CorePCI Target, Master, and Master/Target

Version 5.3 User's Guide

Actel

For more information about Actel's products, call 888-99-ACTEL
or visit our Web site at http://www.actel.com

Actel Corporation • 955 East Arques Avenue • Sunnyvale, CA USA 94086
U.S. Toll Free Line: 888-99-ACTEL • Customer Service: 408-739-1010 • Customer Service FAX: 408-522-8044
Customer Applications Center: 800-262-1060 • Customer Applications FAX: 408-739-1540

Actel Europe Ltd. • Daneshill House, Lutyens Close • Basingstoke, Hampshire RG24 8AG • United Kingdom
Tel: +44 (0)125-630-5600 • Fax: +44 (0)125-635-5420

Actel Japan • EXOS Ebisu Bldg. 4F • 1-24-14 Ebisu Shibuya-ku • Toyko 150 • Japan
Tel: +81 (0)334-457-671 Fax: +81 (0)334-457-668

5029114-5

CorePCI Target, Master, and Master/Target

Version 5.3 User's Guide

Actel

Actel Corporation, Sunnyvale, CA 94086

Printed in the United States of America

Part Number: 5029114-5

Release: August 2002

Trademarks

Table of Contents

Introduction

The *CorePCI Target, Master, and Master/Target User's Guide* contains information for the Actel CorePCI Target-Only, Master/Target and Master-Only macros. This includes macros and associated testbench descriptions, a description of the design flow, and associated information about customizing the macro. Also included are procedures for simulating, synthesizing, placing-and-routing, and performing static-timing analysis of the macros.

Use this guide in conjunction with the *CorePCI Target, Master, and Master/Target* DataSheet. The DataSheet provides low-level details and functions of the macros not covered in this guide and information about the capabilities and the implementation of the macros.

Document Organization

The *CorePCI Target, Master, and Master/Target User's Guide* contains the following chapters:

Chapter 1 - Macro and TestBench Descriptions describes the CorePCI macros, the HDL organization of the macros, and the testbench files included with the them.

Chapter 2 - Design Flow describes the design flow for implementing the CorePCI macros.

Chapter 3 - Customizing contains information about modifying the CorePCI macros to meet specific design criteria.

Chapter 4 - Behavioral Simulation describes the procedure for performing a behavioral simulation of the CorePCI.

Chapter 5 - Synthesis describes the procedure for synthesizing the CorePCI macros.

Chapter 6 - Design Layout describes the procedure for implementing the CorePCI macros using Designer.

Chapter 7 - Static Timing Analysis describes the procedure for performing static-timing analysis of the CorePCI macros using the Designer Embedded Timing tool. This chapter also includes a brief section regarding where to find information about timing simulation.

Appendix A - CorePCI 5.3 Testbench provides an overview of Actel's CorePCI testbench and a guide to modifying existing and building new procedures. It describes the hierarchy of the testbench and provides the syntax for a variety of procedures and functions.

Appendix B - Product Support provides information about contacting Actel for customer and technical support.

Document Assumptions

This document assumes the following:

1. You have installed and are familiar with Actel's Designer or Libero software.
2. You have installed your HDL-synthesis and simulation software.
3. You are familiar with the VHDL or Verilog hardware description language.
4. You are familiar with UNIX workstations and operating systems or PCs and Windows operating systems.

Naming Conventions

This document uses naming conventions to distinguish between the various PCI functions, sizes, and targeted architectures as shown in Table 1:

Table 1.PCI Naming Conventions

Variable	Value	Description
<act_fam>	pa or a500k	ProASIC A500K device family
	apa	ProASIC^{PLUS} APA device family
	sx or 54sx	A54SX device family
	sxa or 54sxa	A54SX-A device family
	sxs or rtsxs	RTSX-S device family
	ax or axcelerator	Axcelerator device family
<PCI-Macro>	targ	Target-Only macro
	tdma	Target+DMA macro
	tmst	Master/Target macro
	32	32-bit macro
	mast	Master-Only macro
	64	64-bit macro
	sdram	PCI macro includes an SDRAM interface

For example, 'targ32sdram' means the macro can support functions of a 32-bit Target-only PCI with an SDRAM interface. Other variables, such as <device> and <package> refer to the target device and package of a specific FPGA family.

The "$ALSDIR" variable used in this guide represents the Actel installation directory. PC users should substitute the full path name of the Actel installation directory for "$ALSDIR" when this variable appears throughout this guide.

CorePCI Documentation

The CorePCI macros include a printed and online version of the *CorePCI Target, Master, and Master/Target User's Guide,* which contains information and procedures for using the CorePCI Target-Only, Master/Target and Master-Only macros. The guide is in PDF format on the CD-ROM in the "\doc" directory. To view the online manual, you must have Adobe® Acrobat Reader® installed. Actel provides Reader on the Designer CD-ROM.

Actel Manuals

Actel Designer software includes printed and online manuals. The online manuals are in PDF format on the CD-ROM in the "/manuals" directory. These manuals are also installed onto your system when you install the Designer software. To view the online manuals, you must install Adobe® Acrobat Reader® from the CD-ROM.

Designer includes the following manuals, which provide additional information on designing Actel FPGAs:

Getting Started User's Guide. This manual contains information for using the Designer Series Development System software to create designs for, and program, Actel devices.

Designer User's Guide. This manual provides an introduction to the Designer series software as well as an explanation of its tools and features.

PinEdit User's Guide. This guide provides a detailed description of the PinEdit tool in Designer. It includes cross-platform explanations of all the PinEdit features.

ChipEdit User's Guide. This guide provides a detailed description of the ChipEdit tool in Designer. It includes a detailed explanation of the ChipEdit functionality.

Timer User's Guide. This guide provides a detailed description of the Timer tool in Designer. It includes a detailed explanation of the Timer functionality.

SmartPower User's Guide. This guide provides a detailed description of using the SmartPower tool to perform power analysis.

A Guide to ACTgen Macros. This Guide provides descriptions of macros that can be generated using the Actel ACTgen Macro Builder software.

Actel HDL Coding Style Guide. This guide provides preferred coding styles for the Actel architecture and information about optimizing your HDL code for Actel devices.

Silicon Expert User's Guide. This guide contains information to assist designers in the use of Actel's Silicon Expert tool.

Synopsys® Synthesis Methodology Guide. This guide contains preferred HDL coding styles and information to assist designers in the design of Actel devices using Synopsys CAE software and the Designer Series software.

VHDL Vital Simulation Guide. This guide contains information to assist designers in simulating Actel designs using a Vital compliant VHDL simulator.

Verilog Simulation Guide. This guide contains information to assist designers in simulating Actel designs using a Verilog simulator.

Silicon Sculptor User's Guide. This guide contains information about how to program Actel devices using the Silicon Sculptor software and device programmer.

Flash Pro User's Guide. This guide contains information about how to program Actel ProASIC and ProASIC PLUS devices using the Flash Pro software and device programmer.

Silicon Explorer II. This guide contains information about connecting the Silicon Explorer diagnostic tool and using it to perform system verification.

Macro Library Guide. This guide provides descriptions of Actel library elements for Actel device families. Symbols, truth tables, and module count are included for all macros.

ProASIC PLUS Macro Library Guide. This guide provides descriptions of Actel library elements for Actel ProASIC and ProASIC PLUS device families. Symbols, truth tables, and tile usage are included for all macros.

1

Macro and TestBench Descriptions

This chapter contains a description of the CorePCI macros and associated testbenches.

Macro Description

This guide provides information regarding nomenclature, file organization, file description, and the use of the CorePCI product in various design flows. There are also instructions on simulation, synthesis, layout, and timing verification. The document includes some technical information about macros, but the primary source for macro information is the CorePCI datasheet.

Product Organization

The CorePCI product consists of three basic components:

1. The CorePCI VHDL macro
2. The CorePCI Verilog macro
3. The testbench used to verify functionality

The VHDL (Verilog) macro is located in the "\vhdl\src (\verilog\src)" and the "\vhdl\wrapper (\verilog\wrapper)" subdirectories.

The "src" files define the master and target functionality of the macro.

The "wrapper" files define the top- or chip-level of the macro and instantiate the master function, the target function, and the back-end function.

The testbench files are in the "\tbench\vhdl" subdirectory. Figure 1-1 illustrates the complete file organization for the CorePCI macro.

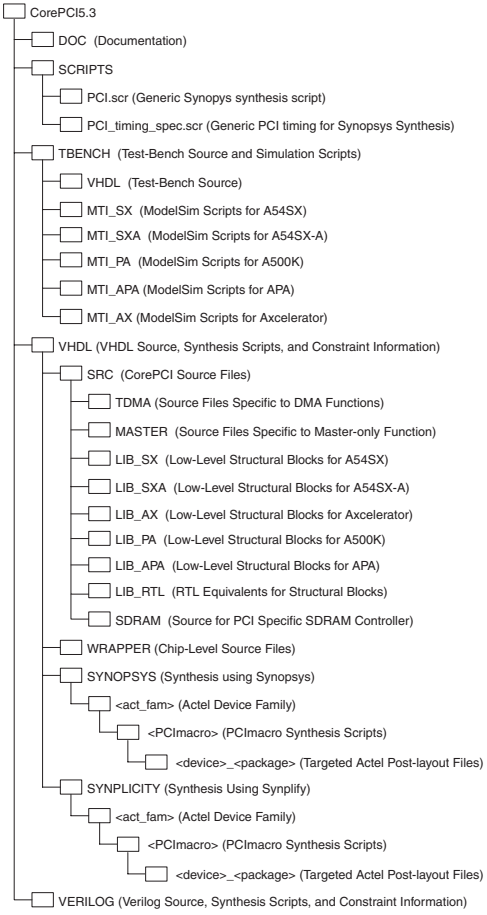

Figure 1-1. File Organization for the CorePCI Macro

In Figure 1-1, <act_fam> is one of the 54SX (SX), 54SX-A (SXA), RTSX-S (SXS), Axcelerator (AX), A500K (PA), or APA (APA) device familiy.

PCI Functions

There are 14 wrapper files that define the various CorePCI functions. The naming conventions are as follows:

The following four-character prefixes define the function types of the "wrapper" files: "targ," "tdma," "tmst," and "mast" (defined in Table 1-1).

The PCI target function is defined in the "target64" module and the PCI master function is defined in the "dma" module.

All wrappers instantiate the "target64" module and all but the "targ" instantiate the "dma" module.

The 32/64 value in the "wrapper" name defines the width of the PCI and back-end data bus.

Finally, the "sdram" wrappers are examples of complete CorePCI applications and include an SDRAM ("sdramctrl") and SSRAM controller. These wrappers are used in the testbench to verify the function of the macro.

The generic wrapper (i.e., no SDRAM) is the bare macro with no application-specific back end. You can use either type of wrapper as a starting point for your application-specific design.

Table 1-1. PCI "Wrapper" Functions

Prefix	Function Name	Description
targ	Target-Only	Strictly acts as target (i.e., slave) on the PCI bus.
tdma	Target+DMA	Can behave as either a target or initiator. The registers that control the initiator (i.e., master) functions are only accessible from the PCI bus via either I/O or configuration commands.
tmst	Target+Master	Can behave as either a target or initiator. The registers that control the initiator (i.e., master) functions are only accessible from the back-end bus.

Table 1-1. PCI "Wrapper" Functions

Prefix	Function Name	Description
mast	Master-Only	Strictly acts as a master (i.e., initiator) on the PCI bus. The registers that control the master functions are only accessible from the back-end bus.

For example:

- The "targ64sdram_wrp" file is a 64-bit Target-Only macro that includes an SDRAM and SSRAM controller. This file is used in the testbench for functional verification.
- The "tmst32_wrp" is a 32-bit Target+Master macro with the PCI and generic back-end signals pinned out.

The complete list of chip-level wrapper files is as follows:

targ32_wrp	targ32sdram_wrp
tdma32_wrp	tdma32sdram_wrp
tmst32_wrp	tmst32sdram_wrp
targ64_wrp	mast64sdram_wrp
tdma64_wrp	targ64sdram_wrp
tmst64_wrp	tdma64sdram_wrp
mast32sdram_wrp	tmst64sdram_wrp

The source files for the "target64" component are in the "\vhdl\src" and "\verilog\src" subdirectories. The source files for the "dma" component are in the "\vhdl\src\tdma (\verilog\src\tdma)" subdirectory. Low-level, family-specific components are defined in the "\lib_sx, \lib_sxa, \lib_ax, \lib_pa," and "\lib_rtl" subdirectories. Note that the RTSX-S implementation uses the same subdirectories as the SX.

