.5	Added Installation and Licensing chapters.
12/01/04	3.6	Updated to include Virtex-4 support.
03/7/05	3.7	Updated to ISE 7.1i and build number 3.0.145
5/13/05	4.0	Updated to ISE 7.1i SP 2, build number 3.0.150
8/31/05	5.0	Updated to ISE 7.1i SP3 and build number 3.0.151
9/12/05	6.0	Updated to build 3.0.152, removed instructions to verify directory structure.
1/18/06	7.0	Updated to ISE 8.1i, build 3.0.155.
2/14/16	8.0	Updated to add SP2 to ISE 8.1i, build to 158, removed 2S100 from Supported Devices Table in Chapter 3.
7/13/06	9.0	Updated to ISE v8.21, build to 160
2/15/07	10.0	Updated to build 161, tools, ISE version to 9.2i
5/17/07	10.5	Fixed all references to PCI to conform to PCI-SIG trademark guidelines. Advanced support for Cadence IUS to v5.7.
8/08/07	11.0	Updated for IP1 Jade Minor release, advanced version to 3.2, device support. Updated VCCO requirements for Spartan-3 FPGA family.
10/10/07	11.5	Updated for IP2 Jade Minor release, add IP to LogiCORE references, replaced legal disclaimer with current issue.

Date	Version	Revision
3/24/08	12.0	Updated for IP0K release.
9/19/08	12.5	Updated core to v3.2 and added support for ISE v10.1 Service Pack 3. Added Chapter 7, "Timing Simulation."
4/24/09	13.0	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	13.5	Updated to support ISE v11.2. Removed support for Spartan-6 devices.

Table of Contents

Schedule of Figures

Chapter 1: Getting Started

Chapter 2: Licensing the Core

Chapter 3: Family Specific Considerations

Chapter 4: Functional Simulation

Chapter 5: Synthesizing a Design

Chapter 6: Implementing a Design

Chapter 7: Timing Simulation

About This Guide

The *LogiCORE™ IP Initiator/Target v3.167 for PCI Getting Started Guide* provides a fully-verified, pre-implemented PCI bus interface available in both 32-bit and 64-bit versions. It contains information about the supported design flows for 32-bit and 64-bit cores based on the Virtex® and Spartan® FPGA architectures, and provides an example design in Cadence® IUS.

Guide Contents

This manual contains the following chapters:

- Chapter 1, "Getting Started" describes the core and related information, getting technical support, and submitting feedback to Xilinx.
- Chapter 2, "Licensing the Core" provides information about installing and licensing the core.
- Chapter 3, "Family Specific Considerations" provides information about design considerations specific to the core, targeting Virtex and Spartan devices.
- Chapter 4, "Functional Simulation" describes how to simulate the example design using the supported functional simulation tools, including Cadence® IUS and Mentor Graphics® ModelSim®.
- Chapter 5, "Synthesizing a Design" describes how to synthesize the example design using the supported synthesis tools, including Synplicity® Synplify®, Exemplar LeonardoSpectrum, and Xilinx XST.
- Chapter 6, "Implementing a Design" describes how to implement the example design using the supported FPGA implementation tools included with the ISE® software.
- Chapter 7, "Timing Simulation" describes how to perform timing simulation using the supported post-route timing simulation tools, including Cadence IUS and Mentor Graphics ModelSim.

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 towww.xilinx.com for the latest speed files.

XILINX®

Chapter 1

Getting Started

The Initiator/Target core for PCI provides a fully verified, pre-implemented PCI bus interface available in both 32-bit and 64-bit versions with support for operation at 33 MHz and 66 MHz. This guide defines the supported design flows for both the 32-bit and 64-bit interfaces targeting devices based on the Virtex® and Spartan® FPGA architectures. An example design is provided in Cadence IUS that lets you simulate, synthesize, and implement the interface to understand the PCI design flow.

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 11.2 with applicable Service Pack

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 Example Design

The example design is a simple user application provided as a training tool and design flow test. The example design consists of the user application *Ping*, and supporting files for simulation and implementation. The 32-bit core ships with the *ping32* design, and the 64-bit core ships with the *ping64* design. The examples in this document reference *ping64*. If you are using the 32-bit core, substitute *ping32* for *ping64*.

The *Ping* design includes a test bench capable of generating simple read and write transactions. This stimulation generation capability is used to set up the configuration space of the design, and then perform some simple transactions. In addition, a special configuration file is provided, and the test bench makes assumptions about the size and number of base address registers used.

Users may change the core options related to implementation, that is, the options that relate to the selected FPGA architecture. However, users must not change core options that

alter the functional behavior of theInitiator/Target core for PCI; such changes cause unpredictable results in the simulation of the example design. For custom designs, users have the flexibility to change the core configuration as described in the *LogiCORE IP Initiator/Target v3.167 for PCI User Guide*.

Step-by-step instructions using supported design tools are provided to simulate, synthesize, and implement the *Ping* example design.

Additional Documentation

For more information about the Initiator/Target core for PCI, see the following documents, located in the zip file:

- *Initiator/Target v3.167 for PCI Release Notes*
- *Initiator/Target v3.167 for PCI User Guide*

Further information is available in the Mindshare PCI System Architecture text, and the PCI Local Bus Specification, available from the PCI Special Interest Group site.

Technical Support

For technical support, visit www.xilinx.com/support. Questions are routed to a team of engineers with expertise using the Initiator/Target core for PCI.

Xilinx provides technical support for use of this product as described in the *Initiator/Target v3.167 for PCI User Guide* and the *Initiator/Target v3.167 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 Initiator/Target core for PCI and the documentation supplied with the core.

PCI Interface Core

For comments and suggestions about the Initiator/Target core for PCI, please submit a WebCase from www.xilinx.com/support/clearexpress/websupport.htm. 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/clearexpress/websupport.htm. Be sure to include the following information:

- Document title and 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, 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 two licensing options. After installing the required Xilinx ISE software and IP Service Packs, choose a license option.

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 full system hardware and full license keys.

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** and follow the instructions to generate a license.

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.

Chapter 3

Family Specific Considerations

This chapter provides important design information specific to the Initiator/Target core for PCI targeting the Virtex and Spartan families of devices.

Design Support

Table 3-1 provides a list of supported device and interface combinations. Each entry in the table consists of a device, a bus interface type, and two or three specific implementation files.

Table 3-1: **Device and Interface Selection Table**

Supported Device	Bus Type	Wrapper File	Constraints File
Spartan-3			
3S1000-FG456-4C/I	33 MHz 3.3V 32-bit	pcim_lc_33_3_s	`3s1000fg456_32_33.ucf`
3S1000-FG456-4C/I	33 MHz 3.3V 64-bit	pcim_lc_33_3_s	`3s1000fg456_64_33.ucf`
Spartan-3E			
3S500E-FT256-4C/I	33 MHz 3.3V 32-bit	pcim_lc_33_3_s	`3s500eft256_32_33.ucf`
3S500E-FT256-5C/I	66 MHz 3.3V 32-bit	pcim_lc_66_3_s	`3s500eft256_32_66.ucf`
3S1200E-FG400-4C/I	33 MHz 3.3V 32-bit	pcim_lc_33_3_s	`3s1200efg400_32_33.ucf`

Table 3-1: **Device and Interface Selection Table *(Continued)***

Supported Device	Bus Type	Wrapper File	Constraints File
3S1200E-FG400-4C/I	33 MHz 3.3V 64-bit	pcim_lc_33_3_s	`3s1200efg400_64_33.ucf`
3S1200E-FG400-5C[a]	66 MHz 3.3V 32-bit	pcim_lc_66_3_s	`3s1200efg400_32_66.ucf`
3S1200E-FG400-5C[a]	66 MHz 3.3V 64-bit	pcim_lc_66_3_s	`3s1200efg400_64_66.ucf`
Spartan-3A			
3S400A-FG400-4C/I	33 MHz 3.3V 32-bit	pcim_lc_33_3_s	`3s400afg400_32_33.ucf`
3S400A-FG400-4C/I	33 MHz 3.3V 64-bit	pcim_lc_33_3_s	`3s400afg400_64_33.ucf`
3S400A-FG400-5C/I	66 MHz 3.3V 32-bit	pcim_lc_66_3_s	`3s400afg400_32_66.ucf`
3S400A-FG400-5C/I	66 MHz 3.3V 64-bit	pcim_lc_66_3_s	`3s400afg400_64_66.ucf`
3S700A-FG400-4C/I	33 MHz 3.3V 32-bit	pcim_lc_33_3_s	`3s700afg400_32_33.ucf`
3S700A-FG400-4C/I	33 MHz 3.3V 64-bit	pcim_lc_33_3_s	`3s700afg400_64_33.ucf`
3S700A-FG400-5C/I	66 MHz 3.3V 32-bit	pcim_lc_66_3_s	`3s700afg400_32_66.ucf`
3S700A-FG400-5C/I	66 MHz 3.3V 64-bit	pcim_lc_66_3_s	`3s700afg400_64_66.ucf`

