Chapter 1

XSI Synopsys Interface/Tutorial Guide

The *XSI Synopsys Interface/Tutorial Guide* presents a series of smaller tutorials for FPGA Compiler and FPGA Express that guide you through VHDL and Verilog FPGA Compiler and FPGA Express design processes for XC4000, Spartan, and Virtex designs. You pick the tutorial that best suits your particular learning needs. Front-to-back tutorials, all take you through the steps in the design process including HDL design entry, design implementation, design simulation, timing simulation, and downloading the design to a functioning device.

The design used in these tutorials is called “stopwatch.” The simple design example demonstrates many system features that you can apply to more complex FPGA designs. The tutorials assume that you have a working knowledge of Synopsys FPGA Compiler or FPGA Express, as well as VHDL and Verilog. These tutorials are UNIX based.

These tutorials include a design you can download to the demo board. If you want to download to the demo board, select XC4003EPC84-3 as a target device.

This tutorial document is divided into two unmarked parts.

- An introduction that addresses issues and preparatory tasks common to all platforms
- Specific instructions for particular platforms and devices (for example, the “Spartan / XC4000 Verilog Alliance FPGA Express v2.1 Tutorial” section)

Before going on to your specific tutorial, you must first read the “Using Common Setup Procedures” section to understand general requirements and setup information. Specific setup information for the different tutorials appear in those individual sections.

This tutorial document is divided into the following sections.

- "Using Common Setup Procedures" section
- "Spartan / XC4000 Verilog FPGA Compiler VSS Tutorial" section
- "Spartan / XC4000 Verilog Alliance FPGA Express v2.1 Tutorial" section
- "Spartan / XC4000 VHDL FPGA Compiler VSS Tutorial" section
- "Spartan / XC4000 VHDL Alliance FPGA Express v2.1 Tutorial" section
- "Virtex Verilog FPGA Compiler VerilogXL Tutorial" section
- "Virtex Verilog Alliance FPGA Express v2.1 / VerilogXL v2.5 Tutorial" section
- "Virtex VHDL FPGA Compiler VSS Tutorial" section
- "Virtex VHDL Alliance FPGA Express v2.1 / VSS Tutorial" section

Throughout this document, a backslash ("\") at the end of an example line means you enter the example as one line on the command line.

Using Common Setup Procedures

This section provides information common to the device groups contained in this tutorial. Specific setup procedures appear at the beginning of each section for particular devices. The setup procedures in this section are common to all devices. With the exception of the list of tutorial files, information for FPGA Compiler appears separately from information for FPGA Express.

To use the tutorials you need A1.5i XSI and either Synopsys v1997.01 (FPGA Compiler and VSS) or better or FPGA Express v2.1 or better. Additionally, you need Verilog XL v2.5 if you wish to create the tutorial design in Verilog.

Checking the Common Tutorial Files

The following list of files is common to both FPGA Compiler and FPGA Express. These files appear (after you uncompress and untar the tutorial file) during the functional simulation process in the WORK directory you create later in this tutorial.

- cnt60.vhd
- filesf.f
- filest.f
- hex2led.vhd
- smallcntr.vhd
- stmchine.vhd
- stopwatch.vhd
- testbenchf.vhd
- testbencht.vhd
- run.script
- pr.script

These are the design files for the tutorial, including a testbench for timing simulation and a testbench for functional simulation. The top-level file in this design is stopwatch.vhd.

The LogiBLOX component in this design is called “tenths.” The LogiBLOX component for place and route is called “tenths.ngc.” The following are other LogiBLOX files

- logiblox.ini
- logiblox.log
- tenths.v
- tenths.vei

You can use LogiBLOX only with non-Virtex designs.

Using Libraries for FPGA Compiler

A1.5i XSI VHDL uses simulation libraries for functional simulation (UNISIM), timing simulation (SIMPRIM), VITAL simulation (also SIMPRIM), and additional functional simulation libraries for LogiBLOX and XDW simulation. You use the LogiBLOX library when instantiating a LogiBLOX component into a design. You use the XDW library in addition to the UNISIM library when performing post-synthesis pre-M1 simulation with non-Virtex FPGA Compiler-based flows.

Setting up for FPGA Compiler

Before you begin a tutorial for FPGA Compiler, in your home directory, create a directory to contain tutorial files. You later, in this empty directory, copy the appropriate tutorial file (for example, spartanxsivhdl.tar.Z).

Next, create a .synopsys_vss.setup using the template provided in the $XILINX/.synopsys/examples area.

1. In the $XILINX/synopsys/examples directory, copy the file template.synopsys_vss.setup into the same directory that contains the tutorial files.
2. Rename the file template.synospys_vss.setup to .synopsys_vss.setup.
3. If you copied the $XILINX/synopsys/libraries directory locally, modify the paths for the various simulation libraries as shown in the following table.

   This table assumes you copied the contents of $XILINX/synopsys/ into the path /home/data.