Synthesis Scripts

A variety of sythesis scripts are also included to simplify using CorePCI. They use the naming conventions in Table 1-1.

Testbench

The CorePCI testbench tests each of the functions using the "sdram" wrapper files. The source for the testbench is in the "\tbench\vhdl" subdirectory. The models for the SDRAM and SSRAM used by the testbench are in the "\tbench\micron" subdirectory. A detailed description of the testbench is in Appendix A.

HDL Organization

The functional HDL models of the macros are written in generic VHDL and Verilog and allow easy customization of the macro. Refer to the *CorePCI Target, Master, and Master/Target User's Guide* Datasheet for additional information. Figure 1-2 on page 6 illustrates the organization of the CorePCI model.

A top-level wrapper is used to merge the "target64," "dma" (used for Target+DMA and Master-Only), and the back-end control logic (for example, an SDRAM controller). As illustrated, the "target64" is composed of 6 functional blocks plus the "TARGPACK," which is used for setting the customization constants.

"Dataphase" and "add_phase64" provide state-machine control for the CorePCI Target.

"Datapath" consists of the logic that steers data and address between the PCI bus and the back end.

"Config" contains the configuration registers and "parity64" is responsible for the generation and checking of parity.

"Addr_cntr64" loads and increments the address counter and configuration pointer.

"Burst64" controls the flow of data between the PCI bus and the back end.

For Target-Only functions, the "dma" module is not required. For Master-Only, the "config" block in "target64" is not needed. For Target+DMA and Master/Target macros, all modules are required.

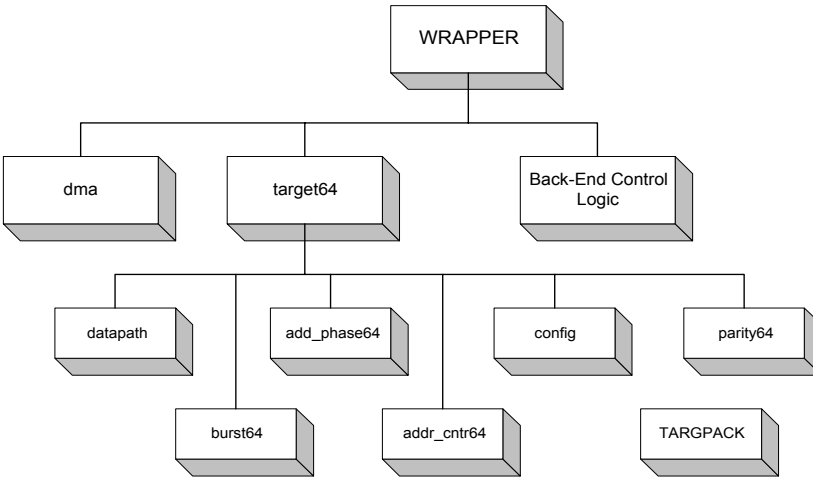

Figure 1-2. CorePCI HDL Organization

VHDL Testbench Description

The PCI testbench consists of a master controller, a PCI monitor, an arbiter, and devices under test. The master model is used for generating configuration cycles and for performing basic read-and-write tests of the macro when operating as a PCI target. The PCI monitor checks for and flags abnormal PCI bus activity. The arbiter determines PCI bus ownership. You can use the testbench for behavioral or timing simulation and modify the testbench to test any custom functions added to the macros.

Figure 1-3 illustrates the organization of the testbench.

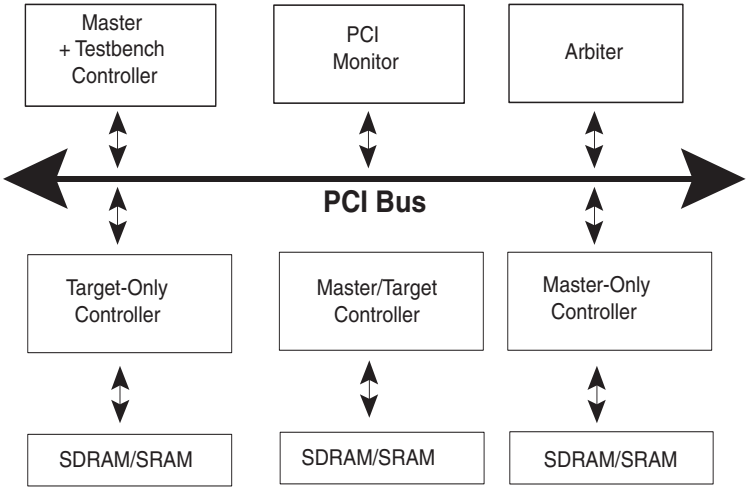

Figure 1-3. Testbench Organization

When you first invoke the testbench, the master (mast_cfg64) does a configuration cycle to PCI slots 1-6 to determine which slots are occupied, which type of macro is used(Target-Only, Master/Target, or Master-Only), and which base-address registers are enabled. The testbench then uses this information to perform a variety of tests described in the following section.

The testbench master is a dynamic 32-bit or 64-bit master. The master always attempts to run 64-bit transfers for memory-burst transactions; however, it automatically defaults to a 32-bit transfer if the addressed device does not respond with an ACK64n. All I/O and configuration cycles are 32-bit.

There are two basic configurations for the testbench included with the CorePCI product:

The first configuration, the 32-bit test (system32), puts a 32-bit Target macro in slots 2 and 3, a 32-bit Target+DMA macro in slot 4, a Target+Master in slot 5, and a 32-bit master in slot 6.

The second configuration, the 64-bit test (system64), puts a 64-bit Target macro in slots 2 and 3, a 64-bit Target+DMA macro in slots 4, a 64-bit Target+Master in slot 5, and a 64-bit master in slot 6.

These configurations quickly test a variety of macro combinations. The user is free to add or delete macros from the slots and the testbench should still run correctly.

Macro and testbench File Information

The PCI macros and testbenches consist of a variety of files. This section describes the files and the directory structure for the macros and testbenches. The testbenches are designed to work with the directory structure that Actel has created. Actel recommends that you maintain the directory and file structure described in this section.

"\vhdl\src" and "\verilog\src" Directories

These directories contain the VHDL and Verilog source files and directories with source files for the Target, Master and Target/Master macros. The "target64," "add_phase64," "datapath," "config," "addr_cntr64," and "parity64" signals are generic to both the Target and Master functions. The "targpacks" are custom for the various CorePCI functions and are in the "\tdma", "\master", and "\target" subdirectories:

target/target64.vhd target64.v	Top level of the Target macro that interconnects all the lower-level blocks.
master/ master_target.vhd master_target.v	This master_target file is equivalent to the target64 except that the config module is eliminated. This module is only used in Master-Only functions.
target/targpack.vhd targpack.v	Customization package that contains user-configurable values to set the address widths, PCI-configuration space values, etc. File defining configuration space-ID-constant values.
add_cntr64.vhd add_cntr64.v	Loads and increments the address counter and configuration pointer.
add_phase64.vhd add_phase64.v	Address phase state machine. This logic monitors the PCI bus and determines if the device is being addressed.
datapath.vhd datapath.v	Main data path for the PCI macro.

burst64.vhd burst64.v	State machine that controls the flow of data between the back end and the PCI bus.
config.vhd config.v	PCI-configuration register logic.
tdma/dma.vhd dma.v	DMA master state machine that controls all DMA interactions and direction.
tdma/dma_reg.vhd dma_reg.v	DMA register information including the PCI Start Address, the RAM (or back-end) Start Address, and the DMA Control register.
parity64.vhd parity64.v	Parity generation and check logic.

"\lib_pa; *\lib_apa; *\lib_sx; *\lib_sxa; *\lib_ax" Directories*

Several low-level cells are available for each device family in the \vhdl or \verilog directories. For example, the APA Verilog directory is "\verilog\lib_apa". These blocks are necessary because synthesis tools are unable to meet PCI performance requirements using generic VHDL or Verilog code. These blocks are shown in the following table

cbe_par.vhd cbe_par.v	Parity generation logic.
cm8d.vhd cm8d.v cm8dp.vhd cm8dp.vcm cm8dx.vhd cm8dx.v dfe1b	Special multiplexed flip-flop. This function is used extensively in the burst state machine to help control setup for IRDYn and FRAMEn.
datapath_registers.vhd datapath_registers.v	The data-path registers for the CorePCI macro. This logic provides the data path between the PCI bus and the back-end logic.

framen_buf	Buffer for the framen input to help control setup on framen
mustclock targclock tdmaclock	Clock buffers for the targeted family
mux4_8.vhd mux4_8.v	An 8-bit, 4:1 mux.

Note: Please refer to the release directory for the actual file list.

"\vhdl\src\sdram" or "\verilog\src\sdram" Directory

This directory contains the SDRAM back-end controller.

"\vhdl or verilog\synopsys" Directory

This directory contains Synopsys-related compilation files for the CorePCI macro.

"\vhdl or verilog\synplicity" Directory

This directory contains Synplicity-related compilation files for the CorePCI macro.

vhdl_macros32.do vhdl_macros64.do	Scripts to compile the VHDL 32-bit or 64-bit CorePCI macros. This includes the Target-Only, Master-Only, Target+DMA, and Target/Master.
vhdl_setup.do	Script to set up the library environment for VHDL simulation
verilog_macros32.do verilog_macros64.do	Scripts to compile the VHDL 32-bit or 64-bit CorePCI macros. This includes the Target-Only, Master-Only, Target+DMA, and Master/Target.
verilog_setup.do	Script to set up the Verilog library environment and compile the Verilog libraries.
comptb32.do	Script to compile the 32-bit PCI test.

comptb64.do	Script to compile the 64-bit PCI test.
run.do	Script to run the testbench.
wave.do	Top-level testbench ports for waveform display.

"\tbench\vhdl" Directory

This directory contains the HDL files for the testbench.

system32.vhd	Top-level testbench containing the 32-bit PCI macro instances.
system64.vhd	Top-level testbench containing the 64-bit PCI macro instances.
mast_cfg64.vhd	64-bit Master PCI model to initiate the tests, etc.
pci64_mon.vhd	PCI bus monitor that detects basic protocol errors and displays the PCI cycles.
arbiter.vhd	Basic PCI bus arbiter allowing the multiple masters to share the PCI bus.
misc.vhd	Package of support procedures.
pci_pack.vhd	Package of PCI formats and conversion functions.
mcfgpack.vhd	Package of procedures to perform read-and-write cycles.
components.vhd	Package of top component definitions.
be_control.vhd	Special testbench block used to control back-end signals like the interrupt, ready, and error signals.
io_block.vhd	32-bit SRAM used in the testbench to exercise the base address register 1 logic.
startup_pack.vhd	Test routines to determine the existence and type of macros present.

sdram_mu.vhd	Micron SDRAM model (located in the "\tbench\micron\sdram" directory).
components32_wrp.vhd components64_wrp.vhd	Component packages for the top-level CorePCI macros used in the testbench.
tests_t1.vhd	Package of tests, mainly target based.
tests_t2.vhd	Package of tests, mainly target based.
tests_t3.vhd	Package of tests, complex DMA and Master tests.

Tests Carried Out

The testbench tests basic PCI configuration reads, configuration writes, memory reads, memory writes, and DMA transfers. The top-level module of the testbench is called "system." Two "target" or "targdma" macros are instantiated in the testbench with instance names SLOT2, and SLOT3 for "target" or SLOT4, and SLOT5 for "targdma." Each "target" or "targdma" has an SDRAM mapped to base-address register 0 and an SRAM module mapped to base-address register 1 on the back end. The following is a list of functions the current version of the testbench simulates. Detailed information on tests is provided in Appendix A.

Target-Only and Target+DMA Tests

1. The PCI configuration space is read and checked so that it matches the expected values at reset.
2. The PCI macro is configured and reread to verify the configuration write cycles.
3. The PCI configuration Command and Status register DWORD is written and read using byte transfers to verify the byte-enable logic.
4. A simple burst-transfer write followed by read-transfer cycles are carried out using zero-wait state-transfer modes.
5. A PCI cycle with an address-phase parity error is generated. The testbench verifies that the target ignores the request and registers the parity error in its status register.

