LogiCORE™
PCI Implementation Guide
Version 3.0

The Xilinx logo shown above is a registered trademark of Xilinx, Inc.

The shadow X shown above is a trademark of Xilinx, Inc.

LogiCORE™ PCI Implementation Guide Version 3.0

LogiCORE/AllianceCORE Fax:	+1 408-626-6440
Xilinx Home Page URL:	www.xilinx.com
Technical Support URL:	support.xilinx.com
LogiCORE PCI Solutions URL:	www.xilinx.com/pci
Information & Feedback E-mail:	logicore@xilinx.com

2100 Logic Drive
San Jose, CA 95124
United States of America
Telephone: +1 408-559-7778
Fax: +1 408-559-7114

November 7, 2000

Contents

Conventions

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

- `Courier font` denotes the following items:
  - Signals on PCI Bus side of the LogiCORE PCI Interface

    `FRAME_IO` (PCI Interface signal name)

    `FRAME#` (PCI Bus signal name)
  - Signals within the user application

    `BACK_UP, START`
  - Command line input and output

    `setenv XIL_MAP_LOC_CLOSED`
  - World Wide Web URLs

    `http://www.xilinx.com`
  - HDL pseudocode

    ```
    assign question = to_be | !to_be;
    assign cannot = have_cake & eat_it;
    ```
  - Design file names

    `pcim_top.v, pcim_top.vhd`
- **`Courier bold`** denotes the following items:
  - Signals on the user side of the LogiCORE PCI Interface

    `ADDR_VLD`

- Menu selections or button presses

  FILE -> OPEN

- *Italic font* denotes the following items:
  - Variables in a statement for which you must supply values

    `ngdbuild` *design_name*
  - References to other manuals

    See the *Libraries Guide* for more information.
  - Emphasis in text

    It is not a bug, it is a *feature*.
- Dark shading indicates items that are not supported or reserved:

SDONE_I	in / out	Snoop Done signal. Not Supported.

- Square brackets "[ ]" indicate an optional entry or a bus index:

  `ngdbuild` [*option_name*] *design_name*

  `DATA[31:0]`
- A vertical or horizontal ellipsis indicates repetitive material that has been omitted.

  `A B C ... X Y Z`
- The use of "`fn(SIG1 . . . SIGn)`" in an HDL pseudocode fragment should be interpreted as "combinational function of signals SIG1 through SIGn.

  ```
  SUM = fn(A, B, Cin);
  ```
- A vertical bar "|" separates items in a list of choices.

  `OPTION = [enabled|disabled]`
- The prefix "`0x`" or the suffix "`h`" indicate hexadecimal notation.

  A read of address `0x00110373` returned `45524943h`.
- A "Q" on a signal means that it is registered; this is only used for PCI Bus signals that are delayed by one cycle. An "_N" means the signal is active low

  PERRQ_N is both registered and active low.

Chapter 1

Getting Started

Thank-you for purchasing a LogiCORE PCI interface from Xilinx!

The Xilinx LogiCORE PCI interface provides you with a fully verified, pre-implemented PCI Bus interface. This interface is available in both 32-bit and 64-bit versions, with support for operation at 33 MHz and 66 MHz. It is designed to support both Verilog-HDL and VHDL.

This book is intended to serve as a reference for use during the implementation phase of a project using the Xilinx PCI interface. This book is comprehensive in nature; some portions may not apply to your design depending on which version of the LogiCORE PCI interface you are using.

This book covers the supported design flows for the 64-bit and 32-bit LogiCORE PCI interfaces targeting devices based on the Virtex architecture.

An example design, “Ping”, is included with the LogiCORE PCI interface to demonstrate design flows. Please take the time to simulate, synthesize, and implement the example design.

Other Documentation

For more details on the LogiCORE PCI interface, refer to the following documents located on the Xilinx PCI Lounges, accessible from the `www.xilinx.com/pci` web site:

- *LogiCORE PCI Databook*
- *LogiCORE PCI Release Notes*
- *LogiCORE PCI Design Guide*

Further information is available in the Mindshare *PCI Systems Architecture* text, which is included in the Xilinx PCI Design Kit, and the

PCI Local Bus Specification, which is available from the PCI Special Interest Group.

Technical Support

The fastest method for obtaining PCI specific technical support for the LogiCORE PCI interface is through the `support.xilinx.com` web site. Questions are routed to a team of engineers with specific expertise in using the LogiCORE PCI interface.

Xilinx will provide technical support for use of the LogiCORE product as described in the *LogiCORE PCI Design Guide* and the *LogiCORE PCI Implementation Guide*. Xilinx cannot guarantee timing, functionality, or support of the LogiCORE product for designs that do not follow these guidelines.

Chapter 2

Family Specific Considerations

This chapter discusses design considerations specific to the LogiCORE PCI interface targeting Virtex devices. Please read this chapter carefully, as it contains important information.

Design Support

Refer to Table 2-1 “Device and Interface Selection Table” for a list of supported device and interface combinations. Each entry in the table consists of a device, a bus interface type, and two or three specific implementation files.

Table 2-1 Device and Interface Selection Table

Supported Device	Bus Type	Wrapper File	Constraints File Guide File
XC2S100-FG456-6	33 MHz 5.0 V 64-bit	pcim_lc_33_5_s	2s100fg456_64_33.ucf no guide file
XC2S100-FG456-6	33 MHz 3.3 V 64-bit	pcim_lc_33_3_s	2s100fg456_64_33.ucf no guide file
XC2S150-FG456-6	33 MHz 5.0 V 64-bit	pcim_lc_33_5_s	2s150fg456_64_33.ucf no guide file
XC2S150-FG456-6	33 MHz 3.3 V 64-bit	pcim_lc_33_3_s	2s150fg456_64_33.ucf no guide file

Table 2-1 Device and Interface Selection Table

Supported Device	Bus Type	Wrapper File	Constraints File Guide File
XCV100E-BG352-6	66 MHz 3.3 V 64-bit	pcim_lc_66_3_d	v100ebg352_64_66.ucf v100ebg352_64_66.ncd
XCV100E-BG352-6	33 MHz 3.3 V 64-bit	pcim_lc_33_3_s	v100ebg352_64_33.ucf no guide file
XCV300-BG432-6	66 MHz 3.3 V 64-bit	pcim_lc_66_3_d	v300bg432_64_66.ucf v300bg432_64_66.ncd
XCV300-BG432-5	33 MHz 5.0 V 64-bit	pcim_lc_33_5_s	v300bg432_64_33.ucf no guide file
XCV300-BG432-5	33 MHz 3.3 V 64-bit	pcim_lc_33_3_s	v300bg432_64_33.ucf no guide file
XCV300E-BG432-6	66 MHz 3.3 V 64-bit	pcim_lc_66_3_d	v300ebg432_64_66.ucf v300ebg432_64_66.ncd
XCV300E-BG432-6	33 MHz 3.3 V 64-bit	pcim_lc_33_3_s	v300ebg432_64_33.ucf no guide file
XCV1000-FG680-6	66 MHz 3.3 V 64-bit	pcim_lc_66_3_d	v1000fg680_64_66.ucf v1000fg680_64_66.ncd
XCV1000-FG680-5	33 MHz 5.0 V 64-bit	pcim_lc_33_5_s	v1000fg680_64_33.ucf no guide file
XCV1000-FG680-5	33 MHz 3.3 V 64-bit	pcim_lc_33_3_s	v1000fg680_64_33.ucf no guide file

Table 2-1 Device and Interface Selection Table

Supported Device	Bus Type	Wrapper File	Constraints File Guide File
XCV1000E-FG680-6	66 MHz 3.3 V 64-bit	pcim_lc_66_3_d	v1000efg680_64_66.ucf v1000efg680_64_66.ncd
XCV1000E-FG680-6	33 MHz 3.3 V 64-bit	pcim_lc_33_3_s	v1000efg680_64_33.ucf no guide file
XC2V1000-FG456-5	66 MHz 3.3 V 64-bit	pcim_lc_66_3_s	2v1000fg456_64_66.ucf 2v1000fg456_64_66.ncd
XC2V1000-FG456-4	33 MHz 3.3 V 64-bit	pcim_lc_33_3_s	2v1000fg456_64_33.ucf no guide file
XC2S30-PQ208-5	33 MHz 5.0 V 32-bit	pcim_lc_33_5_s	2s030pq208_32_33.ucf no guide file
XC2S30-PQ208-5	33 MHz 3.3 V 32-bit	pcim_lc_33_3_s	2s030pq208_32_33.ucf no guide file
XC2S50-PQ208-5	33 MHz 5.0 V 32-bit	pcim_lc_33_5_s	2s050pq208_32_33.ucf no guide file
XC2S50-PQ208-5	33 MHz 3.3 V 32-bit	pcim_lc_33_3_s	2s050pq208_32_33.ucf no guide file
XC2S100-PQ208-5	33 MHz 5.0 V 32-bit	pcim_lc_33_5_s	2s100pq208_32_33.ucf no guide file
XC2S100-PQ208-5	33 MHz 3.3 V 32-bit	pcim_lc_33_3_s	2s100pq208_32_33.ucf no guide file

Table 2-1 Device and Interface Selection Table

Supported Device	Bus Type	Wrapper File	Constraints File Guide File
XC2S150-PQ208-5	33 MHz 5.0 V 32-bit	pcim_lc_33_5_s	2s150pq208_32_33.ucf no guide file
XC2S150-PQ208-5	33 MHz 3.3 V 32-bit	pcim_lc_33_3_s	2s150pq208_32_33.ucf no guide file
XCV100E-BG352-6	33 MHz 3.3 V 32-bit	pcim_lc_33_3_s	v100ebg352_32_33.ucf no guide file
XCV300-BG432-5	33 MHz 5.0 V 32-bit	pcim_lc_33_5_s	v300bg432_32_33.ucf no guide file
XCV300-BG432-5	33 MHz 3.3 V 32-bit	pcim_lc_33_3_s	v300bg432_32_33.ucf no guide file
XCV300E-BG432-6	33 MHz 3.3 V 32-bit	pcim_lc_33_3_s	v300ebg432_32_33.ucf no guide file
XCV1000-FG680-5	33 MHz 5.0 V 32-bit	pcim_lc_33_5_s	v1000fg680_32_33.ucf no guide file
XCV1000-FG680-5	33 MHz 3.3 V 32-bit	pcim_lc_33_3_s	v1000fg680_32_33.ucf no guide file
XCV1000E-FG680-6	33 MHz 3.3 V 32-bit	pcim_lc_33_3_s	v1000efg680_32_33.ucf no guide file
XC2V1000-FG456-4	33 MHz 3.3 V 32-bit	pcim_lc_33_3_s	2v1000fg456_32_33.ucf no guide file

The wrapper files, located in the <Install Path>/*hdl*/src/wrap directory, are actually different "flavors" of the pcim_lc.*hdl* file located in the <Install Path>/*hdl*/src/xpci directory.