Table 3-1: **Device and Interface Selection Table *(Continued)***

Supported Device	Bus Type	Wrapper File	Constraints File
3S700A-FG484-4C/I	33 MHz 3.3V 32-bit	pcim_lc_33_3_s	`3s700afg484_32_33.ucf`
3S700A-FG484-4C/I	33 MHz 3.3V 64-bit	pcim_lc_33_3_s	`3s700afg484_64_33.ucf`
3S700A-FG484-5C/I	66 MHz 3.3V 32-bit	pcim_lc_66_3_s	`3s700afg484_32_66.ucf`
3S700A-FG484-5C/I	66 MHz 3.3 64-bit	pcim_lc_66_3_s	`3s700afg484_64_66.ucf`
3S1400A-FG484-4C/I	33 MHz 3.3V 32-bit	pcim_lc_33_3_s	`3s1400afg484_32_33.ucf`
3S1400A-FG484-4C/I	33 MHz 3.3V 64-bit	pcim_lc_33_3_s	`3s1400afg484_64_33.ucf`
3S1400A-FG484-5C/I	66 MHz 3.3V 32-bit	pcim_lc_66_3_s	`3s1400afg484_32_66.ucf`
3S1400A-FG484-5C/I	66 MHz 3.3V 64-bit	pcim_lc_66_3_s	`3s1400afg484_64_66.ucf`
3S1400A-FG676-4C/I	33 MHz 3.3V 32-bit	pcim_lc_33_3_s	`3s1400afg676_32_33.ucf`
3S1400A-FG676-4C/I	33 MHz 3.3V 64-bit	pcim_lc_33_3_s	`3s1400afg676_64_33.ucf`
3S1400A-FG676-5C/I	66 MHz 3.3V 32-bit	pcim_lc_66_3_s	`3s1400afg676_32_66.ucf`

Table 3-1: **Device and Interface Selection Table *(Continued)***

Supported Device	Bus Type	Wrapper File	Constraints File
3S1400A-FG676-5C/I	66 MHz 3.3 64-bit	pcim_lc_66_3_s	`3s1400afg676_64_66.ucf`
Spartan-3AN			
3S400AN-FGG400 -4C/I	33 MHz 3.3V 32-bit	pcim_lc_33_3_s	`3s400anfgg400_32_33.ucf`
3S400AN-FGG400 -5C/I	66 MHz 3.3V 32-bit	pcim_lc_66_3_s	`3s400anfgg400_32_66.ucf`
3S700AN-FGG484 -4C/I	33 MHz 3.3V 32-bit	pcim_lc_33_3_s	`3s700anfgg484_32_33.ucf`
3S700AN-FGG484 -5C/I	66 MHz 3.3V 32-bit	pcim_lc_66_3_s	`3s700anfgg484_32_66.ucf`
3S1400AN-FGG676 -4C/I	33 MHz 3.3V 32-bit	pcim_lc_33_3_s	`3s1400anfgg676_32_33.ucf`
3S1400AN-FGG676 -5C/I	66 MHz 3.3V 32-bit	pcim_lc_66_3_s	`3s1400anfgg676_32_66.ucf`
Spartan-3ADSP			
3SD1800A-FG676-4C/I	33 MHz 3.3V 32-bit	pcim_lc_33_3_s	`3sd1800afg676_32_33.ucf`
3SD1800A-FG676-4C/I	33 MHz 3.3V 64-bit	pcim_lc_33_3_s	`3sd1800afg676_64_33.ucf`
3SD1800A-FG676-5C/I	66 MHz 3.3V 32-bit	pcim_lc_66_3_s	`3sd1800afg676_32_66.ucf`

Table 3-1: **Device and Interface Selection Table *(Continued)***

Supported Device	Bus Type	Wrapper File	Constraints File
3SD1800A-FG676-5C/I	66 MHz 3.3V 64-bit	pcim_lc_66_3_s	3sd1800afg676_64_66.ucf
3SD3400A-FG676-4C/I	33 MHz 3.3V 32-bit	pcim_lc_33_3_s	3sd3400afg676_32_33.ucf
3SD3400A-FG676-4C/I	33 MHz 3.3V 64-bit	pcim_lc_33_3_s	3sd3400afg676_64_33.ucf
3SD3400A-FG676-5C/I	66 MHz 3.3V 32-bit	pcim_lc_66_3_s	3sd3400afg676_32_66.ucf
3SD3400A-FG676-5C/I	66 MHz 3.3V 64-bit	pcim_lc_66_3_s	3sd3400afg676_64_66.ucf
Virtex-4 Devices			
4VFX20-FF672-10C/I global clock	33 MHz 3.3V 32-bit	pcim_lc_33_3_g	4vfx20ff672_32_33g.ucf
4VFX20-FF672-10C/I global clock	33 MHz 3.3V 64-bit	pcim_lc_33_3_g	4vfx20ff672_64_33g.ucf
4VFX20-FF672-10C/I regional clock	33 MHz 3.3V 64-bit	pcim_lc_33_3_r	4vfx20ff672_64_33r.ucf
4VFX20-FF672-10C/I regional clock	33MHz 3.3V 32-bit	pcim_lc_33_3_r	4vfx20ff672_32_33r.ucf
4VFX20-FF672-11C/I regional clock	66 MHz 3.3V 32-bit	pcim_lc_66_3r	4vfx20ff672_32_66r.ucf
4VFX20-FF672-11C/I regional clock	66 MHz 3.3V 64-bit	pcim_lc_66_3_r	4vlx20ff672_64_66r.ucf

Table 3-1: **Device and Interface Selection Table *(Continued)***

Supported Device	Bus Type	Wrapper File	Constraints File
4VLX25-FF668-10C/I global clock	33 MHz 3.3V 32-bit	pcim_lc_33_3_g	4vlx25ff668_32_33g.ucf
4VLX25-FF668-10C/I regional clock	33 MHz 3.3V 64-bit	pcim_lc_33_3_r	4vlx25ff668_64_33r.ucf
4VLX25-FF668-10C/I regional clock	33MHz 3.3V 32-bit	pcim_lc_33_3_r	4vlx25ff668_32_33r.ucf
4VLX25-FF668-11C/I regional clock	66 MHz 3.3V 32-bit	pcim_lc_66_3r	4vlx25ff668_32_66r.ucf
4VLX25-FF668-11C/I regional clock	66 MHz 3.3V 64-bit	pcim_lc_66_3_r	4vlx25ff668_64_66r.ucf
4VLX25-FF668-10C/I global clock	33 MHz 3.3V 64-bit	pcim_lc_33_3_g	4vlx25ff668_64_33g.ucf
4VSX35-FF668-10C/I global clock	33 MHz 3.3V 32-bit	pcim_lc_33_3_g	4vsx35ff668_32_33g.ucf
4VSX35-FF668-10C/I regional clock	33 MHz 3.3V 64-bit	pcim_lc_33_3_r	4vsx35ff668_64_33r.ucf
4VSX35-FF668-10C/I regional clock	33MHz 3.3V 32-bit	pcim_lc_33_3_r	4vsx35ff668_32_33r.ucf
4VSX35-FF668-11C/I regional clock	66 MHz 3.3V 32-bit	pcim_lc_66_3r	4vsx35ff668_32_66r.ucf
4VSX35-FF668-11C/I regional clock	66 MHz 3.3V 64-bit	pcim_lc_66_3_r	4vsx35ff668_64_66r.ucf

Table 3-1: **Device and Interface Selection Table *(Continued)***

Supported Device	Bus Type	Wrapper File	Constraints File
4VSX35-FF668-10C/I global clock	33 MHz 3.3V 64-bit	pcim_lc_33_3_g	`4vsx35ff668_64_33g.ucf`

a. The XC3S1200E requires post-processing of the routed NCD file to meet hold-time requirements for 66 MHz cores. See Chapter 6, "Implementing a Design," for more information.

See the product release notes included with the core for a complete directory structure and file list.

Wrapper Files

Wrapper files contain an instance of the Initiator/Target core for PCI and the instances of all I/O elements used by the core. Each wrapper file is specific to a particular PCI bus signaling environment.