6. Various read-and-write memory cycles are carried out with different burst lengths and different master transfer rates (IRDYn inactive). All the data writes are verified by rereading the test data that has pattern changes with every cycle.
7. PCI I/O read-and-write cycles are carried out. The operation of the I/O decode logic is tested.
8. The system-interrupt logic for the macro is tested. This test checks the external interrupt-control bits in the DMA control register and checks to verify that the EXT_INT pin will cause the PCI INTAn signal to become active.
9. Read-and-write memory transfers are carried out with data-parity errors. The PCI command and status configuration bits relating to parity are tested.
10. Multiple read-and-write transfers are carried out, with assorted burst lengths with both master IRDYn and BE_RDY deassertions to verify that no data corruption occurs.
11. Target abort test. The back end asserts the ERROR signal, which should initiate a Target abort cycle on the PCI bus.
12. Target retry and disconnect without data test. For the retry test, the BE_REQ signal is asserted prior to a transaction and gains control of the back end. When a PCI cycle is run to the Target, a PCI retry cycle should be performed. For the disconnect with data, a PCI burst transaction is initiated. At some point, the RD_BE_RDY\WR_BE_RDY signals are deasserted. The controller then times out (after 8 cycles) and performs a PCI disconnect cycle.

Target+DMA Tests Only

1. A simple DMA master transfer between the two targets is carried out. Both DMA read-and-write transfers are carried out.
2. DMA master transfers are started and at the same time the testbench polls the DMA status registers within the macro. During this test, the macro is simultaneously carrying out DMA cycles and having its status registers read.
3. Multiple DMA master transfers are carried out with various lengths to verify that the correct numbers of DWORDS are transferred during a DMA cycle. This test is carried out in zero-wait state-transfer mode.

4. DMA master transfer cycles are carried out with short bus-grant periods. This causes the DMA master to stop the transfer and restart it. The target macro in the transfer will also issue a target disconnect causing the DMA transfer to stop. This test checks that the macro correctly restarts the DMA transfer under these conditions. This test is carried out in zero-wait state-transfer mode.
5. Two macros are set up to carry out simultaneous DMA transfers between each other. During this test, each macro carries out DMA transfers while the other macro performs target accesses to it. At the end of the test, the data is checked to verify that no data corruption has occurred. This test is carried out in zero-wait state-transfer mode.

Master-Only Tests

1. Configuration read-and-write cycles are tested.
2. Test I/O read-and-write cycles are tested.
3. Memory transfers of various lengths, including single-DWORD transfers and bursts, are tested.
4. Target Retry and Disconnects are tested.
5. Target Abort are tested.

2 Design Flow

This chapter describes the design flow for creating an Actel design using the CorePCI macros. This includes information about supported design flows and a design flow overview.

Supported Design Flows

Actel provides guidelines for using theCorePCI macros with the following:

- Model Technology V-System simulator 5.5e or later
- Synopsys FPGA Compiler 99.04 or later
- Synplicity Synplify 7.1a or later synthesis tool
- Designer Series Development System R1-2002 or later

The macros are VITAL 95 VHDL- and industry standard Verilog OVI-compliant and should operate with any simulator that supports these standards. However, Actel has only tested the macros with the previously mentioned tools. Although other synthesis tools should work, Actel recommends using one of the previously mentioned synthesis tools because performance results have not been verified using other synthesis tools and may not meet the PCI requirements.

Design Flow Overview

The design flow for generating a PCI-compliant Target-Only or Master/Target Controller using synthesis and simulation tools and Designer has six main steps:

1. Customizing
2. Behavioral Simulation
3. Synthesis
4. Design Layout
5. Static-Timing Analysis
6. Timing Simulation

These steps are described in the following sections.

Customizing

The CorePCI macros have a variety of constants that help customize the macro to meet a variety of needs. These constants are used to set memory and I/O space sizes, as well as PCI-device ID information. The first step in the design process is to understand these constants and to customize them to meet the specific needs of an application. The constants are in the chip-level wrapper files and the "targpack.vhd" or the "targpack.v" files.

Note: Actel highly recommends using the unmodified macro throughout the various steps in the design flow prior to attempting any customizing.

Behavioral Simulation

After you have customized the macro, you can simulate it to verify functionality before you perform synthesis. If you find any problems, you can quickly address them at the behavioral level and simulate the design again. Typically, you use unit delays and include the test bench with the macro used to drive the simulation.

If you use a test bench, you can automate the verification process. A test bench is a behavioral design with built-in functions to provide stimulus to the DUT (device under test) inputs and monitor the outputs. You can set up the test bench to report any functional or timing errors that occur during simulation. Refer to "Behavioral Simulation" on page 27 and the documentation included with your simulation tool for information about performing behavioral simulation.

Synthesis

After customizing and performing a behavioral simulation of the macro, you must synthesize it before placing-and-routing it in Designer. After synthesizing the macro, you must generate an EDIF netlist for use in Designer. Refer to "Synthesis" on page 31 and the documentation included with your synthesis tool for information about generating an EDIF netlist.

Design Layout

Use Designer to lay out all supported Actel devices for CorePCI designs. Make sure to use GENERIC for the Edif flavor and VHDL or Verilog for the Naming Style when importing the EDIF netlist into Designer. Refer to "Design Layout" on page 35 and the *Designer User's Guide* for information about using Designer.

Static-Timing Analysis

Use the timer tool in Designer to perform static timing analysis on your design. Refer to "Static Timing Analysis" on page 43 for information about using Timer to perform static-timing analysis.

Timing Simulation

Perform a timing simulation of your customized macro after placing-and-routing it. Timing simulation is optional if you have performed static-timing analysis. Timing simulation requires information extracted from Designer, which overrides unit delays in the Actel libraries.

Create Your Own CorePCI Application

To create your own custom CorePCI application, perform the following steps:

1. **Copy an existing wrapper file ("\vhdl\wrapper") to a new file name.** You can begin with either an "sdram" wrapper or a generic wrapper.
   - For a Target application, begin with a "targ" wrapper
   - For a Target+DMA, begin with a "tdma" wrapper
   - For a Target+Master application, begin with a "tmst" wrapper
   - For a Master application, begin with a "mast" wrapper
2. **Set customization constants.** Define the constants in the chip-level wrapper (generics in VHDL and parameters in Verilog) and the "targpack" file to meet the needs of your application.
3. **Interface your application to the back-end interface described in the datasheet. You can define the code in the wrapper or in a separate component that is instantiated in the wrapper.**
4. **Modify the port list in the wrapper to reflect the new back-end application. PCI ports should not be altered.**

Test Your CorePCI Application

Test the new application after you create it. You can test the new application with the existing test bench or in a test bench you have created. Testing with the existing test bench is covered in detail in Appendix A, "CorePCI 5.3 Testbench". The following steps summarize the procedure.

1. **Modify the top-level "system32.vhd" or "system64.vhd" file to include your new chip-level wrapper and any external devices the wrapper will connect to.** You can add the wrapper into a new slot (e.g., Slot #1) or modify an existing slot.
2. **Create a component declaration for your application wrapper in a package file.** You can find existing declarations in the "\tbench\vhdl" subdirectory under the "components32_wrp.vhd" or "components64_wrp.vhd" files.
3. **Modify the "vhdl_macros32," "vhdl_macros64," "verilog_macros32," or "verilog_macros64" to include your new wrapper and any additional files required for compilation.**
4. **Create the libraries ("vhdl_setup").**
5. **Compile the macros ("vhdl_macros32").**
6. **Compile the test bench ("comptb32").**
7. **Run the simulator (run file).**

The default test bench should be able to locate the new slot and will attempt to run tests on the slot that are appropriate for the function. Tests are called from the "mast_cfg64.vhd" file. You can delete existing tests and create and add new tests. For more information on this step, refer to Appendix A, "CorePCI 5.3 Testbench".

Synthesize Your New CorePCI Application

Copy a synthesis script of the same function type to a new name. Modify the script to point to the correct targpack, new wrapper, and any additional files required by the application. You do not need to compile the component package created for simulation.

Layout Your New Application and Verify Timing

To layout your new application and verify timing, perform the following steps:

1. **Open Designer**
2. **Import the netlist and set the device, package, and speed grade.**
3. **Import the PCI pin definitions or constraints, if required.**

4. **66MHz designs will require constraints on CLK-OUT and the Input Setup for TRDYn, IRDYn, and FRAMEn.** You will then need to run timing-driven layout. 33MHz does not require constraints and you can run it in standard layout mode.
5. **Verify Reg-Reg and CLK-OUT timing using Timer.**
6. **Verify input setup with Timer.**

3

Customizing

This chapter describes in detail the customizing options for the CorePCI macros. This includes information about configuration registers, memory address space, I/O space, data-transfer options, user-interface options, and back-end interface considerations.

The macros are written in generic VHDL and Verilog, allowing for flexibility and quick modification of the macros. Several customization constants have been defined to control the overall behavior of the macro. These constants are defined in the "targpack" files and in the chip-level wrapper files. The constants are defined in terms of parameters in Verilog.

General Configuration

This section describes the constants (described in the previous sections) and their functions. The following two constants are defined in the chip-level wrapper files.

MHz_66

When this constant is set to a "1", the macro indicates that it is 66 MHz capable, otherwise only 33 MHz is supported. The only impact of this constant is to set bit 5 in the configuration status register.

BIT_64

When combined with the appropriate top-level wrapper, this constant creates either a 32-bit or 64-bit PCI controller. For VHDL, when this constant is set to a "1," the macro runs in 64-bit data mode. Otherwise it runs in 32-bit data mode. This is accomplished by a variety of VHDL-generate statements in various parts of the code.

For Verilog, the same function is accomplished by either defining or not defining the "BIT_64" constant. Correct selection of functions in the code is then controlled by ifdef - else - endif statements.

Configuration Registers

There are eight configuration-register constants that you can modify in the macros. These constants are defined in the "targpack" file or the chip-level wrapper file.

USER_DEVICE_ID

Setting the "USER_DEVICE_ID" constant to the desired value modifies the "DEVICE-ID" (02H) configuration register.

USER_VENDOR_ID

Setting the "USER_VENDOR_ID" constant to the desired value modifies the "VENDOR-ID" (00H) configuration register.

USER_REVISION_ID

Setting the "USER_REVISION_ID" constant to the desired value modifies the "REVISION-ID" (08H) configuration register.

USER_BASE_CLASS

Setting the "USER_BASE_CLASS" constant to the desired value modifies the "Class_Code" (08H) configuration register.

USER_SUB_CLASS

Setting the "USER_SUB_CLASS" constant to the desired value modifies the "Class_Code" (08H) configuration register.

USER_PROGRAM_IF

Setting the "USER_PROGRAM_IF" constant to the desired value modifies the "Class_Code" (08H) configuration register.

USER_SUBSYSTEM_ID

Setting the "USER_SUBSYSTEM_ID" constant to the desired value modifies the "Subsystem_Id" (2CH) configuration register. You must set this constant to comply with PCI Local Bus Specification 2.2 PCI Special Interest Group, *PCI Local Bus Specification v2.2* (Hillsboro, OR. PCISIG, 1999). For information on how to obtain this specification, go to the PCI SIG web site (http:\\www.pcisig.org).

USER_SUBVENDOR_ID

Setting the "USER_SUBVENDOR_ID" constant to the desired value modifies the "Subsystem_Vendor_Id" (2CH) configuration register. You must set this constant to comply with PCI Local Bus Specification 2.2 (PCISIG).

HOT_SWAP_ENABLE

Setting this bit to a '1'b will implement the Hot Swap extended capability (at address "80"h) and cause the Capability Pointer in configuration space to have a value of '80'h.

Memory and I/O Address Space

This section describes constants for memory and I/O address space. The following constants are defined in the "targpack" file or the top-level wrapper file.

MADDR_WIDTH

The amount of memory-address space in base-address register 0 supported by the macros is determined by the "MADDR_WIDTH" constant. This constant determines the width of the address generated. The number of bits of Memory Base Registers (defined by the configuration register at address 10h) that can be written to is also controlled by "MADDR_WIDTH."

BAR1_ENABLE

If a second base-address register is required, then this constant should be set to a "1" and will be at address 14h in configuration space.

BAR1_IO_MEMORY

If "BAR1_ENABLE" is set to a "1," then this constant defines the base address to be an I/O (set to a "0") or a memory ("1").

BAR1_ADDR_WIDTH

The integer that defines the address space for the base-address register at 14h.

BAR1_PREFETCH

If the base address register is a memory, this bit defines whether or not the memory is prefetchable.

Target+DMA Settings

This section describes the Target+DMA settings. The settings can be modified in the chip-level wrapper file.

DMA_IN_IO

There are three DMA registers that are associated with Target+DMA macros. The user has the option of locating these registers in the configuration space at 40h, 44h, and 48h or in I/O space. For the I/O space option, the "DMA_IN_IO" must be set to a "1." For this case, a new base-address register is defined in configuration space 18h and is defined to be I/O. It has a default address space of 256 bytes and the registers are located at 40h, 44h, and 48h within the space.