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

When you are ready to begin a new design, copy the appropriate wrapper file from the wrap/ directory into the xpci/ directory, and rename it as pcim_lc.*hdl*. Absolutely do not modify the wrapper files.

The constraints files, located in the <Install Path>/*hdl*/src/ucf directory, contain various constraints required for the PCI interface. Constraints files must always be used while processing a design. Each constraints file is specific to a particular device and PCI interface.

Based on your specific device and interface selection, use the appropriate constraints file from the ucf/ directory when processing designs with the Xilinx implementation tools.

The guide files, located in the <Install Path>/*hdl*/src/guide directory, contain routing information required for high performance versions of the PCI interface. Guide files must always be used when required. Each guide file is specific to a particular device and PCI interface.

Table 2-2 "Guide File Information" specifies how many guided components and guided connections are included in each guide file. Refer to this table after implementation to verify your results.

Table 2-2 Guide File Information

Guide File	Components	Connections
v100ebg352_64_66.ncd	150	134
v300bg432_64_66.ncd	214	134
v300ebg432_64_66.ncd	214	134
v1000fg680_64_66.ncd	214	134
v1000efg680_64_66.ncd	214	134

Table 2-2 Guide File Information

Guide File	Components	Connections
2v1000fg456_64_66.ncd	214	220

If a guide file is required, use the appropriate guide file from the guide/ directory when processing designs with the Xilinx implementation tools.

Note: The example design relies on the presence of the default pcim_lc.*hdl* wrapper file in the xpci/ directory. If you change this file, you must also change the constraints and guide files used in the processing scripts.

Device Initialization

Immediately after FPGA configuration, both the PCI interface and the user application will be initialized by the startup mechanism present in all Virtex devices.

During normal operation, the assertion of RST# on the PCI Bus will re-initialize the PCI interface and three-state all PCI Bus signals. This behavior is fully compliant with the *PCI Local Bus Specification*.

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

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

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

Bus Width Detection

A PCI interface which provides a 64-bit datapath needs to know if it is connected to a 64-bit bus or a 32-bit bus. The **SL0T64** signal is an input to the PCI64 interface for this purpose.

The PCI Bus specification provides a mechanism for PCI agents to determine the width of the bus by sampling the state of the REQ64# signal at the rising edge of RST#.

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

In designs for open systems, the bus width is not known in advance. In this case, include a separate latch or flip-flop, external to the FPGA, to sample REQ64#. Figure 2-1 shows how this may be accomplished.

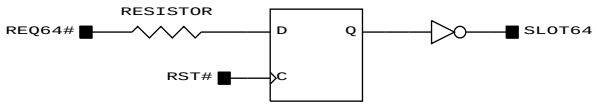

Figure 2-1 Sample SLOT64 Generation

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

An alternate method is to push the entire circuit into the FPGA and use the **REQ64Q_N** and **RST** signals provided to the user application. This method requires that the FPGA be fully configured by the rising edge of RST#.

When **SLOT64** is deasserted, the PCI64 interface automatically three-states the 64-bit extension signals. In this situation, the 64-bit extension signals are undriven, which may result in additional power consumption by the input buffers.

If the additional power consumption is of concern due to design requirements, consider changing the “Disabled Extension Drive” option in the HDL configuration file. This option, when enabled, forces the PCI64 interface to actively drive the extension signals when **SLOT64** is deasserted.

Note: Although this option may reduce power consumption, it creates an electrically unsavory situation. When a 64-bit card is

installed in a 32-bit slot, the 64-bit bus extension is floating in free space and unprotected from roaming screwdrivers.

Datapath Output Clock Enable

The LogiCORE PCI interface targeting Virtex devices uses one of two methods for generating and distributing the datapath output clock enable signal.

- Specialized device resources, the PCIIOBs, PCILOGIC, and PCI_CE.
- Generic device resources, IOBs, LUTs, and general purpose routing.

The specialized device resources offer higher and more predictable performance, and are available in all Virtex devices with the exception of Virtex-II. However, they constrain the LogiCORE PCI interface to the left or right sides of the FPGA device, and limit the number of LogiCORE PCI interface instances to two. The generic device resources, while lower performance and less predictable, offer greater flexibility.

To summarize, it is either appropriate or necessary to use the generic device resources in the following cases:

1. The target device is Virtex-II.
2. More than two instances of the interface are required.
3. The interface cannot be located on the left or right side.

To disable the use of specialized device resources, edit the HDL configuration file and set the `CFG[251]` bit to logic one.

Note: You must set `CFG[251]` to logic one if you target a Virtex-II device. Use of this option with other devices is not supported, and is incompatible with the provided constraint and guide files.

Voltage Supply Requirements

The LogiCORE PCI interface targeting Virtex devices uses one of three Virtex I/O buffer types depending on the signaling environment (this selection is made via the wrapper file).

Note: Virtex-E and Virtex-II devices are not 5.0 volt tolerant. Do not use Virtex-E or Virtex-II devices in a 5.0 volt signaling environment.

Wrapper files for the 5.0 volt signaling environment use the PCI33_5 I/O buffers available on Virtex and Spartan-II devices. This requires V_{CCO} to be set at 3.3 volts, and does not require a V_{REF} supply. Observe the relevant specifications in the *Virtex 2.5V Field Programmable Gate Array Data Sheet* or the *Spartan-II 2.5V Field Programmable Gate Array Data Sheet*. No other restrictions apply.

Wrapper files for the 3.3 volt signaling environment use either the PCI33_3 or the PCI66_3 I/O buffers available on Virtex, Spartan-II, Virtex-E, and Virtex-II devices. These require V_{CCO} to be set at 3.3 volts, and do not require a V_{REF} supply. Observe the relevant specifications in the *Virtex 2.5V Field Programmable Gate Array Data Sheet*, the *Spartan-II 2.5V Field Programmable Gate Array Data Sheet*, the *Virtex-E 1.8V Field Programmable Gate Array Data Sheet*, or the *Virtex-II 1.8V Field Programmable Gate Array Data Sheet*. For 3.3 volt signaling in Virtex-E and Virtex-II devices, no other restrictions apply. However, additional restrictions do apply for 3.3 volt signaling in Virtex and Spartan-II devices, as described below.

For 3.3 volt signaling in Virtex and Spartan-II devices, the data sheets indicate that the V_{IL} and V_{IH} parameters for the input buffers are a function of V_{CCINT}, which is a 2.5 volt supply. In the *PCI Local Bus Specification*, the specifications for the 3.3 volt signaling environment state V_{IL} and V_{IH} as a function of V_{CC}. This may be considered the 3.3 volt system supply.

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

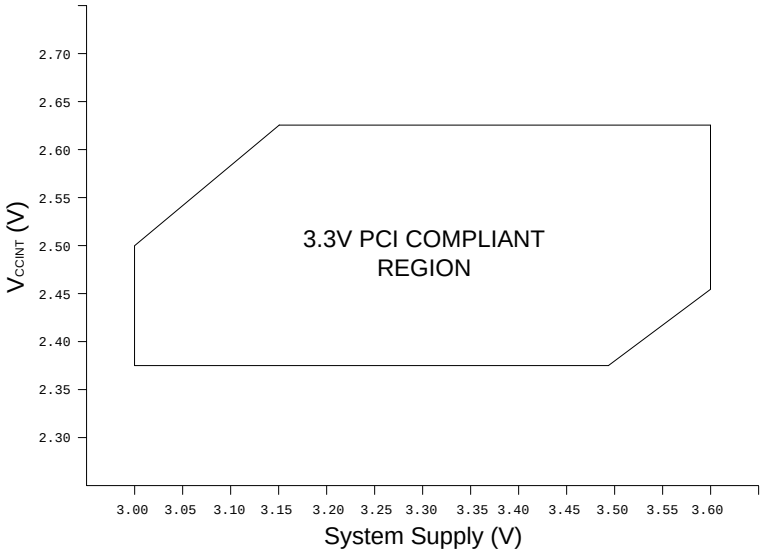

Figure 2-2 Relationship For 3.3V Input Buffer Compliance

Figure 2-2 is provided to show the small range of supply voltage values where V_{IL} or V_{IH} are technically non-compliant. Note that this may occur with any PCI device if the input buffer supply voltage is different from the supply voltage of the driving device. For best results, use a high precision (1% or better) voltage regulator to generate V_{CCINT}.

Generating Bitstreams

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

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

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

Refer to the release notes and the user constraints file for additional information about the use and implications of this required option.

This option is used to introduce additional delay on a global clock net. For 66 MHz designs, it is important to note that this additional delay is observable on the **CLK** output of the PCI interface, which is supplied to the user application. Timing constraints for the user application must be generated with this in mind.

Know the Degree of Difficulty

PCI interfaces are challenging to implement in any technology. Table 2-3, "Degree Of Difficulty" indicates the degree of difficult in implementing various types of PCI designs in Virtex devices.

Table 2-3 Degree Of Difficulty

Clock Frequency	Difficulty
33 MHz	Easy
50 MHz	Moderate
66 MHz	Difficult

The degree of difficulty is sharply influenced by the nature of the user application. For 66 MHz designs, extensive use of pipelining, logic mapping, placement constraints, and logic duplication may be required. Carefully consider all timing specifications in a 66 MHz design to ensure that the user application is not over-constrained.