The wrapper files, located in the `<Install Path>/hdl/src/wrap` directory, are actually variations of the `pcim_lc.hdl` file located in the `<Install Path>/hdl/src/xpci` directory.

When starting a new design, copy the appropriate wrapper file from the `wrap/` directory into the `xpci/` directory, and rename it `pcim_lc.hdl`.

Constraints Files

The user constraints files (UCFs) contain various constraints required for the Initiator/Target core for PCI, and must always be used while processing a design. Each constraints file is specific to a particular device and core—use the appropriate constraints file from the `<Install Path>/hdl/src/ucf` directory when processing designs with the Xilinx implementation tools.

Directed Routing

With the exception of those targeted for Virtex-4 devices, UCFs for 66 MHz designs contain directed-routing constraints. In these cases, the directed routing ensures that the interface performance requirements are met. It is important to verify routing success in a design that uses directed routing; you can determine this by checking the PAR report file (PAR) for the message:

```
INFO:ParHelpers:199 - All "EXACT" mode Directed Routing constrained
nets successfully routed.
```

Unsupported Devices

If you wish to target a device/package combination that is not officially supported (not listed in Table 3-1), 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.

- For Spartan-3 devices, the UCF Generator for PCI is available on the web from the Xilinx PCI Lounge:

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

- For Spartan-3A, Spartan-3ADSP and Virtex-4 devices, 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.

66 Mhz on Unsupported Devices

Due to the stringent requirements of PCI 66 MHz, Spartan 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 Virtex-4 devices, DIRT constraints are not required. However, only the Virtex-4 devices listed in the data sheets as 66 MHz capable are guaranteed to meet the 66 MHz constraints. 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.

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 and Spartan 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.

Bus Width Detection

An Initiator/Target core for PCI that provides a 64-bit datapath must know if it is connected to a 64-bit bus or a 32-bit bus. The SLOT64 signal is an input to the PCI64 interface for this purpose. The PCI bus specification provides a mechanism for PCI agents to determine the width of the bus by sampling the state of the REQ64# signal at the rising edge of RST#.

In embedded systems, where the bus width is known by design, the user application can simply drive SLOT64 with the appropriate value. Note that SLOT64 must never be driven with a static value; it should always be driven from the output of a flip-flop.

In designs for open systems, the bus width is not known in advance. In this case, include a separate latch or flip-flop, external to the FPGA, to sample REQ64#. Figure 3-1 shows how this can be accomplished.

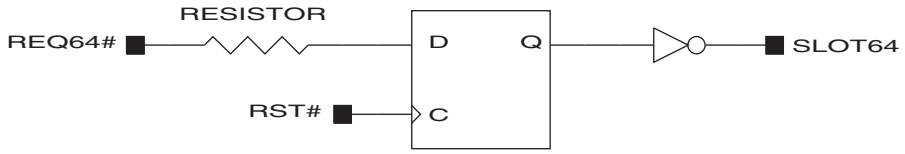

Figure 3-1: **Sample SLOT64 Generation**

Although this technique is not technically compliant with the PCI specification, (due to the extra loading on REQ64# and RST#) the use of a large series resistor helps minimize this effect. The inverter may be pushed into the FPGA.

An alternate method is to push the entire circuit into the FPGA and use the REQ64Q_N and RST signals provided to the user application. This method requires that the FPGA be fully configured by the rising edge of RST#. When SLOT64 is deasserted, the 64-bit Initiator/Target core for PCI automatically three-states the 64-bit extension signals. In this situation, the 64-bit extension signals are undriven, which may result in additional power consumption by the input buffers.

If the additional power consumption is of concern due to design requirements, consider changing the "Disabled Extension Drive" option in the HDL configuration file. This option, when enabled, forces the 64-bit Initiator/Target core for PCI to actively drive the extension signals when SLOT64 is deasserted.

Note: Although this option may reduce power consumption, it creates an electrically objectionable situation. When a 64-bit card is installed in a 32-bit slot, the 64-bit bus extension is floating in free space and unprotected from roaming screwdrivers.

Datapath Output Clock Enable

The Initiator/Target core for PCI uses one of the following methods to generate and distribute the datapath output clock enable signal:

- Specialized device resources: the PCIIOBs, PCILOGIC, and PCI_CE

- Generic device resources: IOBs, LUTs, and general purpose routing

The specialized device resources offer higher and more predictable performance. However, this resource is not available in all device families. When used, the core is constrained to the center left or center right side of the FPGA device, and the number of Initiator/Target core for PCI interface instances is limited to two. The generic device resources, while lower performance and less predictable, offer greater flexibility.

Table 3-2: **Output Clock Enable Settings**

Implementation	Output Clock Enable Setting
Spartan-3, Spartan-3A, Spartan-3E Devices (33 MHz)	"1" (generic)
Spartan-3, Spartan-3A, Spartan-3E Devices (66 MHz)	"0" (special)
Virtex-4 Devices(all)	"1" (generic)

Table 3-2 shows the correct output clock enable configurations. To use generic device resources, set the CFG[251] bit in the HDL configuration file to logic one. To use specialized device resources, set the CFG[251] bit in the HDL configuration file to logic zero.

Input Delay Buffers

Input delay buffers are used to provide guaranteed hold time on all bus inputs. Where possible, the Initiator/Target core for PCI targeting Virtex 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.

Some implementations use alternate delay buffers, selected by using the `CFG[248:245]` bits in the HDL configuration file. Table 3-3 shows the required settings for `CFG[248:245]` in the HDL configuration file.

Table 3-3: **Delay Buffer Settings**

Implementation	Delay Setting
Spartan-3, Spartan-3E, Spartan-3A, Spartan-3A DSP, and Spartan-3AN Devices (All)	"0000"
Virtex-4 Devices (Global Clock)	"0000"
Virtex-4 Devices (Regional Clock) 66 MHz	"1000"
Virtex-4 Devices (Regional Clock) 33 MHz	"0000"

You must set `CFG[248:245]` appropriately for the selected implementation. While the default setting is used for most implementations, failure to match the settings to the selected implementation may result in hardware failures.

Virtex-4 implementations 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. Be sure to observe the settings shown in Table 3-3 and use the appropriate UCF for the design.

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, 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-4 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" for additional information about the allowed behavior of the PCI bus clock in compliant systems.

Regional Clock Usage

Some Virtex-4 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-4 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 3-2 illustrates BUFR driving three clock regions. See the *Virtex-4 FPGA Data Sheet* and *Virtex-4 FPGA User Guide* for more information about regional clocks.

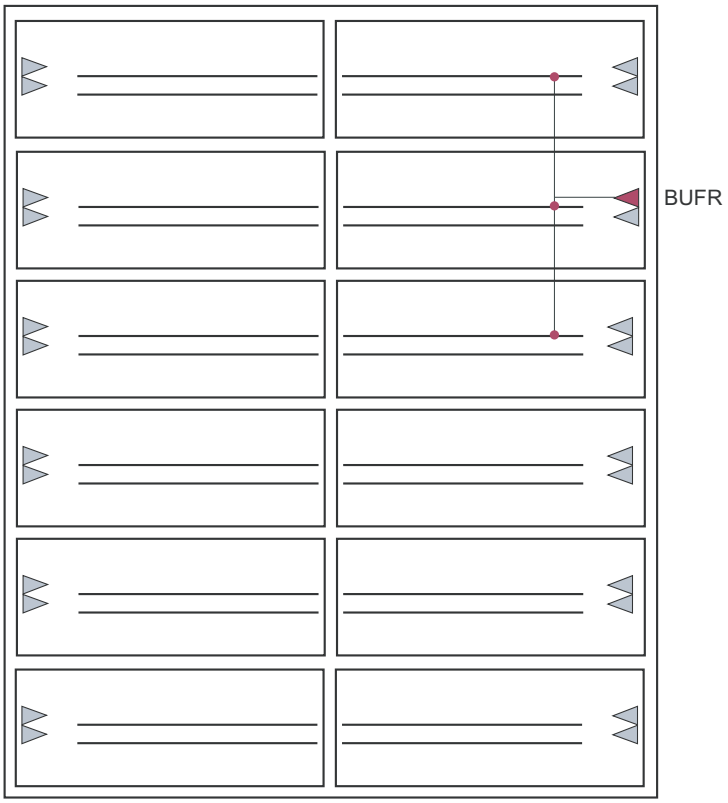

Figure 3-2: **Regional Clocking Illustration**

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 16 CLB / 32 IOB tall and one-half the width of the device. With a regional clock span limited to three regions, this yields a maximum of 96 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.