DMA_CNT_EN

In this revision of the CorePCI macro, you can, as an option, make the PCI Start Address, RAM Start Address, and Transfer Count registers to be counters. This feature allows the Master to terminate a cycle when it loses GNTn and allows for incremental transfers. You can disable this feature for PCI macros that perform single-DWORD or short bursts in order to save resources for other functions.

Back-End Interface Data Flow Customizing

There are a variety of signals that control the flow of data between the PCI controller and the back end. The "RDY" inputs inform the PCI controller that the back end is prepared to transfer data. The "NOW" signals indicate that data is being transferred during the cycle that the "NOW" signal is asserted. This is true for all cases except during the initial start-up when the "pipe_full_cnt" is nonzero. The back-end flow control is defined at the wrapper level of the PCI macro.

rd_be_rdy

This input to the PCI macro is from the back-end memory controller and indicates the back-end memory controllers' readiness to service the PCI controllers request for data.

rd_be_now

This output of the PCI macro informs the back-end memory controller that data on the MEM_DATA bus will be read on the next rising clock edge.

wr_be_rdy

This signal is driven from the back end. It indicates that the back end is ready to receive data.

wr_be_now

This signal is driven by the PCI macro to the back-end memory controller. It tells the back-end memory controller that data on the MEM_DATA bus is valid.

pipe_full_cnt(2:0)

In some cases, the back-end controller has a start-up latency and needs the address incremented prior to data being available. This is true on any type of read to a synchronous device (such as a synchronous SRAM). The "pipe_full_cnt" indicates the number of increments that should be performed prior to data being available. The PCI controller drives the "NOW" signals during this period. However, "NOW" is read until after the number of cycles defined by the "pipe_full_cnt" signal. The "pipe_full_cnt" should be defined following the assertion of DP_START and should remain valid throughout the cycle.

4 Behavioral Simulation

This chapter describes in detail the procedures for performing a behavioral simulation of the CorePCI macros controller using the MTI V-System simulator. Also included is information about compiling an Actel VITAL VHDL library for use in simulation. Refer to the documentation included with your simulation tool for additional information about performing behavioral simulation.

Compiling a VITAL VHDL Library for ModelSim

To simulate the CorePCI macros targeted for the A54SX, A54SX-A, RTSX-S, and Axcelerator device families, you must first compile the Actel VITAL VHDL library. If you are targeting ProASIC devices, library primitive compilation is not required and you may skip this section. The following steps describe the compilation procedure:

1. **Create a directory called "mti" in the "$ALSDIR\lib\vtl\95" directory.**
2. **Invoke the V-System simulator.**
3. **Change to the "$ALSDIR\lib\vtl\95\mti" directory.**
4. **Create an Actel family library directory. T**ype the following command at the prompt:

   ```
   vlib <act_fam>
   ```

   For example:

   ```
   vlib a54sx
   ```

   Possible choices for <act_fam> are a54sx, a54sxa, axcelerator (ax), a500k and apa. Note that behavioral simulation for the RTSX-S utilizes the A54SX-A library. Refer to Table 1 on page ix for more information.
5. **Compile the Actel VITAL VHDL library.** Type the following command at the prompt:

   ```
   vcom -work <act_fam> ..\<act_fam>.vhd
   ```

   For example, to compile the 54SX library, type the following command:

   ```
   vcom -work a54sx ..\54sx.vhd
   ```

Simulating the CorePCI Macros Using ModelSim

Use the following procedures to simulate the VHDL and Verilog versions of the CorePCI macros.

VHDL

Once you have compiled the VHDL library, compile the testbench and simulate the PCI macro. The following steps describe the procedure:

Note: If you have to customize the macro, make sure you have modified the customization constants before simulating the macro. Refer to "Customizing" on page 21 for information about customizing the macros.

1. **Invoke the V-System simulator.**
2. **Change to one of the MTI directories under \tbench.** These directories contain the MTI script files for simulation. They are mti_sx, mti_sxa, mti_apa, mti_pa, and mti_ax.
3. **Create the "work" libraries.** This step creates the required library directories. Type the following command at the prompt:

   ```
   do vhdl_setup.do
   ```

4. **Map the Actel VITAL VHDL library to the family library directory.** Type the following command at the prompt:

   ```
   vmap <act_fam> $ALSDIR\lib\vtl\95\mti\<act_fam>
   ```

   For example, to map the 54SX library, type the following command:

   ```
   vmap a54sx $ALSDIR\lib\vtl\95\mti\a54sx
   ```

5. **Compile the macro and the testbench.** Type one of the following commands at the prompt:

   ```
   do vhdl_macros32.do <32-bit macros>
   ```

   or

   ```
   do vhdl_macros64.do <64-bit macros>
   ```

6. **Compile one of the following testbenches.** Type one of the following commands at the prompt:

   ```
   do comptb32.do <32-bit testbench>
   ```

   or

   ```
   do comptb64.do <64-bit testbench>
   ```

7. **Simulate the testbench.** Type the following command at the prompt:

   ```
   do run.do
   ```

8. **Follow the menu options in the testbench.**

Verilog

The following steps describe the compilation of the testbench and the simulation of the Verilog macro.

Note: If you have to customize the macro, make sure you have modified the customization constants before simulating the macro. Refer to "Customizing" on page 21 for information about customizing the CorePCI macros.

1. **Invoke the V-System simulator.**
2. **Change to one of the MTI directories under \tbench.** These directories contain the MTI script files for simulation. They are mti_sx, mti_sxa, mti_apa, mti_pa, and mti_ax.
3. **Create the ALSDIR variable.** To correctly compile the Verilog A54SX library, you must set the ALSDIR variable to point to the Actel Designer root directory. Type the following command in the V-System simulator:

   ```
   set ALSDIR <designer path>
   ```

4. **Create the "work" libraries.** This step creates the required library directories and compiles the verilog library primitives. Type the following commands at the prompt:

   ```
   do verilog_setup.do
   ```

5. **Add the "\verilog\src" directory.** From the Compile menu, select the Verilog Compile Options and include the "\verilog\src" and the

"\verilog\src\lib_<act_fam>" directories before accepting. This includes the directory for the "targpack" file for compilation. When possible ensure that the "targpack" file used in simulation is the same one that will be used during synthesis. This prevents discrepancies in function between simulation and hardware.

6. **Compile the macro.** Type the following commands at the prompt:

   ```
   do verilog_macros32.do <32-bit macros>
   ```

   or

   ```
   do verilog_macros64.do <64-bit macros>
   ```

7. **Compile one of the following testbenches.** Type the following command at the prompt:

   ```
   do comptb32.do <32-bit testbench>
   ```

   or

   ```
   do comptb64.do <64-bit testbench>
   ```

8. **Simulate the testbench.** Type the following command at the prompt.

   ```
   do run.do
   ```

9. **Follow the menu options in the testbench.**

5 Synthesis

This chapter describes the procedures for synthesizing the CorePCI macros using the Synopsys Design Compiler, Exemplar Leonardo Spectrum, or Synplicity Synplify synthesis tool. Also included are guidelines to follow when synthesizing the macros. Refer to the documentation included with your synthesis tool for additional information about synthesizing a design.

General Synthesis Guidelines

The macros have been designed so that most synthesis tools can meet the PCI design requirements. The challenge with synthesis is meeting the PCI setup requirement of 3 ns, clock to out of 6 ns, and clock period of 15 ns for 66 MHz PCI designs. To meet this requirement, you must use synthesis constraints.

Actel recommends placing the following constraints on the design during synthesis. The instance names will vary with different tools.

- Clock period of 15 ns for 66 MHz and 30 ns for 33 MHz
- Constrain the input setup times for all PCI inputs to be 3 ns for 66 MHz and 7 ns for 33 MHz
- Constrain the output valid times for all PCI outputs to be 6 ns for 66 MHz and 11 ns for 33 MHz. This is most important because these are the most critical paths in the design.
- In VHDL synthesis, the "targpack" files are called out explicitly
- In Verilog synthesis, the "targpack" files are included and each synthesis tool uses different search strategies to locate the included files

You need to ensure that your synthesis tool is using the correct "targpack." To prevent differences between implementation and model simulation, ensure that the "targpack" used by synthesis and simulation are the same file.

Synthesizing Using Synopsys

This section describes the use of Design Compiler from Synopsys to synthesize the CorePCI macro. A set of global variables are used to configure the CorePCI macro for simulation and synthesis.

Synopsys Synthesis Directory Structures

Synopsys scripts are provided with the PCI macros to enhance synthesis results. These files can be found in the "vhdl\synopsys" or "verilog\synopsys" directories. The Synopsys synthesis is organized using the directory structures shown in Figure 5-1.

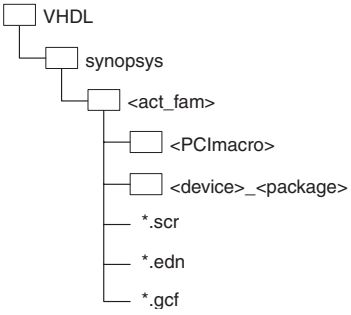

Figure 5-1. Synopsys Synthesis Directory Structure

Please refer to "PCI Naming Conventions" on page ix for possible choices for <act_fam> and <PCImacro>. Refer to the *Actel FPGA Package and Selector Guide* for choices for <device> and <package>. The synthesis scripts and synthesized netlist (*.edn in EDIF format) are located in the <PCImacro> directory. The main Synopsys script to invoke is called <PCImacro>.scr, and the files under the <device>_<package> directory contain the post-layout data.

To compile, perform the following steps:

1. **Modify the parameters in the script file to compile the appropriate design.** There are several parameters that you must set to compile the macro correctly. These include pci_width, pci_func, and pci_frequency. Options for these variables are given next to their declarations in the script file.
2. **Verify that both "bit_64" and "maddr_width" customization constants are set correctly.**
3. **Invoke Design Compiler.**
4. **Execute the script file.**

Notes on Using Synopsys

The scripts are set up in a general fashion to include pad and timing information for all macros. Consequently, depending on which macro you configure, you may see that some of the pad and timing information is irrelevant and will be reported as an error. These errors should not affect the netlist and may be manually removed.

Note: For Verilog compilations, Synopsys uses the "TARGPACK.v" in the "\verilog\synopsys" directory, so ensure that its settings reflect the desired functional intent of the macro.

Synthesizing Using Synplicity

Project files have been defined to assist in the compilation of the CorePCI macro into the different Actel FPGAs. We recommend you compile the designs using Synplicity version 7.1a or later.

Synplicity Synthesis File Structure

Sample Synplicity project files (with macros) are provided in the "\vhdl\synplicity" or "\verilog\synplicity" directories.

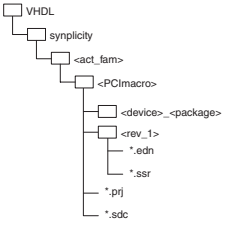

Figure 5-2. Synplicity Synthesis Directory Structure

Please refer to "PCI Naming Conventions" on page ix for possible choices for <family> and <PCImacro>. Refer to the *Actel FPGA Package and Selector Guide* for choices for <device> and <package>. The Synplicity synthesis project file (<act_fam>.prj), Synplicity design constraint file (<act_fam>.sdc) are located in the <act_fam> directory. The synthesized netlist is stored in the directorey <rev_1>; post-layout data is stored in the <device>_<package> directory.

To compile, perform the following steps:

1. **Invoke Synplify.**
2. **Settings Verification.** The script file was created to automatically set up the synthesis environment and all of the design constraints for the uncustomized macro. However, verification of the technology, device, speed-grade, and clock-frequency settings for your particular design is suggested. Only change the Results File if the output directory and file name are different than the default.
3. **Modify the project file.** Move the desired chip-level function to the last position of the wrapper list (files with "wrp" extension). Refer to naming conventions in this chapter for proper selection.
4. **Open the <PCImacro>.prj file in Synplify.**
5. **Compile and map the design.** Click the Run button. Synplicity compiles and maps the design and writes an EDIF netlist and an SDF constraint file with the same name as the project file to the default directory or any directory of choice.
6. **Review the synthesis results.** Synplify provides a suite of tools to analyze the results of the synthesis run. A quick summary of design utilization, warning messages, maximum clock frequency, and a list of longest paths can be viewed in the log file by clicking the view log button. HDL Analyst allows the designer to look at the RTL and schematic views of the results and do some pre-layout timing analysis.