Chapter 3

Functional Simulation

This chapter describes the use of supported tools for functional simulation. These tools are:

- Cadence Verilog-XL v.3.0
- Synopsys VSS v1999.10
- Model Technology ModelSim PE v5.2e

The example design includes a simple testbench. This testbench may be leveraged to create larger testbenches. Optionally, users may purchase third party testbenches or create their own.

The example design is a simple user application which is intended for use as a training vehicle and design flow test. The LogiCORE PCI32 interface ships with the design “ping32”, while the LogiCORE PCI64 interface ships with the design “ping64”. The examples in this chapter refer to “ping64”. If you are using the LogiCORE PCI32 interface, simply substitute “ping32” for “ping64”.

By default, the example design targets a 66 MHz implementation and simulates with a 15 ns clock period. If you are targeting a 33 MHz implementation, adjust the simulation clock period to 30 ns. The clock period is defined in the stimulus source file.

Cadence Verilog-XL

Before attempting functional simulation, ensure that the Verilog-XL environment is properly configured for use.

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

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

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

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

Modify the library search path by changing `<Xilinx Install Path>` to match the Xilinx installation directory. Save the file.

Most of the files listed are related to the example design and its testbench. For other testbenches, the following subset must be used for proper simulation of the PCI interface:

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

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

3. To run the Verilog-XL simulation:

   `verilog -f ping_tb.f`

   Verilog-XL processes the simulation files and exits. The testbench prints status messages to the console. After the simulation completes, view the `verilog.log` file to check for errors.

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

`signalscan`

A sample Signalscan "do file", `signalscan.do` is provided for convenience.

Synopsys VSS

Before attempting functional simulation, ensure that the VSS environment is properly configured for use. Furthermore, the `$XILINX` environment variable must be set to match the Xilinx installation directory.

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

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

2. Run the `analyze_ping` script. This script will analyze the required simprim and unisim libraries, as well as the design files.

   With the exception of the libraries, most of the files analyzed by the script are related to the example design and its testbench. For other testbenches, the following files must be used for proper simulation of the PCI interface:

   ```
   $XILINX/vhdl/src/simprims/simprim_Vpackage.vhd
   $XILINX/vhdl/src/simprims/simprim_Vcomponents.vhd
   $XILINX/vhdl/src/simprims/simprim_VITAL.vhd
   $XILINX/vhdl/src/unisims/unisim_VPKG.vhd
   $XILINX/vhdl/src/unisims/unisim_VCOMP.vhd
   $XILINX/vhdl/src/unisims/unisim_VITAL.vhd
   ../../src/xpci/pci_lc_i.vhd
   ../../src/xpci/pcim_lc.vhd
   ```

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

3. To run the VSS simulation:

   `run_ping`

   VSS processes the simulation files and halts when the testbench has finished. The testbench prints status messages to the console. After the simulation completes, view the console output to check for errors. The `ping.traces` file contains a list of signals which are recorded to the waveform database.

The Waves browser may be used to view the simulation results. A sample Waves setup "restore file", `ping.wfc` is provided for convenience.

Model Technology ModelSim

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

Verilog

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

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

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

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

   Modify the library search path by changing `<Xilinx Install Path>` to match the Xilinx installation directory. Save the file.

   Most of the files listed are related to the example design and its testbench. For other testbenches, the following subset must be used for proper simulation of the PCI interface:

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

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

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

   <Install Path>/verilog/example/func_sim

4. Create the work directory:

   vlib work

5. To run the simulation:

   do modelsim.do

VHDL

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

   cd <Install Path>/vhdl/example/func_sim

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

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

   Most of the files listed are related to the example design and its testbench. For other testbenches, the following subset must be used for proper simulation of the PCI interface:

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

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

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

<Install Path>/vhdl/example/func_sim

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

   vlib simprim

   vcom -work simprim <Xilinx Install Path>/vhdl/src/simprims/simprim_Vpackage.vhd

   vcom -work simprim <Xilinx Install Path>/vhdl/src/simprims/simprim_Vcomponents.vhd

   vcom -work simprim <Xilinx Install Path>/vhdl/src/simprims/simprim_VITAL.vhd

   vlib unisim

   vcom -work unisim <Xilinx Install Path>/vhdl/src/unisims/unisim_VPKG.vhd

   vcom -work unisim <Xilinx Install Path>/vhdl/src/unisims/unisim_VCOMP.vhd

   vcom -work unisim <Xilinx Install Path>/vhdl/src/unisims/unisim_VITAL.vhd

5. Create the work directory:

   vlib work

6. To run the simulation:

   do modelsim.do

Chapter 4

Synthesis

This chapter describes the use of supported tools for synthesis. These tools are:

- Synopsys FPGA Compiler v1999.10
- Synopsys FPGA Express v3.4
- Synplicity Synplify v5.3.1
- Exemplar LeonardoSpectrum v1999.1i

The example design includes a simple user application which is intended for use as a training vehicle and design flow test. The LogiCORE PCI32 interface ships with the design “ping32”, while the LogiCORE PCI64 interface ships with the design “ping64”. The examples and screen shots in this chapter refer to “ping64”. If you are using the LogiCORE PCI32 interface, simply substitute “ping32” for “ping64”.

By default, the example design targets a 66 MHz implementation. If you are targeting a 33 MHz implementation, it is advisable to adjust the timing constraints and the target device during synthesis, although this is not absolutely necessary.

Synopsys FPGA Compiler

Before attempting to synthesize a design, ensure that the Synopsys FPGA Compiler environment is properly configured for use.

1. To begin, move into the synthesis directory:

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

   The synthesis directory contains the `WORK` directory, the compile script `synopsys.dc` and the Design Compiler setup file `.synopsys_dc.setup`.

2. Synthesize the design by running the `synopsys.dc` synthesis script file from the Design Analyzer, or by using `dc_shell`:

   `dc_shell -f synopsys.dc`

   Use of the Design Analyzer is highly recommended, as it will stop if an error occurs during the synthesis process. In contrast, `dc_shell` will not stop if an error occurs.

The end result of the synthesis step is a set of Synopsys EDIF files which are fed into the Xilinx implementation tools during the implementation step.

In practice, the provided `synopsys.dc` file must be modified to accommodate other designs. To provide insight into the synthesis script, the major steps are discussed in the following paragraphs.

1. The entire design is analyzed and elaborated.

```
analyze -format hdl ../../src/xpci/pci_lc_i.hdl
analyze -format hdl ../../src/xpci/pcim_lc.hdl
analyze -format hdl ../source/cfg_ping.hdl
analyze -format hdl ../source/ping.hdl
analyze -format hdl ../source/pcim_top.hdl
elaborate pcim_top > output.ela
```

2. The user application is selected as the current design and constraints are applied to it. Once this is done, the user application is compiled.

```
current_design "ping64"
create_clock "CLK" -period 15
set_output_delay 8 -clock "CLK" all_outputs()
set_max_delay 8 -from {"S_DATA", "S_SRC_EN",\
                       "S_DATA_VLD", "M_DATA",\
                       "M_SRC_EN", "M_DATA_VLD",\
                       "M_ADDR_N"} -to { "ADIO*" }
compile > output.com
```

3. Once the user application is compiled, the current design is set to the top level and the entire design is compiled. The FPGA Compiler is instructed not to touch several modules, including:

   - The PCI interface, a “black box”
   - Components instantiated from the Xilinx Unified Library
   - Modules which have already been compiled

Note that the FPGA Compiler allows selective I/O insertion through use of the `set_port_is_pad` command. Ports may be created using this command instead of directly instantiating I/O structures. Do not insert pads on PCI Bus signals; these are already instantiated in the `pcim_lc` module.

```
current_design "pcim_lc"
set_dont_touch { "XPCI_*", "PCI_LC" }
current_design "pcim_top"
set_dont_touch "ping64"
set_port_is_pad { "PING_REQUEST64",\
                  "PING_REQUEST32", "PING_DONE" }
insert_pads
create_clock "PCLK" -period 15
compile
```

4. After synthesis is complete, the synthesized netlist is written out. In this process, a “black box” representation of the PCI interface is created.

```
remove_attribute "ping64" dont_touch
set_attribute find(design,"*") "xnfout_use_blknames"\
       -type boolean false
set_attribute "pcim_top" "part" -type string part
remove_design "PCI_LC_I"
edifout_design_name = "pcim_top"
write -format db -hierarchy \
       -output "pcim_top_syn.db"
write -format edif -hierarchy \
       -output "pcim_top.sedif"
exit
```

FPGA Compiler may issue a number of warnings about instantiated I/O cells. These warnings are expected. Furthermore, the tool may issue warnings about attributes used for other synthesis tools.

Synopsys FPGA Express

Before attempting to synthesize a design, ensure that the Synopsys FPGA Express environment is properly configured for use.

Verilog

1. Start FPGA Express and create a new project. This may be done from the `File` menu or from the toolbar.

Figure 4-1 Create a New Project

2. A new project dialog box will appear. Create the new project (for example, "`flowtest`") in the appropriate synthesis directory:

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

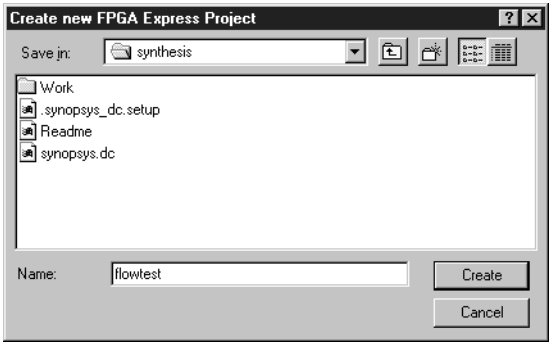

Figure 4-2 New Project Dialog Box

3. After creating the new project, FPGA Express will immediately prompt you to add source files to the project. This process takes two steps.

   The first set of files (used by all PCI designs) is located in:

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

Use the dialog box to move to this directory and select the two files (`pci_lc_i.v` and `pcim_lc.v`) as shown in Figure 4-3 (use CTRL-click to select the files in one step). Then click on the `Open` button. FPGA Express will add these files to the project and return to the main display.