Table 3-4 defines all physically possible 64-bit core interfaces in various Virtex-4 device and package combinations using regional clocks. Note that this does not apply to Virtex-4 designs using global clocks.

Table 3-4: **Virtex-4 Device and Package 64-bit Interfaces**

Package	Device	64-bit Interfaces
SF363	LX15	none
	LX25	none
	FX12	none

Table 3-4: **Virtex-4 Device and Package 64-bit Interfaces**

Package	Device	64-bit Interfaces
FF668	LX15	2
	LX25	4
	LX40	2
	LX60	2
	FX12	2
	SX25	2
	SX35	4
FF1148	LX40	4
	LX60	4
	LX80	6
	LX100	6
	LX160	6
	SX55	4
FF1513	LX100	8
	LX160	8
	LX200	8
FF672	FX20	2
	FX40	none
	FX60	none
FF1152	FX40	4
	FX60	4
	FX100	4
FF1517	FX100	6
	FX140	6
FF1760	FX140	8

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 (Virtex-4 devices at 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).

Electrical Compliance

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

Note: Virtex-4, Spartan-3, Spartan-3E, Spartan-3A, Spartan-3A DSP, and Spartan-3AN 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-4, Spartan-3, Spartan-3E, Spartan-3A, Spartan-3A DSP, and Spartan-3AN. Observe the relevant specifications in the device data sheet.

In the *PCI Local Bus Specification*, the specifications for the 3.3 volt signaling environment state V_{IL} and V_{IH} as a function of VCC. This may be considered the 3.3 volt system supply.When the 2.5 volt and 3.3 volt supplies are at their opposite extremes, the 3.3 volt V_{IL} or V_{IH} specifications will be violated. The violation is only technical, and will not affect functionality. The V_{IL} or V_{IH} will not venture beyond the parameters stated in the *PCI Local Bus Specification* to affect noise margins significantly. For all supply combinations, V_{IL} will always be within 35 mV of the specification, and V_{IH} will be within 75 mV of the specification. They cannot both be out of specification simultaneously.

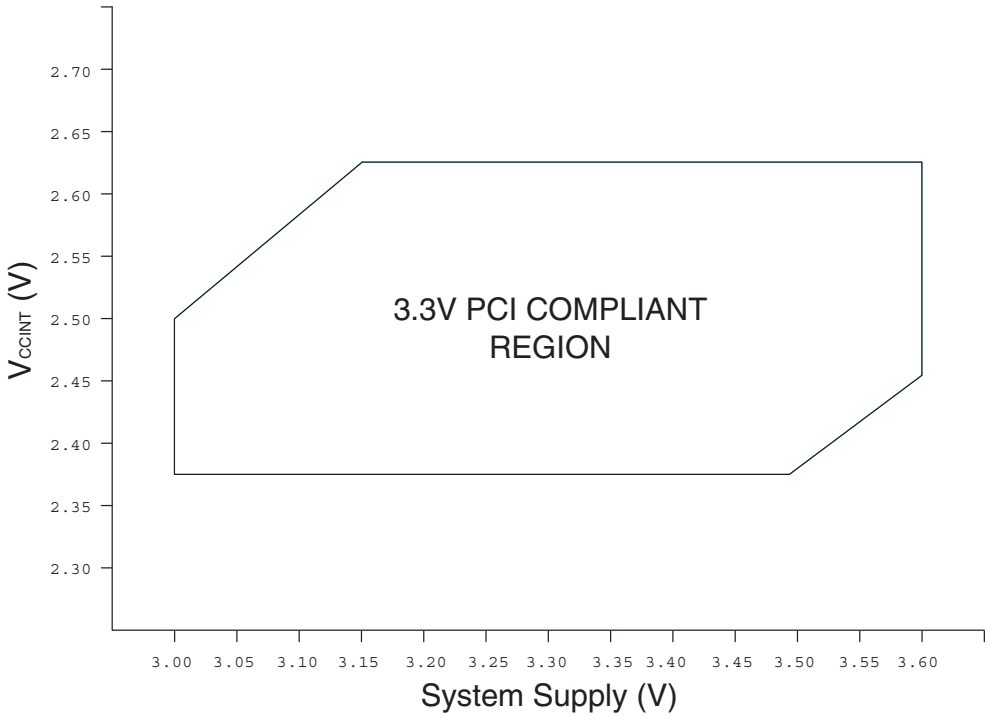

Figure 3-3: **Relationship For 3.3V Input Buffer Compliance**

Figure 3-3 shows the small range of supply voltage values where V_{IL} or V_{IH} are technically non-compliant. Note that this may occur with any PCI device if the input buffer supply voltage is different from the supply voltage of the driving device. For best results, use a high precision voltage regulator to generate V_{CCINT}.

For 3.3 volt signaling in Virtex-4 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*.

For Spartan-3 and Spartan-3E devices, the *Spartan-3 FPGA Data Sheet* and the *Spartan-3E FPGA Data Sheet* list the VCCO requirement as 3.3V with a range of -10% to +5% (3.0V to 3.45V). For existing Spartan-3 and Spartan-3E designs using a regulated VCCO to 3.0 volts, no changes are needed, as this continues to be compliant. For new Spartan-3 and Spartan-3E devices for PCI, it is still necessary to regulate VCCO because the PCI Specification specifies the PCI power rail to be 3.3 volts +/-10% (3.0V to 3.6V). Due to this, a regulator is required for Spartan-3 and Spartan-3E. However, the voltage can now be regulated to 3.3 volts instead of 3.0 volts. See XAPP457 for more information on powering Vcco for Spartan-3 family devices.

For Spartan-3A, Spartan-3A DSP and Spartan-3AN, no VCCO regulation is necessary, as Spartan-3A device PCI buffer VCCO requirements are 3.3V +/- 10%. See the *Spartan-3A FPGA Data Sheet* for more information on device electrical characteristics. VCCO can be connected directly to the PCI 3.3 volt power rail.

Figure 3-4 shows one possible low-cost solution to generate the required 3.0 volt output driver supply for Virtex-4 devices. Figure 3-5 shows the same regulator being used to generate the 3.3 volt output driver supply for the Spartan-3 and Spartan-3E devices. These figures show the different resistor values used for the resistor divider network to create the output supply voltage. Xilinx recommends the use of the circuits shown in Figure 3-4 and Figure 3-5, although other approaches using other regulators are also possible.

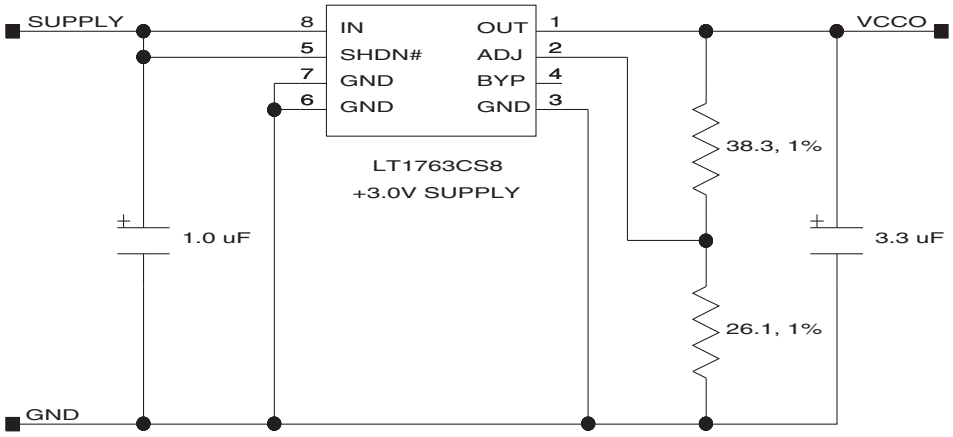

Figure 3-4: **3.0 Volt Output Driver VCCO Generation**

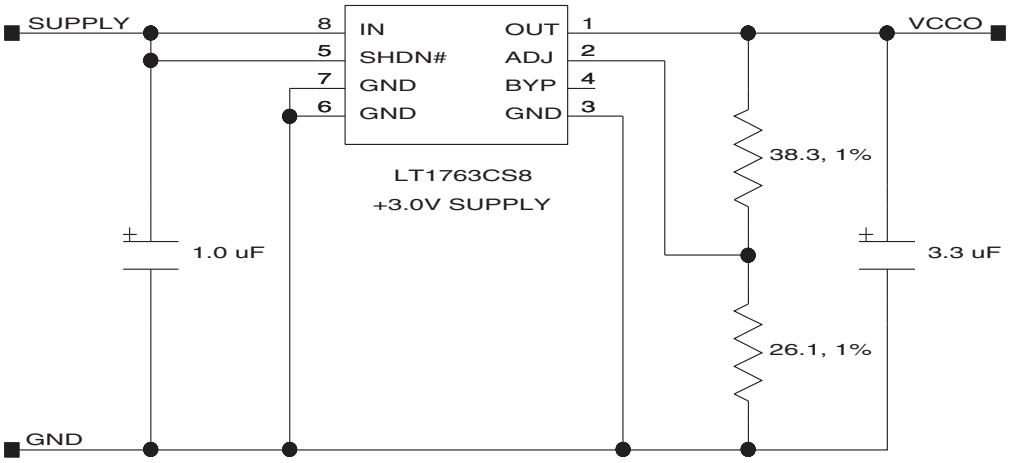

Figure 3-5: **3.3 Volt Output Driver VCCO Generation for Spartan-3 and Spartan-3E**

The Virtex-4, Spartan-3A, Spartan-3A DSP, Spartan-3AN, Spartan-3, and Spartan-3E devices 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.