6 Design Layout

This chapter describes how to use Actel Designer software to perform design layout on the synthesized CorePCI macros. This includes information about compiling the design netlist, assigning pins, and design layout. Refer to the *Designer User's Guide* for additional information about using Designer. Sample designer scripts and data files are provided with the PCI macros to enhance your place-and-route results. You can find these files in the following directory:

```
\vhdl[or verilog]\synopsys[or synplicity]\<family>\<PCImacro>\<device>_<package>
```

Figure 6-1 shows the default VHDL directory structure.

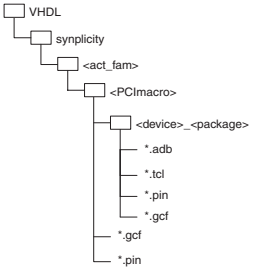

Figure 6-1. Design Layout Directory Structure

Please refer to "PCI Naming Conventions" on page ix for possible choices for <act_fam> and <PCImacro>. Refer to the *Actel FPGA Package and Selector Guide* for choices for <device> and <package>. The files relevant files are as follows:

- **ADB files** - (default name is <PCImacro>_wrp.adb) contain all the design layout information, including netlists, constraints, and post-layout designs
- **Tcl file** - (default name is <PCImacro>_wtp.tcl) is a Tcl script you may use to reproduce the ADB file
- **GCF file** - Layout constraint file for ProASIC and ProASIC^{PLUS} family devices
- **PIN file** - Pin assignment map for the targeted device and package for non-ProASIC families.

Compiling a PCImacro Using Designer

Before performing any task on the design in Designer, you must first import a netlist and compile the design into an ADB file. The following steps describe the procedure:

1. **Invoke Designer.**
   - For PC, choose Designer from the Designer Program Group in the Programs menu under the Start menu.
   - For UNIX, type the following command at the prompt:

     ```
     designer &
     ```

     The Designer Main window is displayed.
2. **Open the Import Netlist dialog box.** From the File menu, choose Import and select the Netlist File command. Click the New button in the Import or Open dialog box. The Import Netlist dialog box is displayed.
3. **Specify netlist options.** Specify EDIF as the Netlist type. Select your netlist by typing the full path name or clicking the Browse button. Select GENERIC as Edif Flavor and VHDL or Verilog as the Naming Style. Click OK.
4. **Set up the design.** In the Design Setup dialog box specify the design name, if not specified, and select the family.
5. **Select the appropriate device, package, and speed grade from the Device Setup Wizard.**
6. **Use the default Device Variations and Operating Conditions.**
7. **Set the PCI-compliancy mode check box if it is an option.** Or, for Axcelerator, select the PCI technology for the IO banks in ChipEdit.
8. **Compile your design.** Click the Compile button in the Designer Main window.
9. **Save the Design.**

Assigning Pin Layout Constraints

For Actel antifuse FPGA devices, you must assign pins manually with the PinEdit tool or import them directly into Designer from the corresponding pin file.

For Actel Flash FPGA devices, you must import layout constraints (as a GCF file) in order to constrain the place-and-route.

Import Constraint File

Pins may be assigned using PinEdit. To do so, perform the following steps:

1. **Import the constraint file.** Select the Import command from the File menu and choose the Auxiliary File command. The Import Auxiliary File dialog box is displayed.
2. **From the File Type menu, select Browse to locate the pin or GCF file.** Pin files are in the \<act_fam>\<PCImacro>\<device>_<package> directory.

   Note: In some cases, an error reading the pin file may occur because the "DEF" name in the pin file does not match the design name in Designer. To correct this problem, simply modify the "DEF" name in the pin file and re-import the file.

Assigning Pins Manually

For information on assigning pins manually, refer to the Designer Online help and the *PinEdit User's Guide.*

After compiling, use PinEdit to assign and fix the pins of your design. The pin files contain optimal placement of the PCI pins and are already placed along one side of the device in a manner that matches the signal order on the PCI PCB connector.

Design Layout

For 33 MHz designs that use the antifuse families (SX, SX-A, or Axcelerator), only standard layout is required. You may use timing-driven layout if you wish to do so. For Flash to achieve 66 MHz performance, Actel recommends timing-driven layout.

Standard Layout

Use the following procedure to perform standard layout:

1. **Click the Layout button in the Designer Main window.** The Layout dialog box is displayed.
2. **Select Standard Mode only and leave all other options OFF.**
3. **Click OK.** Designer performs standard layout on the design.
4. **Extract timing information by clicking the Back-Annotate button.** Specify the post-layout netlist and SDF file name.
5. **Save the design.**

Timing-Driven Layout for Antifuse Families

Use the following procedure to perform timing-driven layout for A54SX, A54SX-A, RT54SX-S and Axcelerator devices. Signal names vary depending on the synthesis tool you use. If you are unable to determine what signals to set, contact Actel's Customer Applications Center.

1. **Invoke Timer.** Click the Timer button in the Designer Main window. This displays the Timer window.
2. **Specify a clock period of 15 ns in the Period field.**
3. **Set Input to Register path constraints on all the PCI inputs.** If some of the PCI signals fail to meet input setup times, set the delay for these paths to 3 ns plus the clock delay for the targeted device. To do so, click the Path tab and from the Edit menu, select Add a set of paths. Then select All PCI Inputs to All Register paths and set your max delay accordingly.
4. **Set Register to PCI Output constraints on all the PCI signals to 6 ns minus the clock delay.** To do so, click the Path tab again, and from the Edit menu choose Add a set of paths. Then select All Registers to All PCI Output paths and set your max delay accordingly.
5. **Save the design and commit to the timing constraints.**
6. **Click the Layout button in the Designer Main window.** This displays the Layout dialog box.
7. **Select Timing Driven only and leave all other options off.**
8. **Extract timing information.** Click the Back-Annotate button. Make sure to specify the post-layout netlist and SDF and filename.

9. Save the design.

You can now examine and verify the postlayout timing information using Timer or perform timing simulation using the postlayout timing information extracted from the design.

Timing-Driven Layout for Flash Families

Use the following procedure to perform timing-driven layout for ProASIC and ProASIC^{PLUS} devices. You must set timing constraints in a GCF file (*.gcf). Sample settings are provided below:

```
// Timing Constraints 30 ns for PCI clock
create_clock -period 30 CLK;

// Assume minimum clock routing delay of 3 ns
// 7 ns - 0.7 ns + 3 ns = 9.3 ns for max input to register delay
set_input_to_register_delay 9.3 -from AD*;
set_input_to_register_delay 9.3 -from CBE*;
set_input_to_register_delay 9.3 -from PAR;
set_input_to_register_delay 9.3 -from FRAMEN;
set_input_to_register_delay 9.3 -from IRDYN;
set_input_to_register_delay 9.3 -from STOPN;
set_input_to_register_delay 9.3 -from IDSEL;
set_input_to_register_delay 9.3 -from DEVSELN;
set_input_to_register_delay 9.3 -from GNTN;
set_input_to_register_delay 9.3 -from PERRN;

// Assume maximum clock routing delay of 3.1 ns
// 11 ns - 3.1 ns -0.7 ns = 7.2 ns for max register to output
// delay
set_register_to_output_delay 7.2 -to AD*;
set_register_to_output_delay 7.2 -to CBE*;
set_register_to_output_delay 7.2 -to PAR;
set_register_to_output_delay 7.2 -to FRAMEN;
set_register_to_output_delay 7.2 -to IRDYN;
set_register_to_output_delay 7.2 -to TRDYN;
set_register_to_output_delay 7.2 -to STOPN;
set_register_to_output_delay 7.2 -to DEVSELN;
set_register_to_output_delay 7.2 -to REQN;
set_register_to_output_delay 7.2 -to PERRN;
set_register_to_output_delay 7.2 -to SERRN;
```

In order to use a GCF file, you must import the constraint file. To do so, follow the instructions in “Import Constraint File” on page 37. After you import the

constraint file successfully, complete the following steps to perform timing-driven layout.

1. **Click the Layout button in the Designer Main window.** This displays the Layout dialog box.
2. **Select Timing Driven only and leave all other options off.**
3. **Extract timing information.** Click the Back-Annotate button. Make sure to specify the post-layout netlist and SDF and filename.
4. **Save the design.**

Meeting Setup and Hold Times Using Axcelerator

To meet the PCI setup and hold timing at both 33 and 66MHz using AX you must use the the Programmable input delay elements (Table 6-1).

Table 6-1. Programmable Input Delay Elements

	Setup	Setup GNTn	Hold
33 Mhz	7 ns	10 ns	0 ns
66 MHz	3 ns	5 ns	0 ns

After layout is complete, examine the setup and hold timing values (Tools, reports, timing). The setup times must be less than the value in Table 6-1 and the hold time less than 0ns (positive values indicate a hold violation). This may indicate that the PCI hold times have been violated; it varies, depending on the actual layout, but you may see violations of up to 1.5 ns, especially on the AD signals.

To correct for this, set the programmable input delay on the IO bank to the worst hold time violation (such as 1.5 ns). Use PinEdit to set the IO bank. Right-click the colored IO bank in the GUI to open the configure IO bank GUI. You must set the IO bank on all the IO banks used for the PCI pins. Once you set the bank delays, enable the input delay on all the PCI pins.

Regenerate your setup and hold report. The 33MHz cores should meet all setup and hold times. For 66MHz cores the some of the setup times (TRDY, IRDY, FRAME and STOP) may violate the 3 ns requirement (by a small amount), disable the input delay on just these pins to correct the setup times. Ensure that the hold times remain within specification at these pins.

Compiling a Design Using Tcl Scripts

Tcl script samples are provided in the <device>_<package> directory for both Verilog and VHDL (see Figure 6-1 on page 35 for more information on the directory structure). To run the Tcl script, type the following command at the DOS prompt:

```
designer script:<PCImacro>_wrp.tcl
```

Please refer to the Scripting section of the Actel Designer User's Guide for instructions on how to compile a design using a Tcl script.

7

Static Timing Analysis

This chapter contains information and procedures for performing static-timing analysis on the design. Included in this chapter is information about verifying setup times, internal delays, and output delays. This chapter also provides a short section regarding where to find more information about the sixth step in design flow process, timing simulation.

Before you perform Static timing analysis, you must set all the timing constraints as described in "Timing-Driven Layout for Antifuse Families" on page 38.

To perform Static timing analysis:

1. **Invoke Timer.** Click the Timer button in Designer.
2. **Check the maximum clock frequency.** Click the Summary tab. The Summary tab reports the maximum clock frequency (calculated from the longest paths between two internal registers). The maximum clock frequency must meet the 33MHz or 66MHz requirements.
3. **Identify the clock buffer delay.** Add another path set from the Clock Input to All Registers (from the Edit menu, select Add Path Set). The timing report shows the clock buffer delay.
4. **Determine the maximum delay from All PCI Inputs to All Registers.** To do so, select the Path tab, and view all PCI Inputs to All Registers (this value was defined when you set the timing-driven place-and-route constraints). Set the max delay to 7 ns (for 33MHz PCI) or 3 ns (for 66MHz PCI), as required.

   Note: Be sure you calculate the actual clock delay accurately; it is the max delay allowed plus the clock buffer delay (i.e. max delay = 7 ns (3 ns) + clock buffer delay).
5. **Determine the maximum delay from all registers to all PCI outputs.** Select the Path tab and view the path values for All Registers to All PCI Outputs (defined when you set the timing-driven place-and-route constraints). Target values for Max delay are 11 ns for 33MHz PCI and 6 ns for 66MHz PCI.

   Note: Be sure you calculate your actual clock delay accurately; for outputs, it is the max delay allowed minus the clock buffer delay (i.e. max delay = 11 ns (6 ns) - clock buffer delay).

A

CorePCI 5.3 Testbench

This appendix provides an overview of Actel's CorePCI testbench and a guide to modifying existing and building new procedures. It describes the hierarchy of the testbench and provides the syntax for a variety of procedures and functions.

Hierarchy of Testbench

The testbench is based on a standard motherboard design with a system master occupying slot 0 of the PCI bus. PCI macros are then "plugged" into one of eight PCI sockets on the motherboard. The macros are distinguished from each other by two mechanisms. For configuration transfers, each slot has a unique IDSEL as an identifier. For all other transfers, address spaces in the base-address registers (BARs) of each slot must be defined to ensure that there are no address conflicts.

A "procedural" testbench is also defined to exercise the CorePCI macros. The procedural testbench initiates configuration, I/O, and memory transfers. In addition, it has some control over the back end of each slot via the "back-end test control" module.

The arbiter function assigns ownership of the bus to the various masters. Arbitration is accomplished via a simple REQn/GNTn scheme defined by the PCI specification. The PCI monitor continuously checks the PCI bus and reports errors and unusual activity.