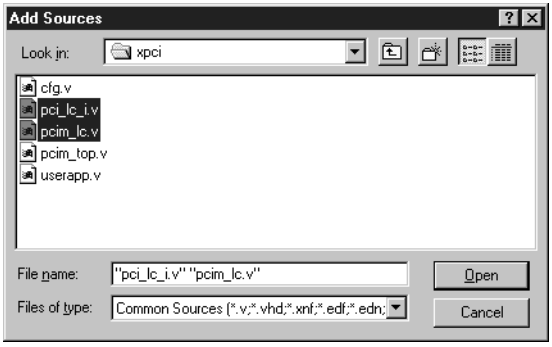

Figure 4-3 Add Sources Dialog Box (LogiCORE Files)

At this point, it is necessary to add additional files required by the user application. Right click on the `flowtest` design source and select `Add sources in WORK...` from the pop-up menu.

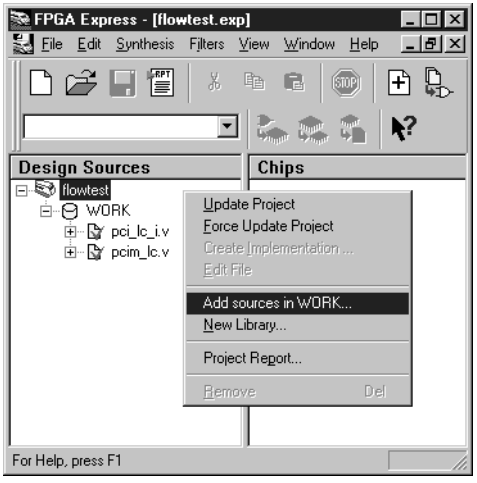

Figure 4-4 Add Additional Files

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

<Install Path>/verilog/example/source

Use the dialog box to move to this directory and select the three files (`cfg_ping.v`, `pcim_top.v`, and `ping.v`) as shown in Figure 4-5 (use CTRL-click to select the files in one step). Then click on the `Open` button. FPGA Express will add these files to the project and return to the main display.

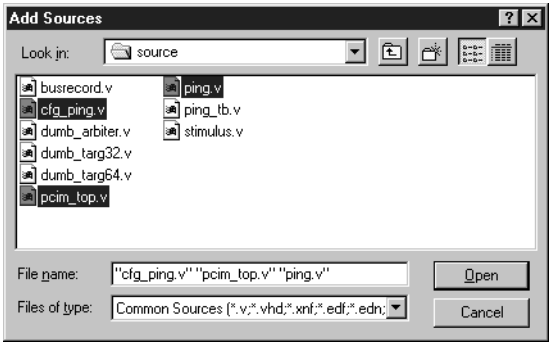

Figure 4-5 Add Sources Dialog Box (User Files)

4. Select the top level module of the design, `pcim_top`, from the drop-down selection box on the toolbar.

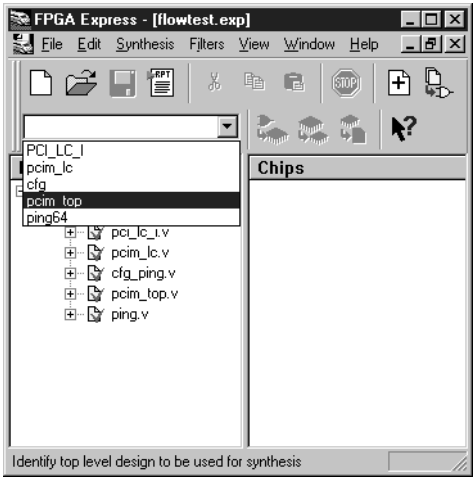

Figure 4-6 Select Top Level Module

FPGA Express will display another dialog box to gather more information for the implementation process.

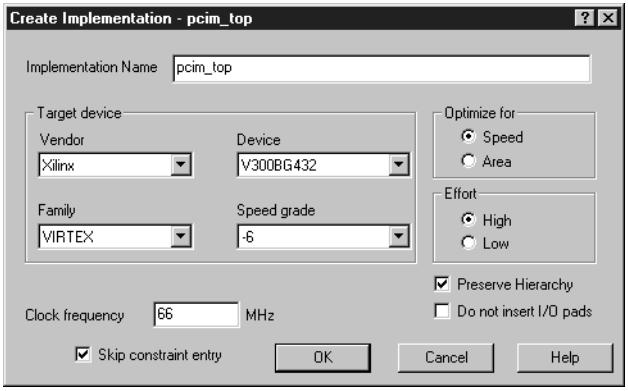

Figure 4-7 Create Implementation Dialog Box

Set the “Vendor”, “Family”, “Device”, and “Speed Grade” options to reflect the targeted device. Additionally, the “Preserve Hierarchy” option *must* be checked. When you have set the correct options, click on the Ok button.

FPGA Express will elaborate and optimize the implementation. When it is complete, the optimized "Chip" will appear in the main display. FPGA Express may issue a number of warnings about instantiated I/O cells and attributes used for other synthesis tools. These warnings are expected.

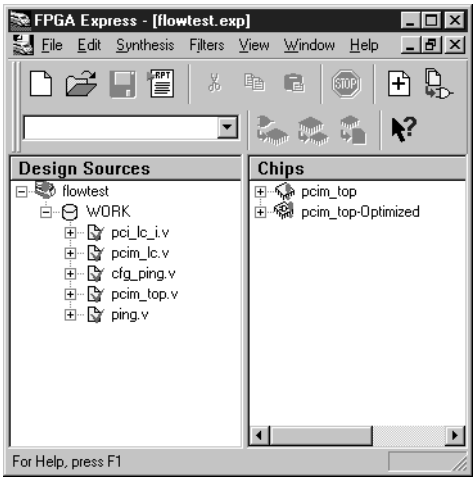

Figure 4-8 Optimization Result

5. The final step is to export a netlist for use by the Xilinx implementation tools. Right click on the optimized implementation in the "Chips" window and select `Export Netlist` from the pop-up menu.

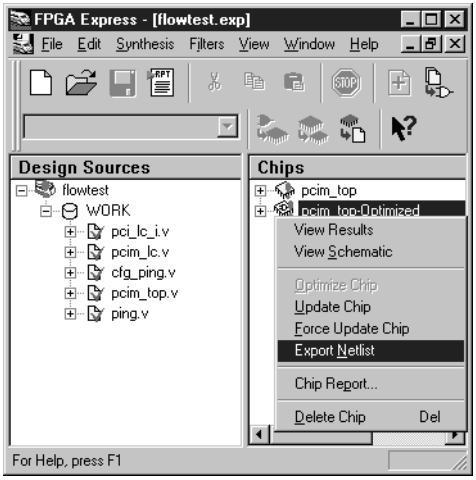

Figure 4-9 Export Netlist

FPGA Express will open a dialog box to gather additional information before it exports a netlist.

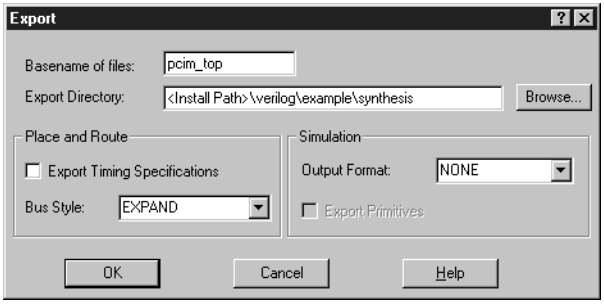

Figure 4-10 Export Netlist Dialog Box

Modify the export directory so that the output files are written to:

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

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

VHDL

1. Start FPGA Express and create a new project. This may be done from the `File` menu or from the toolbar.

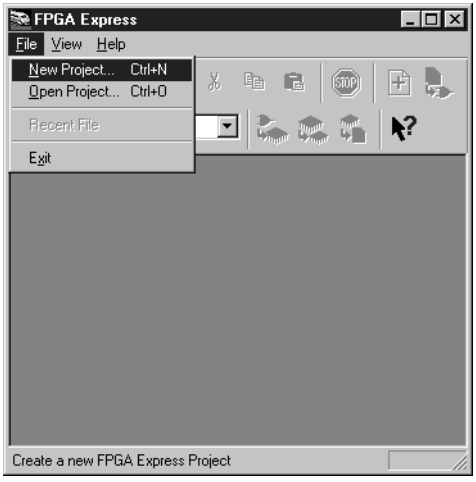

Figure 4-11 Create a New Project

2. A new project dialog box will appear. Create the new project (for example, "`flowtest`") in the appropriate synthesis directory:

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

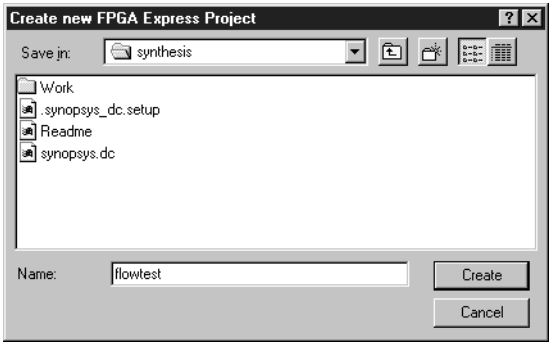

Figure 4-12 New Project Dialog Box

3. After creating the new project, FPGA Express will immediately prompt you to add source files to the project. This process takes two steps.

   The first set of files (used by all PCI designs) is located in:

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

   Use the dialog box to move to this directory and select the two files (`pci_lc_i.vhd` and `pcim_lc.vhd`) as shown in Figure 4-13 (use CTRL-click to select the files in one step). Then click on the `Open` button. FPGA Express will add these files to the project and return to the main display.

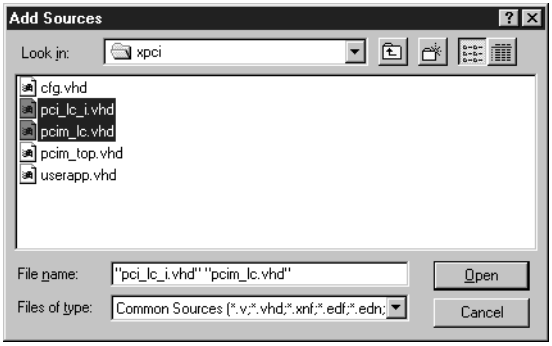

Figure 4-13 Add Sources Dialog Box (LogiCORE Files)

At this point, it is necessary to add additional files required by the user application. Right click on the `flowtest` design source and select `Add sources in WORK...` from the pop-up menu.