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.

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.

For some older 66 MHz designs, `bitgen` must be run with a special option to change the behavior of a global clock buffer used in the design:

```
bitgen -g Gclkdel<buf>:<opt> pcim_top_routed.ncd
```

This option is used to introduce additional delay on a global clock net. It is important to note that this additional delay is observable on the CLK output of the core interface, which is supplied to the user application. Timing constraints for the user application must be generated with this in mind. See the product release notes for additional information about the use and implications of this required option.

Chapter 4

Functional Simulation

This chapter describes how to simulate the *ping64* example design with global clocks using the supported functional simulation tools. For the 32-bit Initiator/Target core for PCI, substitute *ping32* for *ping64*. If you are using a design with reference clocks, substitute with *pcim_top_r* and *ping_tb* with *ping_tb_r*.

Supported functional simulation tools include

- Cadence IUS
- Mentor Graphics ModelSim

Cadence IUS

Before attempting functional simulation, ensure that the IUS environment is properly configured.

1. To start, move into the functional simulation directory:

   `cd <Install Path>/verilog/example/func_sim`

2. Edit the `ping_tb.f` file. This file lists command line arguments for IUS, and is shown below:

```
../source/ping_tb.v
../source/stimulus.v
../source/busrecord.v
../source/dumb_arbiter.v
../source/dumb_targ32.v
../source/dumb_targ64.v
../source/pcim_top.v
../source/ping.v
../source/cfg_ping.v
../source/glbl.v
../../src/xpci/pci_lc_i.v
../../src/xpci/pcim_lc.v
+libext+.vmd+.v
-y <Xilinx Install Path>/verilog/src/unisims
-y <Xilinx Install Path>/verilog/src/simprims
```

3. Modify the library search path by changing `<Xilinx Install Path>` to match the Xilinx installation directory, and then save the file.

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 core interface:

```
../source/glbl.v
../../src/xpci/pci_lc_i.v
../../src/xpci/pcim_lc.v
+libext+.vmd+.v
-y <Xilinx Install Path>/verilog/src/unisims
-y <Xilinx Install Path>/verilog/src/simprims
```

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. To run the IUS simulation, enter the following:

   `ncverilog -f ping_tb.f`

IUS processes the simulation files and exits. The test bench prints status messages to the console. After the simulation completes, view the `ncverilog.log` file to check for errors.

The Simvision browser may be used to view the simulation results.

5. If desired, start Simvision with the following command:

   `simvision`

Mentor Graphics ModelSim

Before attempting functional simulation, ensure that the ModelSim environment is properly configured.

Verilog

1. Navigate to the functional simulation directory:

   `cd <Install Path>/verilog/example/func_sim`

2. Edit the `ping_tb.f` file. This file lists command line arguments, and is shown below:

```
../source/ping_tb.v
../source/stimulus.v
../source/busrecord.v
../source/dumb_arbiter.v
../source/dumb_targ32.v
../source/dumb_targ64.v
../source/pcim_top.v
../source/ping.v
../source/cfg_ping.v
../source/glbl.v
../../src/xpci/pci_lc_i.v
../../src/xpci/pcim_lc.v
+libext+.vmd+.v
-y <Xilinx Install Path>/verilog/src/unisims
-y <Xilinx Install Path>/verilog/src/simprims
```

3. Modify the library search path by changing `<Xilinx Install Path>` to match the Xilinx installation directory and then save the file.

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 interface:

```
../source/glbl.v
../../src/xpci/pci_lc_i.v
../../src/xpci/pcim_lc.v
+libext+.vmd+.v
-y <Xilinx Install Path>/verilog/src/unisims
-y <Xilinx Install Path>/verilog/src/simprims
```

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:

   `<Install Path>/verilog/example/func_sim`

5. To run the simulation:

   `do modelsim.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 <Install Path>/vhdl/example/func_sim`

2. View the `ping.files` file. This file lists the individual source files required, and is shown below:

```
../../src/xpci/pci_lc_i.vhd
../../src/xpci/pcim_lc.vhd
../source/cfg_ping.vhd
../source/ping.vhd
../source/pcim_top.vhd
../source/busrecord.vhd
../source/dumb_arbiter.vhd
../source/dumb_targ32.vhd
../source/dumb_targ64.vhd
../source/stimulus.vhd
../source/ping_tb.vhd
```

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 interface:

```
../../src/xpci/pci_lc_i.vhd
../../src/xpci/pcim_lc.vhd
```

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:

   <Install Path>/vhdl/example/func_sim

4. Create the SimPrim and UniSim libraries. This step only needs to be done once, the first time you perform a simulation:

   ```
   vlib simprim
   vcom -93 -work simprim <Xilinx Install
   Path>/vhdl/src/simprims/simprim_Vpackage_mti.vhd
   vcom -93 -work simprim <Xilinx Install
   Path>/vhdl/src/simprims/simprim_Vcomponents_mti.vhd
   vcom -93 -work simprim <Xilinx Install
   Path>/vhdl/src/simprims/simprim_VITAL_mti.vhd
   vlib unisim
   vcom -93 -work unisim <Xilinx Install
   Path>/vhdl/src/unisims/unisim_VPKG.vhd
   vcom -93 -work unisim <Xilinx Install
   Path>/vhdl/src/unisims/unisim_VCOMP.vhd
   vcom -93 -work unisim <Xilinx Install
   Path>/vhdl/src/unisims/unisim_VITAL.vhd
   ```

5. To run the simulation, enter the following:

   do modelsim.do

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

Chapter 5

Synthesizing a Design

This chapter describes how to synthesize the *ping64* example design with global clocks using the supported synthesis tools. For the 32-bit interface, substitute *ping32* for *ping64*. If you are using a design with reference clocks, substitute *pcim_top* with *pcim_top_r* and *ping_tb* with *ping_tb_r*.

Supported synthesis tools include

- Synplicity Synplify
- Exemplar LeonardoSpectrum
- Xilinx XST

Synplicity Synplify

Before attempting to synthesize a design, ensure that the Synplicity Synplify environment is properly configured.

Verilog

1. Start Synplify and choose File > New (Figure 5-1), or click the new file icon on the toolbar. The Create a new document dialog appears.