The default testbench has two target macros in slots #2 and #3, a Target+DMA function in slot #4, a Target+Master function in slot #5, and a Master-Only function in slot #6 (Figure 7-1).

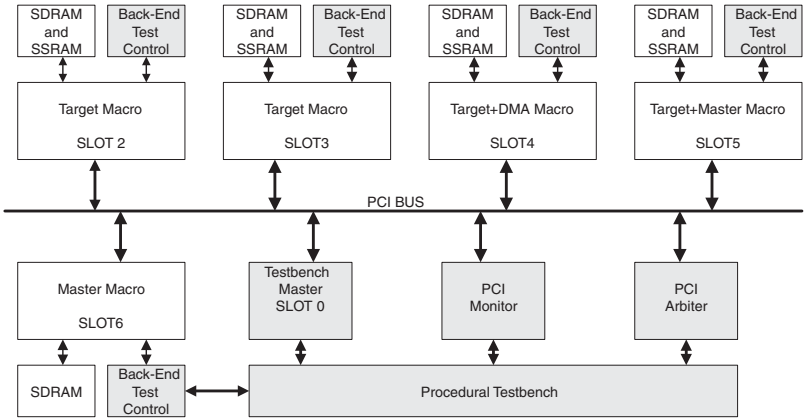

Figure 7-1. Testbench Block Diagram

testbench Hierarchy

- **System** - defined in system32.vhd or system64.vhd files
- **Master_sim** - defined in mast_cfg64.vhd and contains both the system master and procedural testbench
- **Pci_arbiter** - system arbiter defined by arbiter.vhd
- **Pci_monitor** - PCI monitor defined by pci64_mon.vhd
- **Targ32/64sdram_wrp** - top-level chip definition of the Target macro
- **Tdma32/64sdram_wrp** - top-level chip definition of the Target+DMA macro
- **Tmst32/64sdram_wrp** - top-level chip definition of the Target+DMA macro
- **Mast32/64sdram_wrp** - top-level chip definition of the Master macro
- **Sdram_mu** - SDRAM model
- **Ssram_mu** - SSRAM model
- **Be_sys_ctl** - Back-end module under testbench control used to exercise back-end signals. Defined in the "be_control.vhd" file.

Support Packages

- **"Tests_t1.vdh," "Tests_t2.vhd," "Tests_t3.vhd"** - packages defining user tests
- **Startup.vhd** - package defining system start-up procedures
- **Misc.vhd** - miscellaneous types, functions, and procedures
- **Pci_pack.vhd** - PCI specific types, functions, and procedures
- **Mcfgpack.vhd** - type, function, and low-level procedure definitions

General Description of the Procedural Testbench

The procedural test process is in the mast_cfg64.vhd file. This process begins with a power-on reset. Once the reset is complete, the system master (master_sim in the system32.vhd or system64.vhd file) is instructed to strobe each slot with a configuration-read command. Populated slots respond and the test process stores this information in the slot-information ("slot_info") record array. Additional configuration cycles are applied to populated slots to determine what BARs (base-address registers) are enabled, if the slot has DMA capabilities, and how the DMA registers can be accessed (configuration space, I/O space, or back end). Finally, the enabled BARs are assigned addresses according to Table A-1:

Table A-1. PCI Address Assignments

Slot	Bar0 - Memory	Bar1 - I/O	Bar1 - Memory	Bar2 - I/O[a]
1	10000000h	10000000h	11000000h	10000000h or 11000000h
2	20000000h	20000000h	22000000h	20000000h or 22000000h
3	30000000h	30000000h	33000000h	30000000h or 33000000h
4	40000000h	40000000h	44000000h	40000000h or 44000000h

Table A-1. PCI Address Assignments (Continued)

Slot	Bar0 - Memory	Bar1 - I/O	Bar1 - Memory	Bar2 - I/O[a]
5	50000000h	50000000h	55000000h	50000000h or 55000000h
6	60000000h	60000000h	66000000h	60000000h or 66000000h
7	70000000h	70000000h	77000000h	70000000h or 77000000h
8	80000000h	80000000h	88000000h	80000000h or 88000000h

a. The second address is used when Bar1 is an I/O to prevent address space conflicts

The procedures that check and configure the system are in the "startup_pack.vhd" file. Table A-2 describes the procedures.

Table A-2. Testbench Start-Up Procedures

Procedure/ Function	Description
Check_slots	Performs configuration read cycles to each slot to determine which slots are present.
Check_target_reset	Checks the value of Target configuration registers after reset.
Check_tdma_reset	Checks the value of Target+DMA or Target+Master configuration registers after reset.
Check_bars	Determines which BARs are enabled and their type (I/O or memory).
Setup_target_config	Generates the BAR values (see previous Table).

Table A-2. Testbench Start-Up Procedures (Continued)

Procedure/ Function	Description
Config_target	Writes the BAR values into Target configuration space.
Config_tdma	Writes the BAR values into Target+DMA and Target+Master configuration space.

User-Defined Tests

Once the start-up procedures are complete, the testbench is ready to run user-defined tests. A variety of tests are included with the standard testbench and can be run interactively from a menu in the test routine. These tests are defined in Table A-3 and are in the files "tests_t1.vhd," "tests_t2.vhd," and "tests_t3.vhd." The main test process uses information in the slot information ("slot_info") record to determine the appropriateness of each test. For example, if BAR1 is defined to be I/O space, then the "Test_io_cycles" is an appropriate test.

Table A-3. Listing of User-Defined Tests

Test	Description
Test_byte_enable_cfg	Tests the use of byte enables.
Simple_test	Memory read/write test using a single burst.
Read_write_test	Memory read/write tests using bursts transfers of various lengths.
Test_devsel	Test the timing of DEVSELn.
Addr_parity_error	Generates address parity errors and ensures that the Target macro responds with a SERRn correctly depending on settings in its control register.
Test_io_cycles	I/O read/write test. Only valid if BAR1 is enabled to be an I/O.

Table A-3. Listing of User-Defined Tests (Continued)

Test	Description
Test_interrupts	Generates an interrupt on the back end and ensures that the interrupt is correctly posted on the INTAn signal.
Parity_test_target	Generates data parity errors and ensures that the Target macro responds with a PERRn correctly depending on settings in its control register.
Tdma_target_mode	Tests deassertion of IRDYn during a memory read/write burst.
Test_two_targets	Test to ensure that only a single target responds to any given address.
Target_retry_test	Tests retry function using both the BUSY and BE_REQ back-end signals. Only valid if BAR1 is enabled to be memory or I/O.
Ready_variation_test	Target-wait state test. Target-wait states are inserted on the PCI bus (deassertion of TRDYn) by toggling the RD_BE_RDY and WR_BE_RDY back-end signals. An additional test combines both IRDYn and TRDYn deassertion in the same transfer. Only valid if BAR1 is enabled to be memory or I/O.
Target_abort	Tests the target-abort response of a target by asserting the back-end ERROR signal during a transfer. Only valid if BAR1 is enabled to be a memory or I/O.
Tdma_dma_single_transfer	Simple DMA transfer in both the read and write directions.
Tdma_dma_single_transfer_break	Tests the maximum burst-length DMA capability (bits 29-31 of the DMA control register). Only valid if the DMA_CNT_EN is set to a '1'.
Tdma_dma_single_transfer_wait	Tests the response of the DMA master to target-injecting wait states on TRDYn.
Tdma_dma_single_transfer_abort	Tests the response of the DMA master to a target issuing a target-abort cycle.
Tdma_dma_poll_status	Tests operation of the DMA master using a polling method to determine when the DMA is complete.

Table A-3. Listing of User-Defined Tests (Continued)

Test	Description
Tdma_dma_counts	Tests the operation of the DMA master for transfer lengths of 1 - 5. Only valid id DMA_CNT_EN is set to a '1'.
Tdma_mega_tests	Tests the operation of multiple macros being configured to perform DMA operations simultaneously.
Master32/64_test	Tests the operation of the Master-Only function.

Procedures and Functions Used to Build Tests

There are a variety of predefined procedures and functions that are used to build a PCI test. These predefined structures break down into the following categories:

- PCI Transfer Commands (Table A-4)
- DMA Commands (Table A-5)
- Data Generation/Checking (Table A-6)

Table A-4. PCI Transfer Commands

Procedure	Description
Config_write	Directs the system master (Master_sim) to execute a configuration write command.
Config_read	Directs the system master (Master_sim) to execute a configuration read command.
Write_cycle	Directs the system master (Master_sim) to execute a memory write command.
Read_cycle	Directs the system master (Master_sim) to execute a memory read command.
Iowrite_cycle	Directs the system master (Master_sim) to execute an I/O write command.
Ioread_cycle	Directs the system master (Master_sim) to execute an I/O read command.

Table A-5. DMA Commands

Procedure	Description
DMA_setup	Directs the system to configure and enable a DMA with the information provided in the procedure call. If the DMA is mapped to configuration space, then this procedure executes Config_write commands. If the DMA is mapped to I/O space (DMA_IN_IO = '1'), then this procedure executes Iowrite_cycle commands. If the DMA is mapped to the back end (Master or Target+Master), then the procedure directs the back-end processor (BE_SYS_CTL) to execute a write cycle to the DMA registers.
Read_DMA_status	Reads the current values of the DMA-address and control registers. If the DMA is mapped to configuration space, then this procedure executes Config_read commands. If the DMA is mapped to I/O space (DMA_IN_IO = '1'), then this procedure executes Ioread_cycle commands. If the DMA is mapped to the back end (Master or Target+Master), then the procedure directs the back-end processor (BE_SYS_CTL) to execute a read cycle from the DMA registers.
Print_DMA_satus	Prints the value of the PCI address, RAM address, and DMA control register to the standard output device

Table A-6. Data Generation/Checking

Command	Description
Init_data	Generates an array of DWORD data that is sequential from the initial seed defined.
Compare_data	Compares the value of two DWORD arrays and reports any mismatches

System Information

There are two defined record types that contain system information. The first record, "slot_info," contains static information about each slot including presence, BAR types, and assigned addresses. The second record, status, contains dynamic information that is updated during each transfer. The information contained in the records useful to building tests is defined in the Table A-7 and Table A-8.

Table A-7. Description of the "Slot_info" Record

Value[a]	Description
Slot_info(i).present	When true, indicates that a macro is present in this slot.
Slot_info(i).cap_list	When true, indicates that the macro capability list is enabled. This typically means that the hot swap extended capability is enabled.
Slot_info(i).mhz66	When true, indicates that the macro can operate at 66MHz.
Slot_info(i).pci_function	Values are either TARGET or TDMA. TARGET indicates a Target-Only. TDMA indicates a Target+DMA or a Target+Master.
Slot_info(i).dma_count	When true, indicates that the count feature of the DMA-address registers is enabled.
Slot_info(i).bar1	Values are NONE, IO, or MEMORY. NONE indicates that bar1 is not enabled, IO indicates that bar1 is enabled to be I/O space, and MEMORY indicates that bar1 is enabled to be memory space.
Slot_info(i).bar2	Values are NONE or IO. NONE indicates that bar2 is not enabled, IO indicates that bar1 is enabled to be I/O space. This only occurs when the DMA_IN_IO global constant is set to a '1'.
Slot_info(i).bar0_addr	DWORD address mapping for BAR0.
Slot_info(i).bar1_addr	DWORD address mapping for BAR1.

Table A-7. Description of the "Slot_info" Record (Continued)

Value[a]	Description
Slot_info(i).bar2_addr	DWORD address mapping for BAR2.
Slot_info(i).bar0_integer	Integer address mapping for BAR0.
Slot_info(i).bar1_integer	Integer address mapping for BAR1.
Slot_info(i).bar2_integer	Integer address mapping for BAR2.
Slot_info(i).dma_loc	Values are CONFIG, IO, and BACKEND. CONFIG indicates that the DMA registers are mapped to configuration space, IO indicates mapping to I/O space, and BACKEND indicates back-end control of the DMA registers.

a. i is the array index for the various slots (1-8).

Table A-8. Description of the Status Record

Value	Description
Status.perr	When true, indicates that the PERRn signal was driven low during the previous command.
Status.tabort	When true, indicates that a target terminated the previous transfer with a target abort.
Status.retry	When true, indicates that a target responded with at least one retry cycle during the previous transfer.
Status.no_devsel	When true, indicates that a target did not respond to a command and the cycle terminated with a master abort.
Status.interrupt	When true, indicates that the INTAn line is being driven low by a macro.

Table A-8. Description of the Status Record

Value	Description
Status.serr	When true, indicates that the SERRn signal was asserted during the previous command.