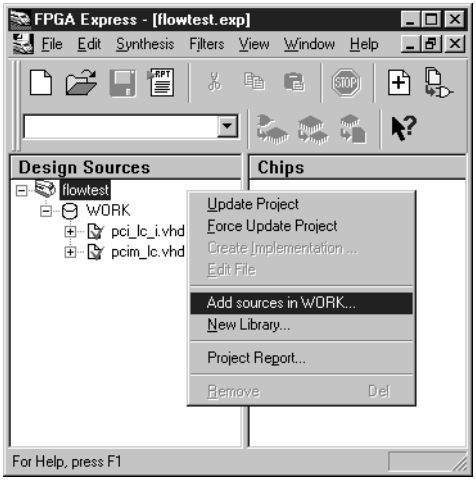

Figure 4-14 Add Additional Files

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

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

Use the dialog box to move to this directory and select the three files (`cfg_ping.vhd`, `pcim_top.vhd`, and `ping.vhd`) as shown in Figure 4-15 (use CTRL-click to select the files in one step). Then click on the `Open` button. FPGA Express will add these files to the project and return to the main display.

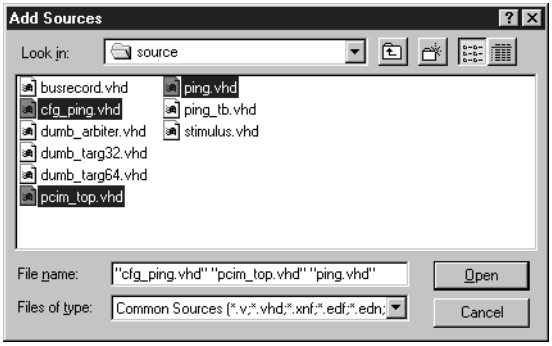

Figure 4-15 Add Sources Dialog Box (User Files)

4. Select the top level module of the design, `pcim_top`, from the drop-down selection box on the toolbar.

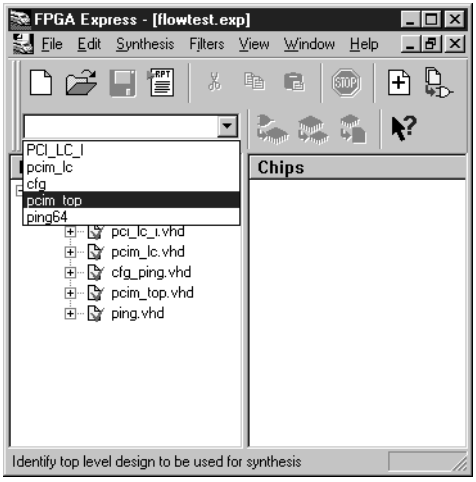

Figure 4-16 Select Top Level Module

FPGA Express will display another dialog box to gather more information for the implementation process.

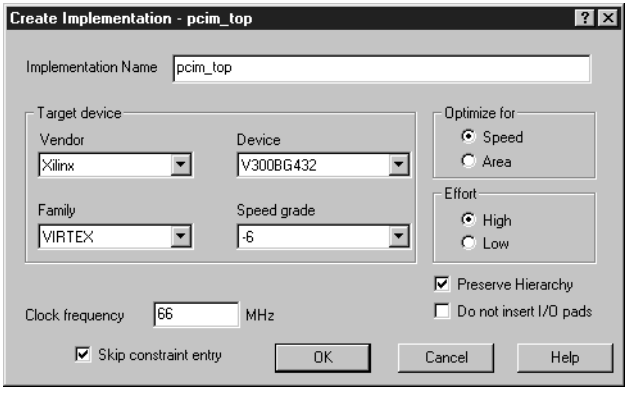

Figure 4-17 Create Implementation Dialog Box

Set the “Vendor”, “Family”, “Device”, and “Speed Grade” options to reflect the targeted device. Additionally, the “Preserve

Hierarchy" option *must* be checked. When you have set the correct options, click on the `Ok` button.

FPGA Express will elaborate and optimize the implementation. When it is complete, the optimized "Chip" will appear in the main display. FPGA Express may issue a number of warnings about instantiated I/O cells and attributes used for other synthesis tools. These warnings are expected.

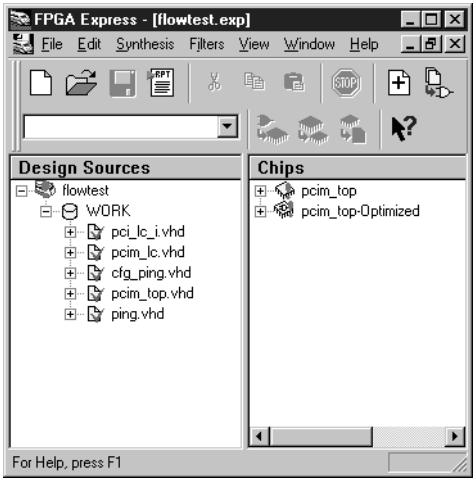

Figure 4-18 Optimization Result

5. The final step is to export a netlist for use by the Xilinx implementation tools. Right click on the optimized implementation in the "Chips" window and select `Export Netlist` from the pop-up menu.

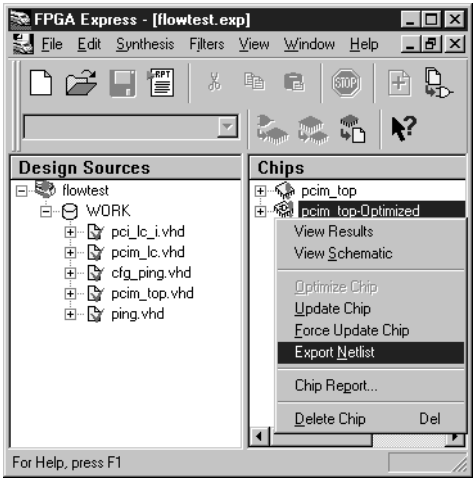

Figure 4-19 Export Netlist

FPGA Express will open a dialog box to gather additional information before it exports a netlist.

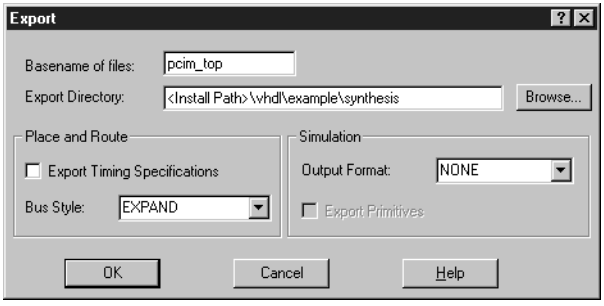

Figure 4-20 Export Netlist Dialog Box

Modify the export directory so that the output files are written to:

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

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

Synplicity Synplify

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

Verilog

Synthesis with Synplify is a three step process. First, the user application (in this example, the Ping design) is synthesized separately to create an EDIF netlist. Then, the LogiCORE PCI interface is synthesized to create an EDIF wrapper file. Finally, the user application and the LogiCORE PCI interface are declared as black boxes in the top level design, and are synthesized to create the top level EDIF file. All three steps are required.

Begin by synthesizing the user application:

1. Start Synplify and create a new project named "`ping.prj`" in the appropriate synthesis directory:

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

2. Add the Synplify Virtex library and the user application source file(s) to the project. These files are:

   `<Synplify Install Path>/lib/xilinx/virtex.v`

   `<Install Path>/verilog/example/source/ping.v`

3. Set the global frequency as required in the project window, and disable the generation of vendor constraint files using the `Options -> Write Vendor Constraint File` menu option. Then, set the synthesis result to:

   `<Install Path>/verilog/example/synthesis/ping64.edf`

At this point, the project window should appear similar to that shown in Figure 4-21. The exact path names in the source window will vary depending on your installation.

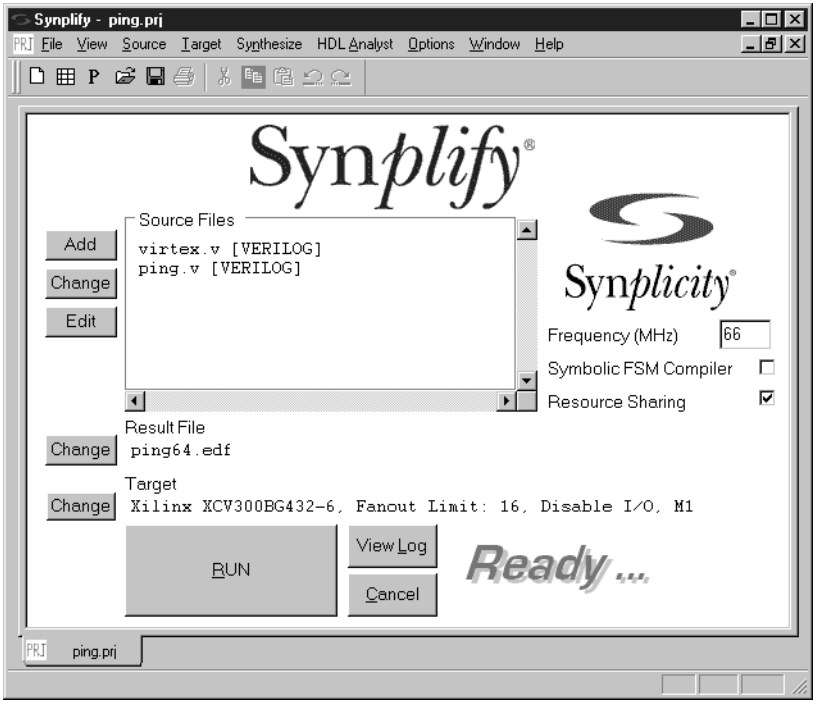

Figure 4-21 User Application Project Window

4. Change the synthesis target by clicking on the button in the project window. Set the “Technology”, “Part”, “Package”, and “Speed Grade” options to reflect the targeted device. Additionally, the other options must be set as shown in Figure 4-22.