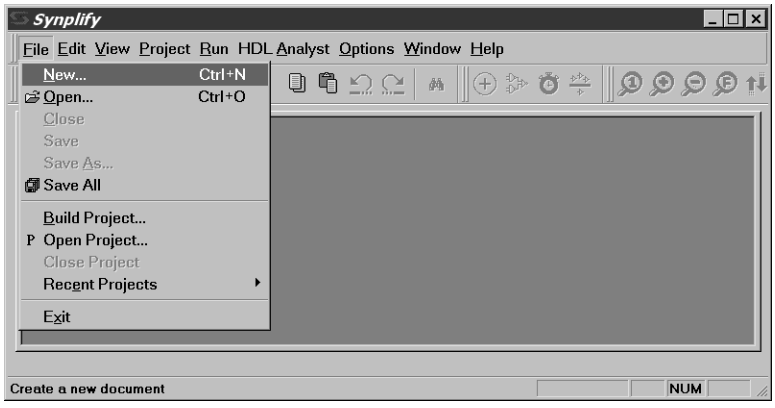

Figure 5-1: **Create a New Document**

2. Select Project File under File Type, and enter the project name (*flowtest* in this example) and synthesis directory:

   `<Install Path>/verilog/example/synthesis`

3. Click OK to exit the dialog and return to the project window.

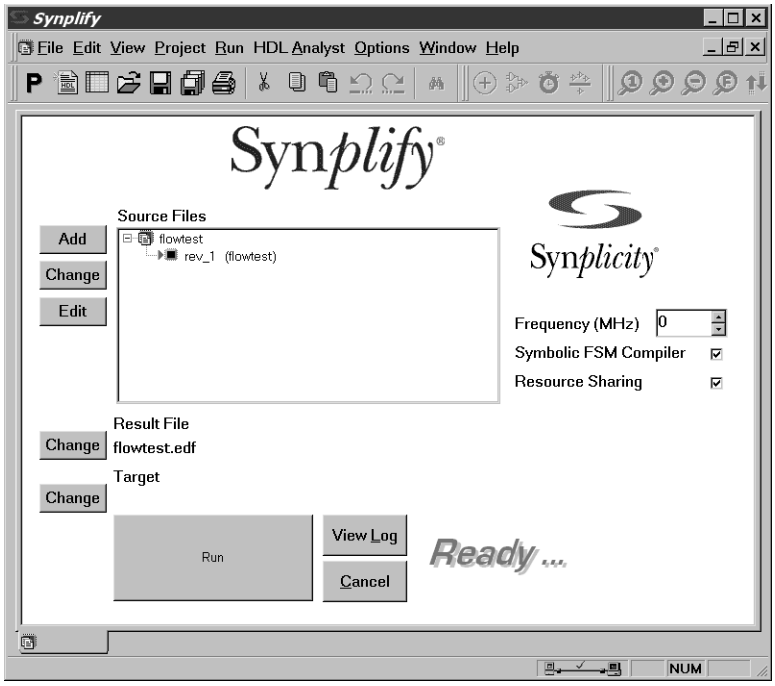

Figure 5-2: **Main Project Window**

4. Click Add to add source files to the new project (Figure 5-2). The first file (used by any design that instantiates Xilinx primitives) is located in:

   `<Synplicity Install Path>/lib/xilinx`

5. Navigate to the `virtex.v` file (Figure 5-3); then click Add to move this source file into the Files To Add list.

Figure 5-3: **Select Files to Add (Library)**

The next files are located in:

```
<Install Path>/verilog/src/xpci
```

6. Navigate to the xpci directory (Figure 5-4), select the simulation model and the wrapper files (pci_lc_i.v and pcim_lc.v), and click Add to move these files into the Files To Add list. (Ctrl + click to select multiple files.)

Figure 5-4: **Files to Add (LogiCORE IP Files)**

The final set of design files (the user application) is located in:

```
<Install Path>/verilog/example/source
```

7. Navigate to the source directory, select the cfg_ping.v, pcim_top.v, and ping.v files (Figure 5-5), and then click Add.

Figure 5-5: **Select Files to Add (User Application)**

8. Click OK after adding the three final files (for a total of six source files) to return to the main project window.
9. In the Source Files list, view the list of newly added source files by double-clicking the `flowtest/verilog` folder (if it is not already open).
10. Reorder the source files in the folder by dragging them to list them in the hierarchical order displayed in Figure 5-6.

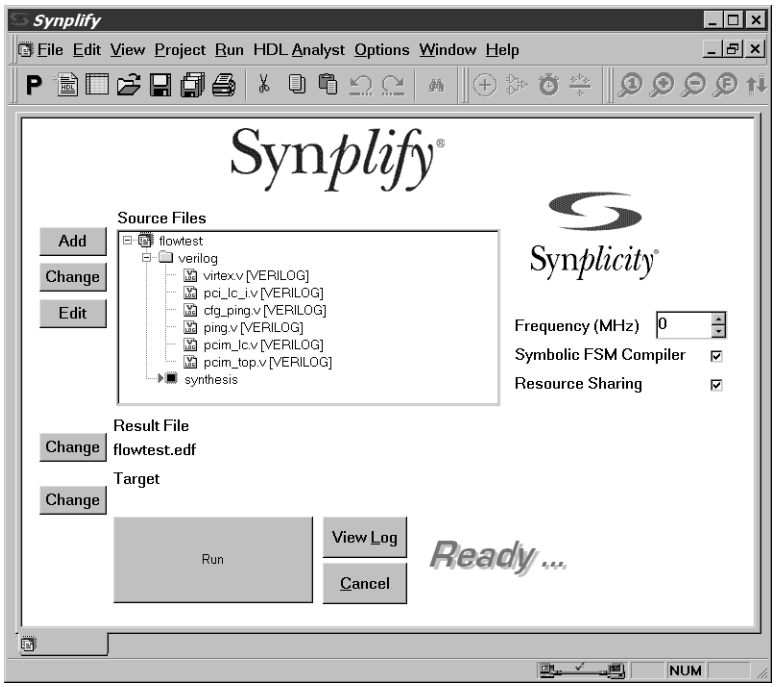

Figure 5-6: **Main Project Window Showing Source Files**

11. Click Change Result File to display the EDIF Result File dialog box; then move it to following directory:

    `<Install Path>/verilog/example/synthesis`

12. Name the file `pcim_top.edf` and click OK to set the name of the result file and return to the main project window.

    Note: In practice, the directory for the EDIF result file does not need to be changed. However, the sample processing scripts included with the example design assume that the output EDIF files will be located in the `synthesis` directory.

13. Click Change Target to display the Options for Implementation dialog box, as shown in Figure 5-7.

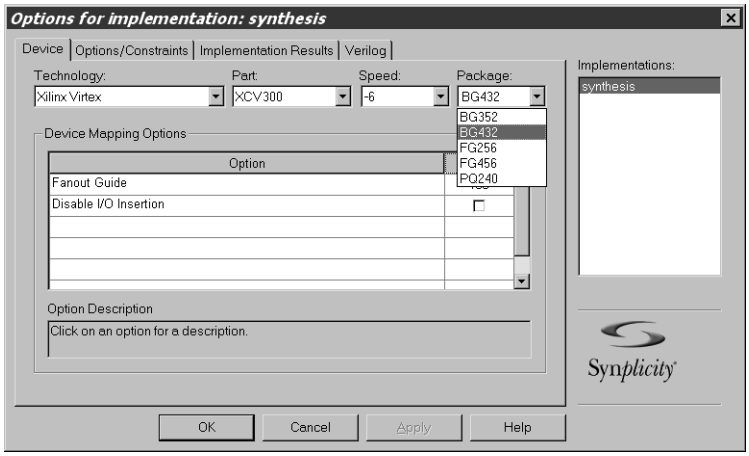

Figure 5-7: **Options for Implementation: Device**

14. Set the Technology, Part, Speed, and Package options on the Device tab to reflect the targeted device (a V300BG432-6 in this example).

    Be sure that Disable I/O Insertion is deselected.

15. On the Options/Constraints tab, deselect Symbolic FSM Compiler (but leave Resource Sharing selected) and set the Frequency to 66 MHz.
16. Deselect Write Vendor Constraint File on the Implementation Results tab.
17. Click OK to return to the main project window.
18. Click Run.

    Synplify synthesizes the design and writes out an optimized EDIF file. In the lower-right corner of the window, the various stages or synthesis, such as Compiling or Mapping, are displayed.

    When the process is complete, *Done* is displayed. Note that Synplify may issue a number of warnings (are expected) about instantiated I/O cells and attributes used for other synthesis tools.

VHDL

1. Start Synplify and choose File > New (Figure 5-8), or use the new file icon on the toolbar. The Create a New Document dialog appears.