Additional Control of the System Master and Back End

To control wait states on the PCI bus, each PCI transfer command passes two integer variables, "ilatency" and "tlatency." When "ilatency" is defined as a positive integer, the system master inserts the number of wait states (on "IRDYn") defined by "ilatency" between each transfer. When "tlatency" is defined to be a positive integer, the BE_SYS_CTL block uses this to insert wait states on the back end by toggling the RD_BE_RDY or WR_BE_RDY signals. Toggling these signals results in a toggling action on the "TRDYn" signals during burst transfers.

To control back-end signals, the PCI transfer commands pass a T_ERRORTYPE variable ("errortype") and an INTEGER variable ("errorpos"). These two variables can be used to control a variety of master and back-end control signals. "Errortype" defines the type of control and "errorpos" indicates at what point in the cycle the control should be activated. Table A-9 defines the values of "errortype" and their impact on PCI and back-end control signals.

Table A-9. Errortype and Back-End Control Signals

Error Type	Description
None	Default. No action.
Par_addr	Causes the system master to generate a parity error during the address phase. "Errorpos" has no impact on this function.
Par_data	Causes the system master to generate a data parity error on the cycle defined by "errorpos."
Terror	Causes the back-end ERROR signal (output of BE_SYS_CTL) to be asserted on the nth cycle defined by "errorpos."

Table A-9. Errortype and Back-End Control Signals

Error Type	Description (Continued)
Tbusy_cycle	Causes the back-end BUSY signal (output of BE_SYS_CTL) to be asserted on the nth cycle defined by "errorpos."

Testing an Application-Specific PCI Macro

The existing testbench tests the following macros:

- Target-Only with back-end SDRAM and SSRAM (slots #2 and #3)
- Target+DMA with back-end SDRAM and SSRAM (slot #4)
- Target+Master with back-end SDRAM and SSRAM (slot #5)
- Master-Only with back-end SDRAM (slot #6)

The testbench is designed to detect which slots are present and the type of macro. Appropriate tests are then run to exercise these functions.

Modifying the System32.vhd or System64.vhd Top Level

To test your application-specific macro, you can either add a new slot (e.g. slot #1) or you can modify an existing slot. To improve run times, you can also remove any slots that you are not interested in. When building a modified "system" file, you should note the following:

1. You should always populate at least two slots. One of the target tests ("test_two_targets") and all of the DMA tests use at least two macros to exchange data.
2. If you add a Master-Only function, you need to hardwire the "slot_info(i).dma_loc <= BACKEND" in the "mast_cfg64.vhd" file because the testbench does not have visibility to these functions.
3. Most or all of the tests that control back-end signals do not work correctly on your application because these signals are typically not available.

Creating a New Test

The easiest method to create a new test is to modify an existing test that is close to what you need. This section provides a very basic overview to creating a new test. Many test examples are in the "tests_t1.vhd," "tests_t2.vhd," and "tests_t3.vhd" files. Test procedures are called from the "mast_cfg64.vhd" file. The basic steps for adding a test are as follows:

1. **Create a new test name and decide what information needs to be passed to the test procedure.** Basic information includes slot number, address, and "slot_info." Errors, status, and control should always be passed by all tests.
2. **Instantiate the test in the "mast_cfg64.vhd" file.** There is an existing interactive framework to run individual tests or groups of tests. You can use this structure, modify it, or create your own.
3. **Create and define the test procedure in one of the existing test packages ("tests_t1.vhd," etc.) or create your own test file.** If you create your own, you need to reference this package in the "mast_cfg64.vhd" file.

The basic steps to test a Target function are as follows:

1. **Initialize data to read and write using the "init_data" function.**
2. **Execute a write command using a configuration, memory, or I/O command.** Memory and I/O only need an address. Configuration commands requires an address and a slot number.
3. **Execute a read command from the same address.**
4. **Compare the written data to the read data using the "compare_data" procedure.**

During a test, if you are expecting some event to occur, like a target abort, then you may test this target event two ways. First, the monitor reports unusual activity on the standard output device like target abort, retry, master abort, parity errors, etc. You check the monitor visually during a test or you can use the values in the "status" record to confirm an event.

The basic steps to test a DMA/Master are as follows:

1. **Initialize data to read and write using the Init_data function.**

2. **Using a memory write command, write this data into a known location.**
3. **To initiate a DMA command, use the "DMA_setup" procedure by defining the PCI address to be the address from Step 2.** The first DMA should move the known data to the Master's back end.
4. **To verify completion of the DMA, use the "Read_dma_status" and "check_dma_status."** You can also view the contents of the DMA register using the "print_dma_status."
5. **Test the DMA write function.** The new data can be read from the back-end memory and placed in a new location out on the PCI bus. Again, this is initiated using the "DMA_setup" procedure and checked using the "read_dma_status," "check_dma_status," and "print_dma_status."
6. **Verify correct transfer.** To do so, use a memory read command to read the data from the newest address location.
7. **Use the "compare_data" procedure to check the data.**

Command Syntax

The following sections define the syntax for the most commonly used procedures and functions.

Config_read

This reads data from configuration space. "Config_read" reads the number of DWORDs defined by "words" from the slot # defined by "slot" beginning from the byte address defined by "addr." The read data is stored in the dword_array "data." "Errortype" and "errorpos" are optional in this procedural call.

```
procedure config_read  ( slot : in  integer range 1 to 7;
addr : in  integer range 0 to 511;
words : in  integer range 1 to 16;
data : out dword_array;
signal control : out T_MCFG_CONTROL;
signal status : in  T_MCFG_STATUS;
```

```
errortype : T_ERRORTYPE := none;
errorpos : INTEGER := -1);
```

Config_write

This writes data to configuration space. “Config_write” writes data defined in the “dword_array” “data” to the slot # defined by “slot” beginning at the address defined by “addr.” The length of the write is defined by the integer “words.” Cbe, errortype, and errorpos are optional in this procedural call. Cbe defines byte-lane enables for the write and is active high.

```
procedure config_write ( slot  : integer range 1 to 7;
addr : integer range 0 to 511;
words : integer range 1 to 16;
data : dword_array;
signal control : out T_MCFG_CONTROL;
signal status : in  T_MCFG_STATUS;
cbe : nibble := "1111";
errortype : T_ERRORTYPE := none;
errorpos : INTEGER := -1 );
```

DMA_setup

The “DMA_setup” procedure writes information defined in the procedural call to the DMA registers and initiates a DMA transfer. “Slot” is the slot # of the DMA/Master that executes the transfer. “Slot_info” is used to inform the procedure where the DMA registers are located (configuration space, I/O space, or back-end). “Pciaddr” is the byte address on the PCI bus for the transfer and is written into the PCI Address Register. “Ramaddr” is the back-end byte address. “Cyc_type” defines the type of cycle to be either configuration (“config”), I/O, (io), memory (memory), or interrupt acknowledge (“int_ack”). Count defines the number of DWORDs to be transferred. “Direct” defines the direction of the transfer. A DMA read is defined to be “PCI_SRAM” and a DMA write is defined to be “SRAM_PCI”. “Length” is optional and is used to define the maximum burst-length bits in the DMA control register. “Errortype” and “errorpos” are both optional.

```
procedure dma_setup ( slot : integer range 1 to 7;
```

```
slot_info : T_SLOT_INFO_ARRAY;
pciaddr : integer;
ramaddr : integer;
cyc_type : T_MASTER_CYC;
count : integer range 0 to 1023;
direct : T_DMADIRECTION;
signal control : out T_MCFG_CONTROL;
signal status : in  T_MCFG_STATUS;
length : integer := 0;
errortype : T_ERRORTYPE := none;
errorpos : INTEGER := -1);
```

Read_DMA_status

The" read_DMA_status" procedure reads the contents of the DMA registers and stores the value in the "dma" record. "Slot" defines which slot to read from. "Slot_info" defines the location of the DMA registers (configuration, I/O, or back-end). "Errortype" and "errorpos" are optional.

```
procedure read_dma_status( dma : out T_DMA_RECORD;
slot : integer range 1 to 7;
slot_info : T_SLOT_INFO_ARRAY;
signal control : out T_MCFG_CONTROL;
signal status : in  T_MCFG_STATUS;
errortype : T_ERRORTYPE := none;
errorpos : INTEGER := -1);
```

Check_DMA_status

The "check_DMA_status" procedure checks the information stored in the dma record to determine if the DMA transfer completed correctly. For normal transfers, the "discon" variable should be set to false. If you anticipate that the DMA will not complete correctly because of a target or master abort, then "discon" should be set to true. If an incorrect termination occurs, then the procedure prints the "pciaddr," "ramaddr," and "count" values to the standard output device.

```
procedure check_dma_status ( errors : inout INTEGER;
dma : T_DMA_RECORD;
```

```
pciaddr : INTEGER;
ramaddr : INTEGER;
count : INTEGER := 0;
discon : BOOLEAN := FALSE );
```

Compare_data

The "compare_data" procedure compares two DWORD_ARRAYs ("exp" and "got") and reports any mismatches to the standard output device.

```
procedure compare_data ( errors : inout integer;
msg : STRING;
exp : DWORD_ARRAY;
got : DWORD_ARRAY );
```

Print_DMA_status

The "print_dma_status" procedure prints the information stored in the dma record to the standard output device.

```
procedure print_dma_status ( slot : INTEGER;
dma : T_DMA_RECORD );
```

Init_data

This function creates a DWORD_ARRAY of sequential data. The start address of the sequence is a type string that should be hex characters (0-9, A-F). The size of the array is defined by size.

```
function init_data( seed : STRING; size : INTEGER) return
DWORD_ARRAY;
```

Ioread_cycle

This reads data from I/O space. "Ioread_cycle" executes an I/O read command ("0010") on the PCI bus and read the number of DWORDs defined by "burst" beginning with the PCI address defined by "addr". The read data is stored in the "dword_array" "data." "Tlatency," "ilatency," "errortype," and "errorpos" are optional in this procedural call.

```
procedure ioread_cycle  ( addr  : in  integer;
burst : in  integer range 0 to 511;
```

```
data : out dword_array;
signal control : out T_MCFG_CONTROL;
signal status : in  T_MCFG_STATUS;
tlatency : in  INTEGER := DEF_TLATENCY;
ilatency : in  INTEGER := DEF_ILATENCY;
errortype : T_ERRORTYPE := none;
errorpos : INTEGER := -1);
```

Iowrite_cycle

This writes data to I/O space. “Iowrite_cycle” executes an I/O write command (“0011”) on the PCI bus and writes the data stored in the “dword_array” “data” to the PCI address defined by “addr.” The length of the write in DWORDs is defined by the integer “burst.” “Tlatency,” “ilatency,” “errortype,” and “errorpos” are optional in this procedural call.

```
procedure iowrite_cycle ( addr  : integer;
burst : integer range 0 to 511;
data : dword_array;
signal control: out T_MCFG_CONTROL;
signal status : in  T_MCFG_STATUS;
tlatency : in  INTEGER := DEF_TLATENCY;
ilatency : in  INTEGER := DEF_ILATENCY;
errortype : T_ERRORTYPE := none;
errorpos : INTEGER := -1);
```

Read_cycle

This reads data from memory space. “Read_cycle” executes a memory read command (“0110”) on the PCI bus and reads the number of DWORDs defined by “burst” beginning with the PCI address defined by “addr”. The read data is stored in the “dword_array” “data.” “Tlatency,” “ilatency,” “errortype,” and “errorpos” are optional in this procedural call.

```
procedure read_cycle  ( addr  : in  integer;
burst : in  integer range 0 to 511;
data : out dword_array;
signal control : out T_MCFG_CONTROL;
```

```
signal status : in  T_MCFG_STATUS;
tlatency : in  INTEGER := DEF_TLATENCY;
ilatency : in  INTEGER := DEF_ILATENCY;
errortype : T_ERRORTYPE := none;
errorpos : INTEGER := -1);
```

Write_cycle

This writes data to memory space. “Write_cycle” executes a memory write command (“0111”) on the PCI bus and writes the data stored in the “dword_array” “data” to the PCI address defined by “addr.” The length of the write in DWORDs is defined by the integer “burst.” “Tlatency,” “ilatency,” “errortype,” and “errorpos” are optional in this procedural call.

```
procedure write_cycle ( addr  : integer;
burst : integer range 0 to 511;
data : dword_array;
signal control : out T_MCFG_CONTROL;
signal status : in  T_MCFG_STATUS;
tlatency : in  INTEGER := DEF_TLATENCY;
ilatency : in  INTEGER := DEF_ILATENCY;
errortype : T_ERRORTYPE := none;
errorpos : INTEGER := -1);
```

Defined Types

The following information defines the various types used by the testbench. These definitions are for reference only.