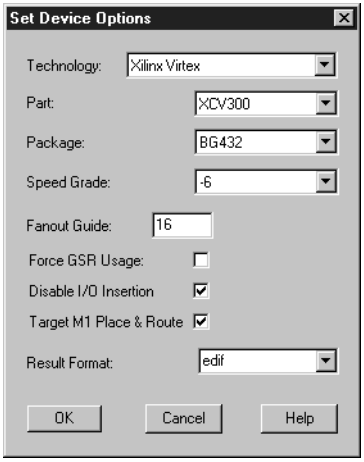

Figure 4-22 User Application Device Options

5. Synthesize the project by clicking on the RUN button in the project window. The tool should finish, and may indicate warnings about unused signals. These warnings are expected.

Next, synthesize the LogiCORE PCI interface:

1. Start Synplify if it is not already running and create a new project named "pcim_lc.prj" in the appropriate synthesis directory:

   <Install Path>/verilog/example/synthesis

2. Add the Synplify Virtex library and the LogiCORE PCI interface source files to the project. These files are:

   <Synplify Install Path>/lib/xilinx/virtex.v

   <Install Path>/verilog/src/xpci/pci_lc_i.v

   <Install Path>/verilog/src/xpci/pcim_lc.v

3. Set the global frequency as required in the project window, and disable the generation of vendor constraint files using the Options -> Write Vendor Constraint File menu option. Then, set the synthesis result to:

   <Install Path>/verilog/example/synthesis/pcim_lc.edf

At this point, the project window should appear similar to that shown in Figure 4-23. The exact path names in the source window will vary depending on your installation.

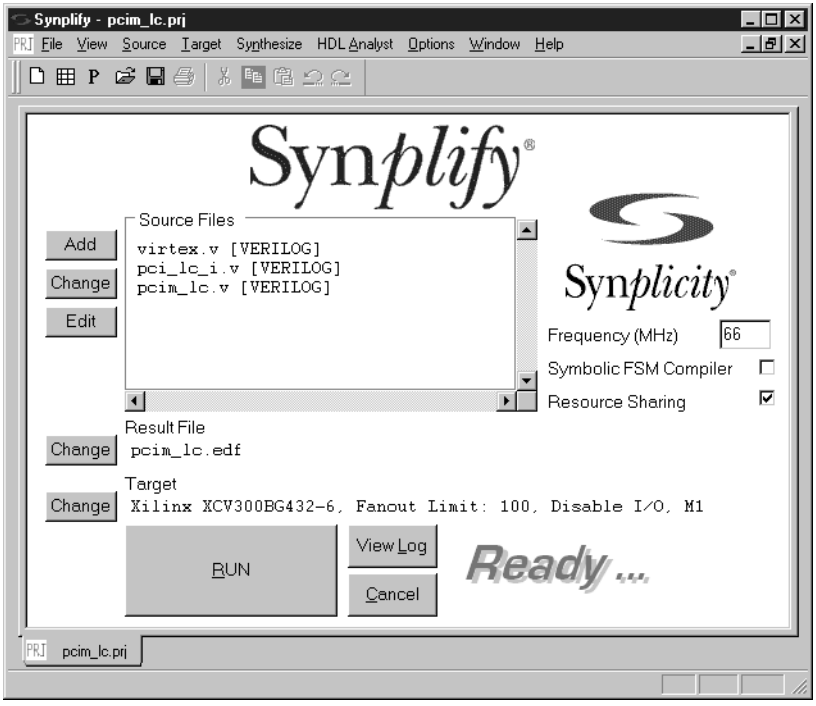

Figure 4-23 LogiCORE PCI Interface Project Window

4. Change the synthesis target by clicking on the button in the project window. Set the “Technology”, “Part”, “Package”, and “Speed Grade” options to reflect the targeted device. Additionally, the other options must be set as shown in Figure 4-24.

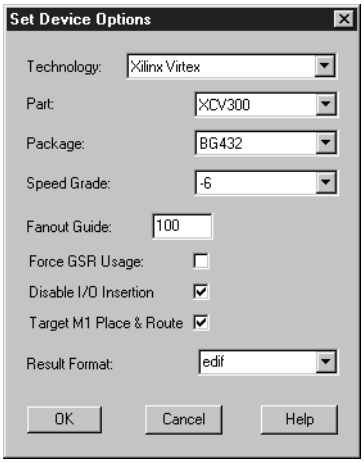

Figure 4-24 LogiCORE PCI Interface Device Options

5. Synthesize the project by clicking on the RUN button in the project window. The tool should finish with no warnings.

The final step is to synthesize the top level design:

1. Create a black box file for the user application. A suitable black box file is provided with the example design. Use this file as a reference when creating black box files for other designs.

   In general, a black box file may be created for any user application by copying the original source file and making the following modifications:

   Before:

```
module my_userapp (. . .);
// synthesis syn_noclockbuf = 1
// synthesis syn_edif_bit_format = "%u<%i>"
// synthesis syn_edif_scalar_format = "%u"
```

   After:

```
module my_userapp (. . .);
// synthesis syn_edif_bit_format = "%u<%i>"
// synthesis syn_edif_scalar_format = "%u"
// synthesis black_box
```

   Then, remove all lines after the port definitions up to the final "endmodule" statement.

2. Start Synplify if it is not already running and create a new project named "`pcim_top.prj`" in the appropriate synthesis directory:

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

3. Add the Synplify Virtex library and the required source files to the project. These files are:

   `<Synplify Install Path>/lib/xilinx/virtex.v`

   `<Install Path>/verilog/example/source/ping_bb.v`

   `<Install Path>/verilog/example/source/cfg_ping.v`

   `<Install Path>/verilog/src/xpci/pcim_lc_bb.v`

   `<Install Path>/verilog/example/source/pcim_top.v`

   Be sure to add the source files in the listed order to maintain their hierarchical relationship. If you need to reorder the files after adding them, you may do this in the Source Files list box.

4. Set the global frequency as required in the project window, and disable the generation of vendor constraint files using the `Options -> Write Vendor Constraint File` menu option. Then, set the synthesis result to:

   `<Install Path>/verilog/example/synthesis/pcim_top.edf`

At this point, the project window should appear similar to that shown in Figure 4-25. The exact path names in the source window will vary depending on your installation.

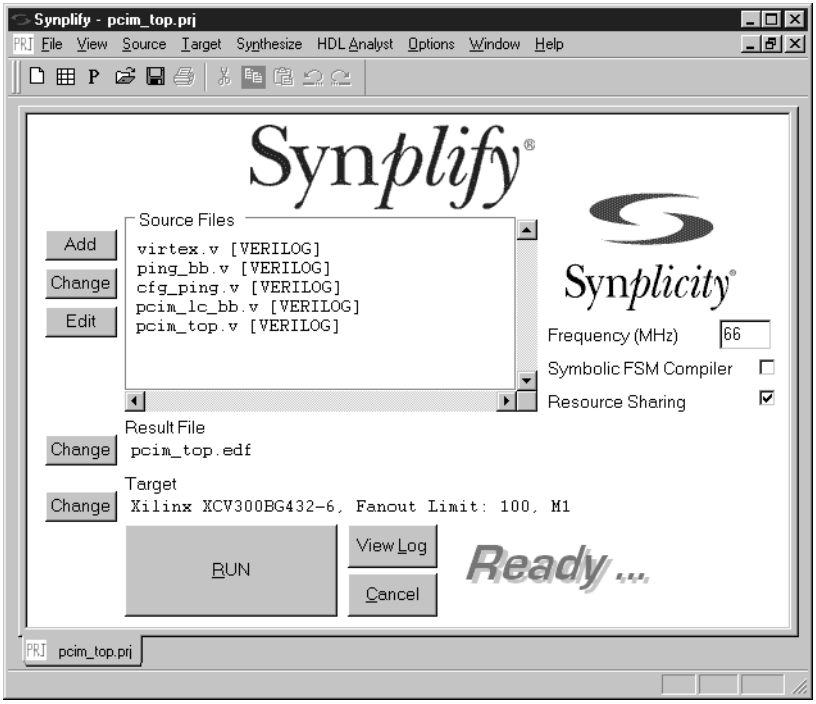

Figure 4-25 Top Level Project Window

5. Change the synthesis target by clicking on the button in the project window. Set the “Technology”, “Part”, “Package”, and “Speed Grade” options to reflect the targeted device. Additionally, the other options must be set as shown in Figure 4-26. Do not disable I/O insertion.

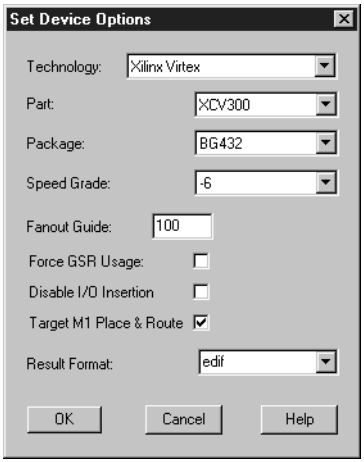

Figure 4-26 Top Level Device Options

6. Synthesize the project by clicking on the RUN button in the project window. The tool should finish with no warnings.

VHDL

Synthesis with Synplify is a three step process. First, the user application (in this example, the Ping design) is synthesized separately to create an EDIF netlist. Then, the LogiCORE PCI interface is synthesized to create an EDIF wrapper file. Finally, the user application and the LogiCORE PCI interface are declared as black boxes in the top level design, and are synthesized to create the top level EDIF file. All three steps are required.

Begin by synthesizing the user application:

1. Start Synplify and create a new project named "`ping.prj`" in the appropriate synthesis directory:

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

2. Add the Synplify Virtex library and the user application source file(s) to the project. These files are:

   `<Synplify Install Path>/lib/xilinx/virtex.vhd`

   `<Install Path>/vhdl/example/source/ping.vhd`

3. Set the global frequency as required in the project window, and disable the generation of vendor constraint files using the `Options -> Write Vendor Constraint File` menu option. Then, set the synthesis result to:

   `<Install Path>/vhdl/example/synthesis/ping64.edf`

At this point, the project window should appear similar to that shown in Figure 4-27. The exact path names in the source window will vary depending on your installation.