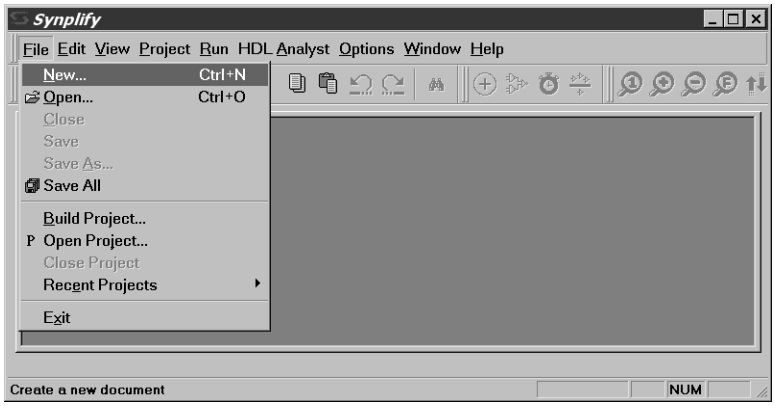

Figure 5-8: **Create a New Document**

2. Under File Type, select Project File and enter the project name (*flowtest* in this example) and synthesis directory:

   `<Install Path>/vhdl/example/synthesis`

3. Click OK to exit the dialog and return to the project window (Figure 5-9).

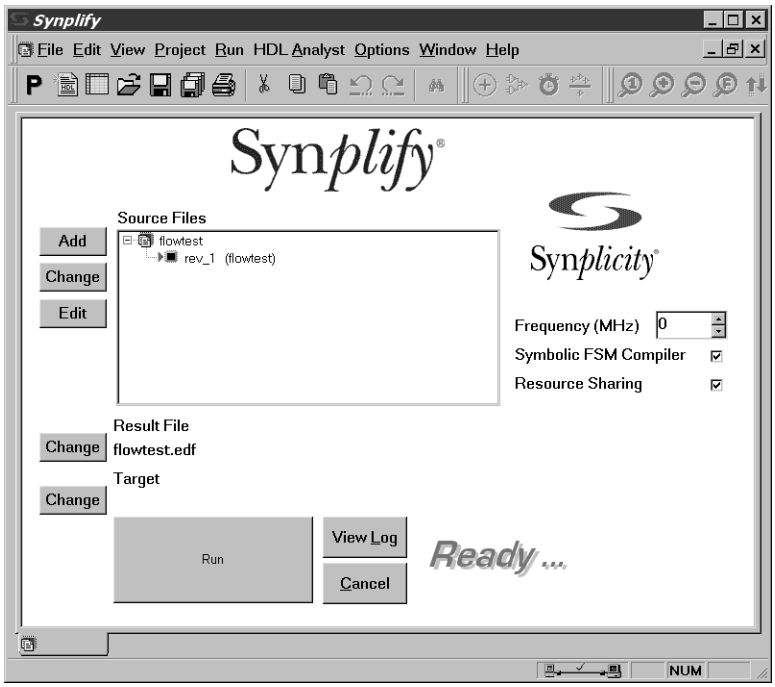

Figure 5-9: **Project Window**

4. To add source files to the new project, click Add.

   The first file (used by any design that instantiates Xilinx primitives) is located in:

   `<Synplicity Install Path>/lib/xilinx`

5. Select the `virtex.vhd` file (Figure 5-10); then click Add to move this source file into the Files To Add list.

Figure 5-10: **Select Files to Add (Library)**

The next files are located in:

`<Install Path>/vhdl/src/xpci`

6. Navigate to the `xpci` directory, select the simulation model and the wrapper files (pci_lc_i.vhd and pcim_lc.vhd) (Figure 5-11)
7. Click Add to move these files into the Files To Add list. (Ctrl + click to select multiple files.)

Figure 5-11: **Select Files to Add (LogiCORE IP Files)**

The final set of design files (the user application) is located in:

```
<Install Path>/vhdl/example/source
```

8. Navigate to the source directory, select the cfg_ping.vhd, pcim_top.vhd, and ping.vhd files and click Add.
9. After adding the three final files (for a total of six source files), click OK to return to the main project window.
10. In the Source Files list, view the list of newly added source files by double-clicking the `flowtest/vhdl` folder (if it is not already open). Click and drag to reorder the source files in the hierarchical order shown in Figure 5-12.

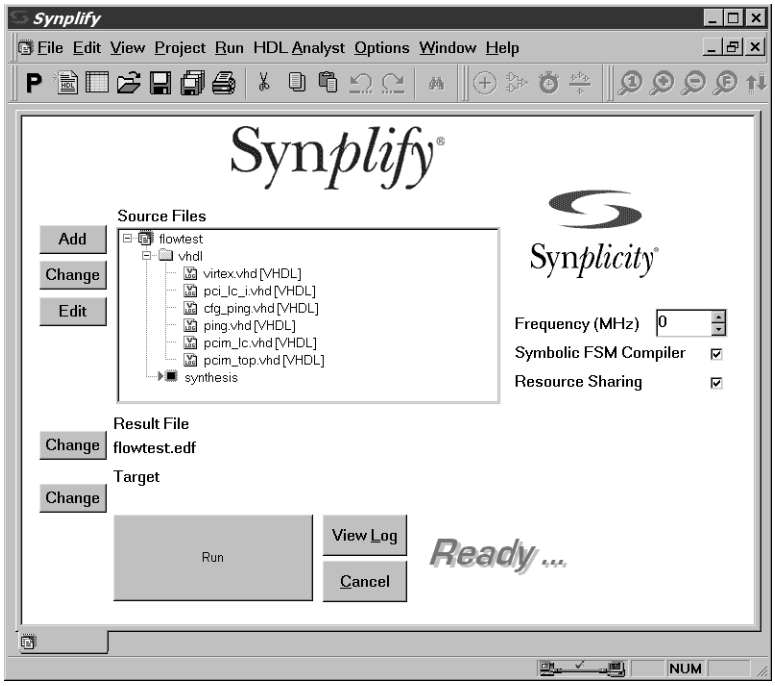

Figure 5-12: **Project Window with Source Files**

11. Click Change Result File to display the EDIF Result File dialog box; then move the to following directory:

    ```
    <Install Path>/vhdl/example/synthesis
    ```

12. Rename the file `pcim_top.edf` and click OK to return to the main project window.

    Note: In practice, the directory for the EDIF result file does not need to be changed. However, the sample processing scripts included with the example design assume that the output EDIF files will be located in the `synthesis` directory.

13. Click Change Target to display the Options for Implementation dialog, as shown in Figure 5-13.

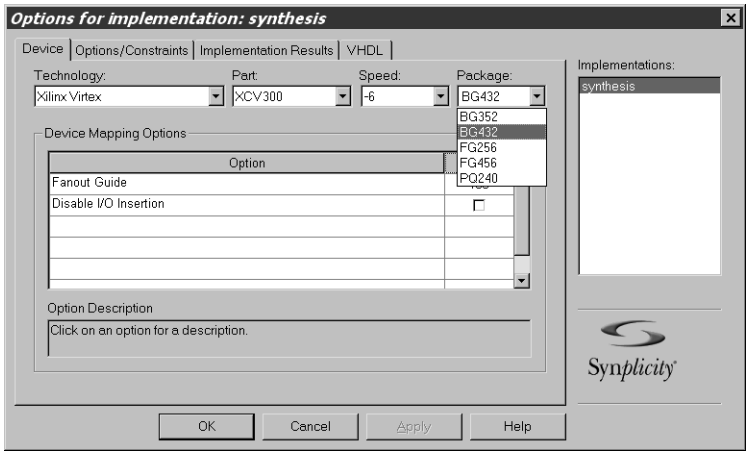

Figure 5-13: **Options for Implementation**

14. On the Device tab, set the Technology, Part, Speed, and Package options to reflect the targeted device (a V300BG432-6 in this example).

    Be sure that Disable I/O Insertion is deselected.

15. On the Options/Constraints tab, deselect Symbolic FSM Compiler (leave Resource Sharing selected) and set the Frequency to 66 MHz.
16. On the Implementation Results tab, deselect Write Vendor Constraint File.
17. Click OK to return to the main project window; then click Run.

Synplify synthesizes the design and writes out an optimized EDIF file. In the lower-right corner of the window, the various stages or synthesis, such as Compiling or Mapping, are displayed. When the process is complete, *Done* is displayed. Note that Synplify may issue a number of warnings (which are expected) about instantiated I/O cells and attributes used for other synthesis tools.

Exemplar LeonardoSpectrum

Before attempting to synthesize a design, ensure that the Exemplar LeonardoSpectrum environment is properly configured for use.

1. Navigate to the synthesis directory:

   ```
   cd <Install Path>/hdl/example/synthesis
   ```

The synthesis directory contains a script for use with LeonardoSpectrum.

2. Edit the script to change the following line:

   ```
   cd <Install Path>/hdl/example/synthesis
   ```

Modify the path to point to the actual installation location, and then save the file.

3. Invoke LeonardoSpectrum.
4. Synthesize the design by running the script `leonardo.tcl`.

   Note: If you run LeonardoSpectrum with the graphical user interface, the quick setup form cannot be used to synthesize the design. Instead, choose File > Run Script from the menu.

The end result of the synthesis step is an EDIF file that is fed into the Xilinx implementation tools during the implementation step.

In practice, the provided script file must be modified to accommodate other designs. To provide insight into the synthesis script, the major steps are presented below:

1. Various synthesis options are set through the use of environment variables.

   These must be present in the script, and should not be modified. The synthesis library is also loaded; this may be altered for different devices and speed grades.