Data Types

- type **T_ABORT_TYPE** is (retry, abort);
- subtype **BYTE** is “std_logic_vector (7 downto 0)”;
- type **BYTE_ARRAY** is array (INTEGER range <>) of BYTE;
- type **BOOLEAN_VECTOR** is array (INTEGER range <>) of BOOLEAN;

- type **DMA_REG** is (NONE, IO, CONFIG, BACKEND);
- subtype **DWORD** is “std_logic_vector (31 downto 0)”;
- type **DWORD_ARRAY** is array (INTEGER range <>) of DWORD;
- type **FUNCTION_TYPE** is (TARGET,TDMA);
- type **INTEGER_ARRAY** is array (INTEGER range <>) of INTEGER;
- type **IO_OR_MEM** is (IO, MEMORY, NONE);
- subtype **NIBBLE** is “std_logic_vector (3 downto 0)”;
- type **T_PCI_COMMAND** is (IACK,SPECIAL, IOREAD, IOWRITE, MEMREAD,MEMWRITE,CFGREAD, CFGWRITE, MRDMULT, DUALADDR, MRDLINE, MEMWINVAL, RESERVED, UNKNOWN);
- subtype **WORD** is “std_logic_vector (15 downto 0)”;
- type **WORD_ARRAY** is array (INTEGER range <>) of WORD;

Record Types

- type **T_ARB_CONTROL** is

```
record
max_time : INTEGER;
backtoback : BOOLEAN;
end record;
```

- type **T_BE_CONTROL** is

```
record
LATENCY : INTEGER range -23 to 23;
SET_BUSY : INTEGER range -1 to 31;
SET_ERROR : INTEGER range -1 to 31;
SET_FATAL_ERROR : INTEGER range -1 to 31;
BUSY_CYCLE : INTEGER range -1 to 31;
ARB_LENGTH : INTEGER range -1 to 256;
ARB_WAIT : INTEGER range -1 to 256;
ARB_INIT : BOOLEAN;
SET_INTERRUPT : BOOLEAN;
MASTER_LOAD : BOOLEAN;
MASTER_READ : BOOLEAN;
```

```
MASTER_DATA : DWORD_ARRAY (0 to 3);
end record;
```

- type **T_BE_CONTROL_ARRAY** is array (INTEGER range <>) of "T_BE_CONTROL" ;
- type **T_BE_STATUS** is

```
record
RETRY : BOOLEAN;
LAST_ADDRESS : INTEGER;
DATA : DWORD_ARRAY (0 to 3);
MASTER_READ_DONE : BOOLEAN;
MASTER_LOAD_DONE : BOOLEAN;
end record;
```

- type **T_BE_STATUS_ARRAY** is array (INTEGER range <>) of "T_BE_STATUS" ;
- type **t_CONFIGURATION** is

```
record
DeviceID : WORD;
VendorID : WORD;
Status : WORD;
Command : WORD;
ClassCode : STD_LOGIC_VECTOR(23 downto 0);
RevisionId : BYTE;
BIST : BYTE;
HeadType : BYTE;
Latentime : BYTE;
CacheSize : BYTE;
BaseAddress: DWORD_ARRAY( 1 to 6);
CardBusPtr : DWORD;
SubSysID : WORD;
SubSysVID : WORD;
ExpanROM : DWORD;
CapPtr : DWORD;
Reserved2 : DWORD;
MaxLatency : BYTE;
```

```
MinGrant : BYTE;
IntPin : BYTE;
Intline : BYTE;
end record;
```

- type **T_DMADIRECTION** is (SRAM_PCI , PCI_SRAM);
- type **T_DMA_RECORD** is

```
record
PCI_ADDRESS : INTEGER;
RAM_ADDRESS : INTEGER;
WAIT_STATE : BOOLEAN;
DIRECTION : T_DMADIRECTION;
CONFIG_CYC : BOOLEAN;
INT_ACK_CYC : BOOLEAN;
IO_CYC : BOOLEAN;
COUNT : INTEGER range 0 to 1023;
ENABLE : BOOLEAN;
ABORT : BOOLEAN;
CLEARDONE : BOOLEAN;
INT_ENABLE : BOOLEAN;
INT_ACTIVE : BOOLEAN;
EINT_ENABLE : BOOLEAN;
EINT_ACTIVE : BOOLEAN;
CYCLE64 : BOOLEAN;
ERROR : BOOLEAN;
DONE : BOOLEAN;
REQUEST : BOOLEAN;
BURST_LENGTH : INTEGER range 0 to 64;
end record;
```

- type **T_ERRORTYPE** is (none, par_addr, par_data, tbusy, tferror, terror, tbusy_cycle, addr_cache, addr_01,addr_11, backtobackWR, backtobackWW, no_devsel, maskerror);
- type **T_MASTER_CYC** is (memory, config, io, int_ack);
- type **T_MCFG_CONTROL** is

```
record
```

```
START : BOOLEAN;
WORDS : NATURAL range 0 to 511;
CYCLE_TYPE : T_PCI_COMMAND;
ADDRESS : INTEGER;
DATA : DWORD_ARRAY ( 0 to 511);
BYTES : STD_LOGIC_VECTOR( 3 downto 0);
ERRORTYPE : T_ERRORTYPE;
ERRORPOS : INTEGER;
LATENCY : INTEGER;
MON_CONTROL : T_MON_CONTROL;
ARB_CONTROL : T_ARB_CONTROL;
BE_CONTROL : T_BE_CONTROL_ARRAY ( 2 to 6);
end record;
```

- type **T_MCFG_STATUS** is

```
record
CLK : STD_LOGIC;
BUSY : BOOLEAN;
DATA : DWORD_ARRAY ( 0 to 511);
PERR : BOOLEAN;
ECYCLE : INTEGER;
DEVSEL : INTEGER;
STOP : BOOLEAN;
TABORT : BOOLEAN;
RETRY : BOOLEAN;
SERR : BOOLEAN;
LASTADDR : INTEGER;
INTERRUPT : BOOLEAN;
NO_DEVSEL : BOOLEAN;
FRAME_ON : INTEGER;
BUS_DRIVEN : BOOLEAN;
BEND : T_BE_STATUS_ARRAY ( 2 to 6);
MASTER_ACTIVE : BOOLEAN;
end record;
```

- type **T_MON_CONTROL** is

```
record
```

```
TRACE_ENABLE : BOOLEAN;
PAR_CHK_ENABLE : BOOLEAN;
end record;
```

- type **T_SLOT_INFO** is

```
record
PRESENT : BOOLEAN;
CAP_LIST : BOOLEAN;
MHZ66 : BOOLEAN;
PCI_FUNCTION : FUNCTION_TYPE;
DMA_COUNT : BOOLEAN;
BAR0 : IO_OR_MEM;
BAR1 : IO_OR_MEM;
BAR2 : IO_OR_MEM;
BAR0_ADDR : DWORD;
BAR1_ADDR : DWORD;
BAR2_ADDR : DWORD;
BAR0_INTEGER : INTEGER;
BAR1_INTEGER : INTEGER;
BAR2_INTEGER : INTEGER;
DMA_LOC : DMA_REG;
end record;
```

- type **T_SLOT_INFO_ARRAY** is array (1 to 8) of “T_SLOT_INFO”;

B

Product Support

Actel backs its products with various support services including Customer Service, a Customer Applications Center, a web site, an FTP site, electronic mail, and worldwide sales offices. This appendix contains information about contacting Actel and using these support services.

Actel U.S. Toll-Free Line

Use the Actel toll-free line to contact Actel for sales information, technical support, requests for literature about Actel and Actel products, Customer Service, investor information, and using the Action Facts service.

The Actel toll-free line is (888) 99-ACTEL.

Customer Service

Contact Customer Service for nontechnical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From Northeast and North Central U.S.A., call (408) 522-4480.
From Southeast and Southwest U.S.A., call (408) 522-4480.
From South Central U.S.A., call (408) 522-4434.
From Northwest U.S.A., call (408) 522-4434.
From Canada, call (408) 522-4480.
From Europe, call (408) 522-4252 or +44 (0) 1256 305600.
From Japan, call (408) 522-4743.
From the rest of the world, call (408) 522-4743.
Fax, from anywhere in the world (408) 522-8044.

Customer Applications Center

Actel staffs its Customer Applications Center with highly skilled engineers who can help answer your hardware, software, and design questions. The Applications Center spends a great deal of time creating application notes and answers to FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your question(s).

Guru Automated Technical Support

Guru is a web-based automated technical support system accessible through the Actel home page (**http://www.actel.com/guru/**). Guru provides answers to technical questions about Actel products. Many answers include diagrams, illustrations, and links to other resources on the Actel web site. Guru is available 24 hours a day, seven days a week.

Web Site

Actel has a World Wide Web home page where you can browse a variety of technical and nontechnical information. Use a Net browser (Netscape recommended) to access Actel's home page.

The URL is **http://www.actel.com**. You are welcome to share the resources provided on the Internet.

Be sure to visit the Technical Documentation area on our web site, which contains information regarding products, technical services, current manuals, and release notes.

You can visit the Product Support area of the Actel website from your Designer software. Click the Product Support button in your Designer Main Window to access the latest datasheets, application notes, and more.

FTP Site

Actel has an anonymous FTP site located at **ftp://ftp.actel.com**. Here you can obtain library updates, software patches, design files, and data sheets.

Contacting the Customer Applications Center

Highly skilled engineers staff the Customer Applications Center from 7:30 A.M. to 5:00 P.M., Pacific Time, Monday through Friday. Several ways of contacting the Center follow:

Electronic Mail

You can communicate your technical questions to our e-mail address and receive answers back by e-mail, fax, or phone. Also, if you have design problems, you can e-mail your design files to receive assistance. We constantly monitor the e-mail account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support e-mail address is **tech@actel.com**.

Telephone

Our Technical Message Center answers all calls. The center retrieves information, such as your name, company name, phone number and your question, and then issues a case number. The Center then forwards the information to a queue where the first available application engineer receives the data and returns your call. The phone hours are from 7:30 A.M. to 5:00 A.M., Pacific Time, Monday through Friday.

The Customer Applications Center number is (800) 262-1060.

European customers can call +44 (0) 1256 305600.

Worldwide Sales Offices

Headquarters

Actel Corporation
955 East Arques Avenue
Sunnyvale, California 94086
Toll Free: 888.99.ACTEL

Tel: 408.739.1010
Fax: 408.739.1540

US Sales Offices

California

Bay Area
Tel: 408.328.2200
Fax: 408.328.2358

Irvine
Tel: 949.727.0470
Fax: 949.727.0476

Newbury Park
Tel: 805.375.5769
Fax: 805.375.5749

Colorado

Tel: 303.420.4335
Fax: 303.420.4336

Florida

Tel: 407.977.6846
Fax: 407.977.6847

Georgia

Tel: 770.277.4980
Fax: 770.277.5896

Illinois

Tel: 847.259.1501
Fax: 847.259.1575

Massachusetts

Tel: 978.244.3800
Fax: 978.244.3820

Minnesota

Tel: 651.917.9116
Fax: 651.917.9114

New Jersey

Tel: 609.517.0304

North Carolina

Tel: 919.654.4529
Fax: 919.674.0055

Pennsylvania

Tel: 215.830.1458
Fax: 215.706.0680

Texas

Tel: 972.235.8944
Fax: 972.235.965

International Sales Offices

Canada
235 Stafford Rd. West,
Suite 106
Nepean, Ontario K2H 9C1

Tel: 613.726.7575
Fax: 613.726.8666

France
Actel Europe S.A.R.L.
361 Avenue General de Gaulle
92147 Clamart Cedex

Tel: +33 (0)1.40.83.11.00
Fax: +33 (0)1.40.94.11.04

Germany
Lohweg 27
85375 Neufahrn

Tel: +49 (0)8165.9584.0
Fax: +49 (0)8165.9584.1

Italy
Via de Garibaldini, No. 5
20019 Settimo Milanese,
Milano, Italy

Japan
EXOS Ebisu Building 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150

Tel: +81 (0)3.3445.7671
Fax: +81 (0)3.3445.7668

Korea
30th Floor, ASEM Tower,
159-1 Samsung-dong,
Kangam-ku,
Seoul, Korea

Tel: +82.2.6001.3382
Fax: +82.2.6001.3030

United Kingdom
Maxfli Court,
Riverside Way
Camberley,
Surrey GU15 3YL

Tel: +44 (0)1276.401452
Fax: +44 (0)1276.401490

Index

W