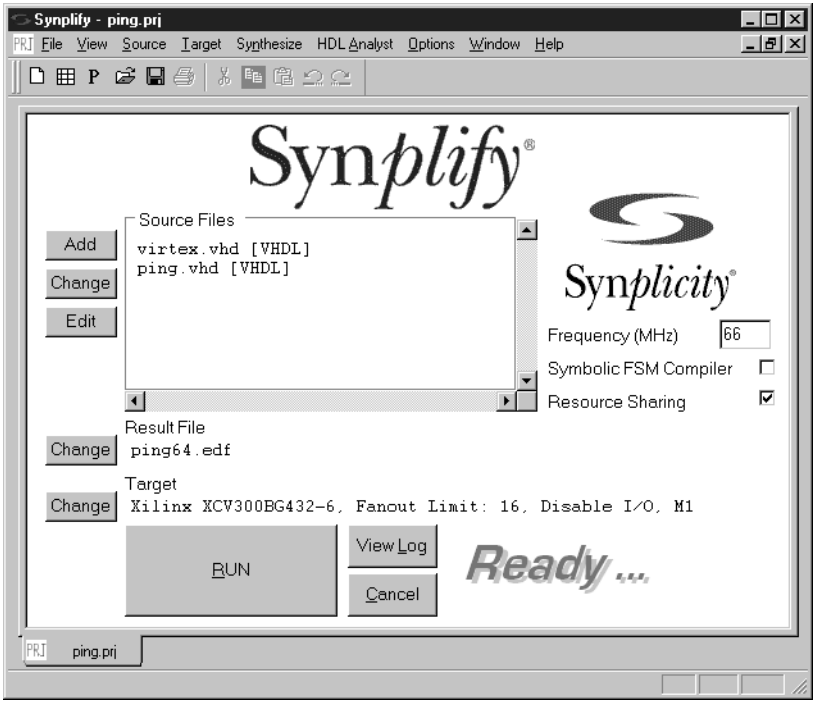

Figure 4-27 User Application Project Window

4. Change the synthesis target by clicking on the button in the project window. Set the “Technology”, “Part”, “Package”, and “Speed Grade” options to reflect the targeted device. Additionally, the other options must be set as shown in Figure 4-28.

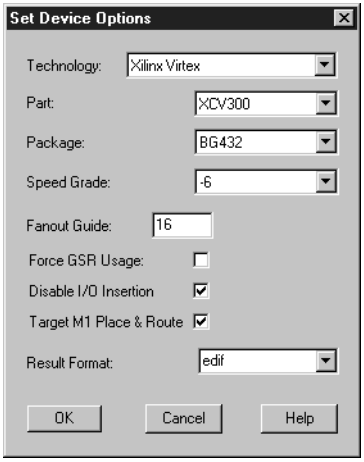

Figure 4-28 User Application Device Options

5. Synthesize the project by clicking on the RUN button in the project window. The tool should finish, and may indicate warnings about unused signals. These warnings are expected.

Next, synthesize the LogiCORE PCI interface:

1. Start Synplify if it is not already running and create a new project named "pcim_lc.prj" in the appropriate synthesis directory:

   <Install Path>/vhdl/example/synthesis

2. Add the Synplify Virtex library and the LogiCORE PCI interface source files to the project. These files are:

   <Synplify Install Path>/lib/xilinx/virtex.vhd

   <Install Path>/vhdl/src/xpci/pci_lc_i.vhd

   <Install Path>/vhdl/src/xpci/pcim_lc.vhd

3. Set the global frequency as required in the project window, and disable the generation of vendor constraint files using the Options -> Write Vendor Constraint File menu option. Then, set the synthesis result to:

   <Install Path>/vhdl/example/synthesis/pcim_lc.edf

At this point, the project window should appear similar to that shown in Figure 4-29. The exact path names in the source window will vary depending on your installation.

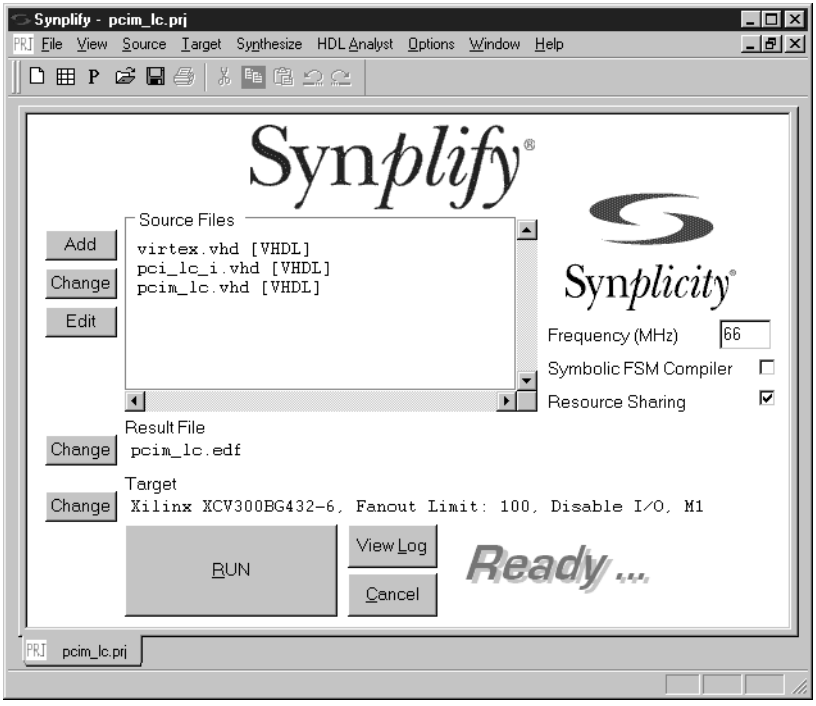

Figure 4-29 LogiCORE PCI Interface Project Window

4. Change the synthesis target by clicking on the button in the project window. Set the “Technology”, “Part”, “Package”, and “Speed Grade” options to reflect the targeted device. Additionally, the other options must be set as shown in Figure 4-30.

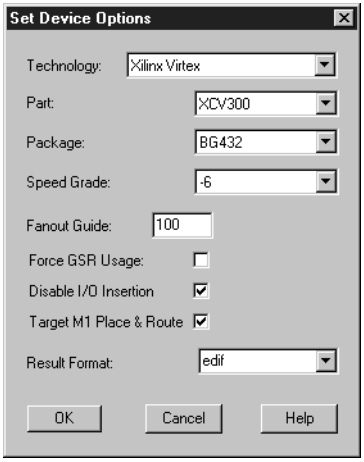

Figure 4-30 LogiCORE PCI Interface Device Options

5. Synthesize the project by clicking on the RUN button in the project window. The tool should finish with no warnings.

The final step is to synthesize the top level design:

1. Create a black box file for the user application. A suitable black box file is provided with the example design. Use this file as a reference when creating black box files for other designs.

   In general, a black box file may be created for any user application by copying the original source file and making the following modifications:

   Before:

```
architecture rtl of my_userapp is
attribute syn_noclockbuf : boolean;
attribute syn_edif_bit_format : string;
attribute syn_edif_scalar_format : string;
attribute syn_noclockbuf of rtl :
          architecture is true;
attribute syn_edif_bit_format of rtl :
          architure is "%u<%i>";
attribute syn_edif_scalar_format or rtl :
          architecture is "%u";
```

   After:

```
architecture rtl of my_userapp is
attribute black_box : boolean;
```

```
attribute syn_edif_bit_format : string;
attribute syn_edif_scalar_format : string;
attribute black_box of rtl :
          architecture is true;
attribute syn_edif_bit_format of rtl :
          architure is "%u<%i>";
attribute syn_edif_scalar_format or rtl :
          architecture is "%u";
```

Then, remove all lines after the attributes, leaving only the "`begin`" and "`end rtl;`" statements.

2. Start Synplify if it is not already running and create a new project named "`pcim_top.prj`" in the appropriate synthesis directory:

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

3. Add the Synplify Virtex library and the required source files to the project. These files are:

   `<Synplify Install Path>/lib/xilinx/virtex.vhd`

   `<Install Path>/vhdl/example/source/ping_bb.vhd`

   `<Install Path>/vhdl/example/source/cfg_ping.vhd`

   `<Install Path>/vhdl/src/xpci/pcim_lc_bb.vhd`

   `<Install Path>/vhdl/example/source/pcim_top.vhd`

   Be sure to add the source files in the listed order to maintain their hierarchical relationship. If you need to reorder the files after adding them, you may do this in the Source Files list box.

4. Set the global frequency as required in the project window, and disable the generation of vendor constraint files using the `Options -> Write Vendor Constraint File` menu option. Then, set the synthesis result to:

   `<Install Path>/vhdl/example/synthesis/pcim_top.edf`

At this point, the project window should appear similar to that shown in Figure 4-31. The exact path names in the source window will vary depending on your installation.

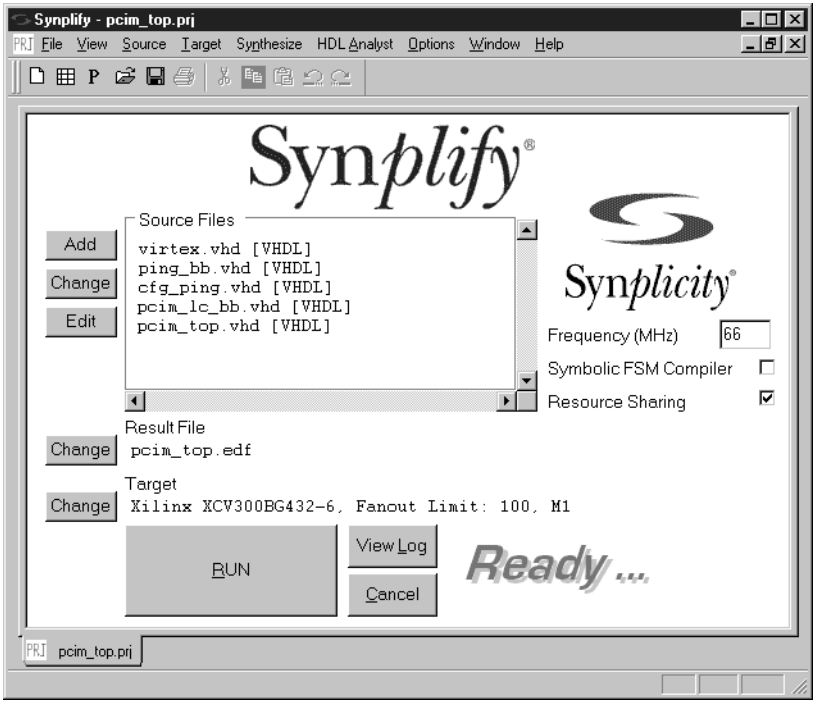

Figure 4-31 Top Level Project Window

5. Change the synthesis target by clicking on the button in the project window. Set the “Technology”, “Part”, “Package”, and “Speed Grade” options to reflect the targeted device. Additionally, the other options must be set as shown in Figure 4-32. Do not disable I/O insertion.