2. The design is loaded by reading in the design files.

   At this point, the top-level module is declared as the `present_design`. The script adds `nopad` attributes (with a value of `FALSE`) to all PCI bus interface signals. The I/O structures for these ports are directly instantiated in the wrapper file.

3. The optimization step is done with the `-hierarchy preserve` and the `-chip` options.

   The `-hierarchy preserve` option prevents LeonardoSpectrum from dissolving the design hierarchy. The `-chip` option indicates that automatic I/O buffer insertion should be performed.

4. After synthesis is complete, the synthesized netlist is written.
5. The tool may issue warnings about unused signals; these warnings are expected.

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 synthesis directory:

   ```
   cd <Install Path>/hdl/example/synthesis
   ```

   The synthesis directory contains a script for use with Xilinx XST; this script is called `run_xst.bat` for PC platforms and `run_xst.sh` for UNIX platforms. The `run_xst.cmd` and `run_xst.prj` files are common and used by both scripts.

2. If required, modify the files as required to suit your application. You may need to change the target architecture and select different wrapper and configuration files.
3. Synthesize the design by running the script.

The end result of the synthesis step is an NGC file that is fed into the Xilinx implementation tools during the implementation step. The tool may issue warnings about unused signals; these warnings are expected.

Chapter 6

Implementing a Design

This chapter describes how to implement the *ping64* example design with global clocks using the supported FPGA implementation tools (included with the ISE Foundation Development System). For the 32-bit Initiator/Target core for PCI, substitute *ping32* for *ping64*. If you are using a design with reference clocks, substitute *pcim_top* with *pcim_top_r* and *ping_tb* with *ping_tb_r*.

ISE Foundation

Before implementing a design, ensure that the Xilinx environment is properly configured and the design has been successfully synthesized.

1. Navigate to the implementation directory:

   ```
   cd <Install Path>/hdl/example/xilinx
   ```

   This directory contains a script for use with Xilinx ISE to place and route the example design; this script is called `run_xilinx.bat` for PC platforms and `run_xilinx.sh` for UNIX platforms.

   Note: If targeting PCI 66 MHz to XC3S1200E, this script runs special post-processing of the routed NCD file to meet hold-time requirements.

2. Review the appropriate `run_xilinx` script file, noting the following:
   - Several required special environment variables are set at the beginning of the script; do not remove them.
   - The `ngdbuild` command lists both `../../src/xpci` and `../synthesis` as search directories. The `xpci` directory contains a netlist of the PCI interface, and the `synthesis` directory must contain the EDIF netlist generated during design synthesis.

     The `ngdbuild` command also reads a UCF that corresponds to a desired target device and a particular version of the core interface. The UCFs provided with the core interface include constraints that guarantee pinout and timing specifications. These constraints must always be used during processing.

     Additional constraints that relate to the user application must be placed in this file. Before making additions to the user constraints file, back up the original so that it can be restored if necessary.
   - The `map` command requires no special arguments, but uses an input/output register packing option.
   - The `par` command also requires no special arguments. PAR effort levels and other implementation options may be adjusted if necessary.

     Note: When targeting PCI 66 MHz for XC3S1200E, the `run_xilinx` script runs the following commands after `par`:

```
xdl -ncd2xdl pcim_top_unloaded.ncd

cd s3e_66

xilperl 3s1200efg400.pl -xin ..\pcim_top_unloaded.xdl -xout
  ..\pcim_top_routed.xdl -pads 3s1200efg400.info

cd ..

xdl -xdl2ncd pcim_top_routed.xdl

fpga_edline pcim_top_routed.ncd -p s3e_66/3s1200efg400.scr
```

This allows XC3S1200E to meet hold-time requirements in 66 MHz PCI. The script achieves this by loading the data-bus clock enable with unbonded I/O flip-flops, to properly realign its setup-hold window with the global clock.

- The `trce` command performs a static timing analysis based on the design constraints originally specified in the user constraints file.
- The `netgen` command generates a simulation model of the placed and routed design.

3. Implement the design by running the appropriate script.

During initial processing trials, it is useful to enter commands one at a time from the command line rather than running the script so that you can inspect the output of each step.

XILINX®

Chapter 7

Timing Simulation

This chapter describes how to perform timing simulation using the *ping64* example design with global clocks using the supported timing simulation tools. For the PCI 32 interface, substitute *ping32* for *ping64*. If you are using a design with reference clocks, substitute *pcim_top* with *pcim_top_r* and *ping_tb* with *ping_tb_r*.

Supported timing simulation tools include

- Cadence IUS v8.1 -s009 and above
- Mentor Graphics ModelSim v6.4b and above

Cadence NC-Verilog

Before attempting timing simulation, ensure that the NC-Verilog environment is properly configured for use. In addition, you must have successfully completed the implementation phase using the Xilinx tools.

1. Navigate to the timing simulation directory and copy the back-annotated timing models from the implementation directory:

   ```
   cd <Install Path>/verilog/example/post_sim
   cp ../xilinx/pcim_top_routed.v .
   cp ../xilinx/pcim_top_routed.sdf .
   ```

2. Edit the `ping_tb.f` file. This file lists command line arguments for NC-Verilog, and is shown below:

   ```
   ../source/ping_tb.v
   ../source/stimulus.v
   ../source/busrecord.v
   ../source/dumb_arbiter.v
   ../source/dumb_targ32.v
   ../source/dumb_targ64.v
   ./pcim_top_routed.v
   +libext+.vmd+.v
   -y <Xilinx Install Path>/verilog/src/simprims
   ```

3. Modify the library search path by changing `<Xilinx Install Path>` to match the Xilinx installation directory and save the file.
4. To run the NC-Verilog simulation:

   ```
   ncverilog -f ping_tb.f
   ```

NC-Verilog processes the simulation files and exits. The test bench prints status messages to the console. After the simulation completes, view the `ncverilog.log` file to check for errors.

The Simvision browser may be used to view the simulation results. Simvision is started with the following command:

```
simvision
```

Model Technology ModelSim

Before attempting timing simulation, ensure that the ModelSim environment is properly configured for use. In addition, you must have successfully completed the implementation phase using the Xilinx tools.

Verilog

1. Move into the timing simulation directory and copy the back-annotated timing models from the implementation directory:

   ```
   cd <Install Path>/verilog/example/post_sim
   cp ../xilinx/pcim_top_routed.v .
   cp ../xilinx/pcim_top_routed.sdf .
   ```

2. Edit the `ping_tb.f` file. This file lists command line arguments, and is shown below:

   ```
   ../source/ping_tb.v
   ../source/stimulus.v
   ../source/busrecord.v
   ../source/dumb_arbiter.v
   ../source/dumb_targ32.v
   ../source/dumb_targ64.v
   ./pcim_top_routed.v
   +libext+.vmd+.v
   -y <Xilinx Install Path>/verilog/src/simprims
   ```

3. Modify the library search path by changing `<Xilinx Install Path>` to match the Xilinx installation directory. Save the file.
4. Invoke ModelSim, and make sure that the current directory is set to:

   ```
   <Install Path>/verilog/example/post_sim
   ```

5. Type the following to run the simulation:

   ```
   do modelsim.do
   ```

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

VHDL

1. Navigate to the timing simulation directory and copy the back-annotated timing models from the implementation directory:

   ```
   cd <Install Path>/vhdl/example/post_sim
      cp ../xilinx/pcim_top_routed.vhd .
      cp ../xilinx/pcim_top_routed.sdf .
   ```

2. View the `ping.files` file. This file lists the individual source files required, and is shown below:

   ```
   ./pcim_top_routed.vhd
   ../source/busrecord.vhd
   ../source/dumb_arbiter.vhd
   ../source/dumb_targ32.vhd
   ../source/dumb_targ64.vhd
   ../source/stimulus.vhd
   ../source/ping_tb.vhd
   ```

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

   <Install Path>/vhdl/example/post_sim

4. Create the SimPrim library. This step is required only once, the first time you perform a simulation:

   ```
   vlib simprim
   vcom -93 -work simprim <Xilinx Install
   Path>/vhdl/src/simprims/simprim_Vpackage_mti.vhd
   vcom -93 -work simprim <Xilinx Install
   Path>/vhdl/src/simprims/simprim_Vcomponents_mti.vhd
   vcom -93 -work simprim <Xilinx Install
   Path>/vhdl/src/simprims/simprim_VITAL_mti.vhd
   ```

5. Type the following to run the simulation:

   ```
   do modelsim.do
   ```

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