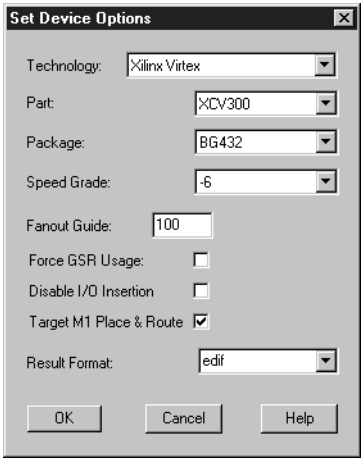

Figure 4-32 Top Level Device Options

6. Synthesize the project by clicking on the RUN button in the project window. The tool should finish with no warnings.

Exemplar LeonardoSpectrum

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

1. To begin, move into the synthesis directory:

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

   The synthesis directory contains a script, `leonardo.tcl`, for use with LeonardoSpectrum.

2. Edit the `leonardo.tcl` script to change the following line:

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

   Modify the path so that it points to the actual installation location, and then save the file. Invoke LeonardoSpectrum.

3. Synthesize the design by running the `leonardo.tcl` script. Note that if you run LeonardoSpectrum with the graphical user interface, the quicksetup form cannot be used to synthesize the design. Instead, use the `File -> Run Script` menu option.

The end result of the synthesis step is a set of EDIF files which are fed into the Xilinx implementation tools during the implementation step.

In practice, the provided `leonardo.tcl` file must be modified to accommodate other designs. To provide insight into the synthesis script, the major steps are discussed in the following paragraphs.

1. Various synthesis options are set through the use of environment variables. These must be present in the script, and should not be modified. The synthesis library is also loaded; this may be altered for different devices and speed grades.
2. The design is loaded by reading in the design files. At this point, the top level module is declared as the `present_design`. The script adds `nopad` attributes (with a value of `FALSE`) to all PCI Bus interface signals. The I/O structures for these ports are directly instantiated in the wrapper file.
3. The optimization step is done with the `-hierarchy preserve` and the `-chip` options. The `-hierarchy preserve` option prevents LeonardoSpectrum from dissolving the design hierarchy. The `-chip` option indicates that automatic I/O buffer insertion should be performed.
4. After synthesis is complete, the synthesized netlist is written out.

The tool may issue warnings about unused signals. These warnings are expected.

Chapter 5

Implementation

This chapter describes the use of supported tools for FPGA implementation. These tools are:

- Xilinx Alliance v3.3i
- Xilinx Foundation v3.3i

The example design includes a simple user application which is intended for use as a training vehicle and design flow test.

Xilinx Alliance

Before attempting to implement a design, ensure that the Xilinx environment is properly configured for use. Additionally, you must have successfully synthesized a design.

1. Move into the physical implementation directory:

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

   This directory contains the `run_xilinx` script which calls the appropriate tools to place and route the example design. Scripts are provided for Unix and Microsoft Windows operating systems.

2. Inspect the appropriate `run_xilinx` script file. Note the following:

   a) Several special environment variables are set at the beginning of the script; these are required and must not be removed.

   b) The `ngdbuild` command lists both `../../src/xpci` and `../synthesis` as search directories. The `xpci` directory contains a netlist of the PCI interface, and the `synthesis`

directory must contain the EDIF netlist generated during design synthesis.

The `ngdbuild` command also reads a user constraints file that corresponds to a desired target device and a particular version of the PCI interface.

To target a different device or to use a different version of the PCI interface, the constraints file must be changed to match the device and interface selection. The available selections are listed in the "Family Specific Considerations" chapter.

The user constraints files which are provided with the PCI interface contain constraints that guarantee pinout and timing specifications. These constraints must always be used during processing.

Any additional constraints that pertain to the user application must be placed in this file. Before making additions to the user constraints file, back up the original so that it may be restored if necessary.

c) The `map` command requires no special arguments, but uses an input/output register packing option.

d) The `par` command, as provided in the script, uses a guide file in exact guide mode. Note that some designs do not require the use of guide files.

   To target a different device or to use a different version of the PCI interface, refer to the "Family Specific Considerations" chapter to determine which guide file, if any, is required.

   If a guide file is required, ensure that the correct guide file is used by editing the script and changing the file name. If a guide file is not required, remove the following input arguments from the `par` command line in the script:

   `-gm exact -gf ../../src/guide/`*`guidefile`*`.ncd`

   If a guide file is required, the guide file must always be used. The effort levels and delay cleanup iterations may be adjusted if necessary.

e) The `trce` command performs a static timing analysis based on the design constraints originally specified in the user constraints file.

f) The ngdanno and ngd2*hdl* commands generate a simulation model of the placed and routed design.

3. Implement the design by running the run_xilinx script.

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

If the use of a guide file is required, it is important to verify that the guiding process was successful. This may be done by inspecting the pcim_top_routed.par file. The remainder of this section is specific to designs which require the use of guide files.

Search for the following text to discover how many connections were pre-routed during the routing phase:

```
<###> connection(s) routed; <###> unrouted.
Starting router resource preassignment.
```

The number of pre-routed connections should exactly match the number listed in the selection table shown in the "Family Specific Considerations" chapter. The number of unrouted signals will vary depending on the size of the user application.

If this number does not match, the guide process has failed. This can occur for several reasons. First, check that the correct user constraints and guide files have been used. Second, verify that the user application observes all signal-driving rules presented in the *LogiCORE PCI Design Guide*.

Note: Do not attempt re-entrant routing on a guided design. Re-entrant routing must not be used as it may re-route nets that were initially guided by the guide file.

Xilinx Foundation

Before attempting to implement a design, ensure that the Xilinx environment is properly configured for use. Additionally, you must have successfully synthesized a design.

The Xilinx Foundation tools may be used from the command line exactly as shown for the Alliance tool set. Do not use the graphical user interface.

Chapter 6

Timing Simulation

This chapter describes the use of supported tools for timing simulation. These tools are:

- Cadence Verilog-XL v.3.0
- Synopsys VSS v1999.10
- Model Technology ModelSim PE v5.2e

The example design is a simple user application which is intended for use as a training vehicle and design flow test. The LogiCORE PCI32 interface ships with the design “ping32”, while the LogiCORE PCI64 interface ships with the design “ping64”. The examples in this chapter refer to “ping64”. If you are using the LogiCORE PCI32 interface, simply substitute “ping32” for “ping64”.

By default, the example design targets a 66 MHz implementation and simulates with a 15 ns clock period. If you are targeting a 33 MHz implementation, adjust the simulation clock period to 30 ns. The clock period is defined in the stimulus source file.

Cadence Verilog-XL

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

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

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

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

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

   Modify the library search path by changing `<Xilinx Install Path>` to match the Xilinx installation directory. Save the file.

3. To run the Verilog-XL simulation:

```
verilog -f ping_tb.f
```

   Verilog-XL processes the simulation files and exits. The testbench prints status messages to the console. After the simulation completes, view the `verilog.log` file to check for errors.

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

```
signalscan
```

A sample Signalscan "do file", `signalscan.do` is provided for convenience.

Synopsys VSS

Before attempting timing simulation, ensure that the VSS environment is properly configured for use. Furthermore, the `$XILINX` environment variable must be set to match the Xilinx installation directory. In addition, you must have successfully completed the implementation phase using the Xilinx tools.

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

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

```
cp ../xilinx/pcim_top_routed.sdf .
```

2. Run the `analyze_ping` script. This script will analyze the required simprim libraries, as well as the design files.

3. To run the VSS simulation:

   ```
   run_ping
   ```

   VSS processes the simulation files and halts when the testbench has finished. The testbench prints status messages to the console. After the simulation completes, view the console output to check for errors. The `ping.traces` file contains a list of signals which are recorded to the waveform database.

The Waves browser may be used to view the simulation results. A sample Waves setup "restore file", `ping.wfc` is provided for convenience.

Model Technology ModelSim

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

Verilog

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

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

   ```
   cp ../xilinx/pcim_top_routed.v .
   ```

   ```
   cp ../xilinx/pcim_top_routed.sdf .
   ```

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

   ```
   +licq_all+
   ../source/ping_tb.v
   ../source/stimulus.v
   ../source/busrecord.v
   ../source/dumb_arbiter.v
   ../source/dumb_targ32.v
   ../source/dumb_targ64.v
   ../source/glbl.v
   ```

```
./pcim_top_routed.v
+libext+.vmd+.v
-y <Xilinx Install Path>/verilog/src/simprims
```

Modify the library search path by changing `<Xilinx Install Path>` to match the Xilinx installation directory. Save the file.

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

   <Install Path>/verilog/example/post_sim

4. Create the work directory:

   vlib work

5. To run the simulation:

   do modelsim.do

VHDL

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

   cd <Install Path>/vhdl/example/post_sim

   cp ../xilinx/pcim_top_routed.vhd .

   cp ../xilinx/pcim_top_routed.sdf .

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

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

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

   <Install Path>/vhdl/example/post_sim

4. Create the simprim library. This step only needs to be done once, the first time you perform a simulation:

   vlib simprim

   vcom -work simprim <Xilinx Install Path>/vhdl/src/simprims/simprim_Vpackage.vhd

```
vcom -work simprim <Xilinx Install Path>/vhdl/src/
simprims/simprim_Vcomponents.vhd

vcom -work simprim <Xilinx Install Path>/vhdl/src/
simprims/simprim_VITAL.vhd
```

5. Create the work directory:

   ```
   vlib work
   ```

6. To run the simulation:

   ```
   do modelsim.do
   ```