Table 1-1 Simulation Libraries and Paths

Library	Path
UNISIM	/home/data/libraries/sim/lib/unisims
SIMPRIM	/home/data/libraries/sim/lib/simprims
LOGIBLOX	/home/data/libraries/sim/lib/logiblox
XC9000	/home/data/libraries/sim/lib/xc9000/ftgs

If you use a version of Synopsys newer than v1997.01 and simulate with VSS, you must recompile the XSI XDW and simulation libraries. If you use a version of Synopsys newer than v1997.01 and use Verilog, you need recompile only the XSI XDW synthesis libraries.

The A1.5i XSI XDW libraries are a collection of Synopsys DesignWare libraries provided by Xilinx. There is a separate XSI XDW DesignWare library for each chip family in A1.5i. For example, there are separate families for Spartan, 4000XL, 4000EX, and Virtex. You need only recompile the libraries for the device family you target.

Make sure that the XC4000E XDW DesignWare libraries are compiled.

All A1.5i XSI libraries have compile scripts so you can compile these libraries in the $XILINX area, provided you have write permissions to the $XILINX area. If you lack write permissions, make a local copy of the $XILINX/synopsys/libraries directory and follow the instructions below, but instead of changing directories to $XILINX/synopsys/libraries, change to your local copy of that area.

1. Go to the $XILINX/synopsys/libraries/dw/src/*xxnnnnx* directory, where *xxnnnnx* is the target device name (such as XC4000E).

   a) At the UNIX prompt, enter the following

   ```
   dc_shell –f install_dw.dc
   ```

   b) After copying, uncompressing, and untarring tutorial files in the directory you previously created, in that same directory, create a directory called WORK. Type the following at the unix prompt to create WORK.

   ```
   mkdir WORK
   ```

2. After creating the work directory, you must create a .synospys_dc.setup file. You can find templates for these files in the $XILINX/.synopsys/examples area.

   In the $XILINX/synopsys/examples directory, copy the file template.synopsys_dc.setup_fc into the same directory that contains WORK. Rename template.synopsys_dc.setup_fc to .synopsys_dc.setup, adding missing lines using the A1.5i XSI tool *synlibs*. Synlibs displays the synthesis library information needed for a given die-speed combination. For example, if you synthesize the design in a 4003Epc84-3 device, to add the correct information into the .synopsys_dc.setup file for the 4000E you type the following in the same directory as the .synopsys_dc.setup file.

   ```
   synlibs –fc 4003E-3 >> .synopsys_dc.setup
   ```

   This command appends the output of synlibs into the .synopsys_dc.setup file. If you have compiled the 4000E XDW libraries in the $XILINX tree, you can proceed to checking the .synopsys_vss.setup file.

Note: This tutorial uses Spartan, 4000X, and Virtex devices. You must use the appropriate device name in the previous example when following tutorial instructions.

If you compiled the XDW libraries in the $XILINX area, you need not modify the define_design_lib line in the .synopsys_dc.setup file. If you copied locally the $XILINX/synopsys/libraries directory, change the path in the .synopsys_dc.setup file to reflect the copied directory path. For example, if you copied $XILINX/libraries to /home/data, edit the define_design_lib setting made by synlibs for the 4003E-3 shown previously to reflect the /home/data location as shown in the following example.

```
define_design_lib xdw_xc4000e /home/data/ \
libraries/dw/lib/xc4000e
```

A backslash ("\") at the end of an example line means you enter the example as one line on the command line.

Before starting the tutorial, in the directory where you created .synopsys_dc.setup and the WORK directory, type the following command.

```
ls -l
```

Make sure that directory has at least the following items.

- .synopsys_dc.setup
- .synopsys_vss.setup
- WORK

Setting Up for FPGA Express

To properly set your A1.5i environment for FPGA Express, follow the instructions provided in the A1.5i *Installation Guide*, the FPGA Express *Installation Guide*, and the VerilogXL installation guide.

Download the FPGA Express tutorial files from the Xilinx web site, uncompressing and untarring these files into the directory of your choice; you can create a new directory for the tutorial.

You can find more specific information about setting up your system for FPGA Express in the *Quick Start Guide for Alliance Series 1.5i* (referred to throughout this document as the *Quick Start Guide*).

Spartan/XC4000 Verilog FPGA Compiler VSS Tutorial

This tutorial familiarizes you with the A1.5i FPGA Compiler/VSS design flow and includes a VHDL design that you can optionally download to the demo board, if you choose a XC4003EPC84-3 as a target device.

This tutorial takes you from functional simulation to timing simulation. The tutorial presents common design flow tasks, such as locking pins and setting slew rate.

Getting Ready for this Tutorial

To use this tutorial, you need the software described in the "Using Common Setup Procedures" section. If you use a version of Synopsys newer than v1997.01, you must follow the additional steps in the "Setting Up for the Spartan/XC4000 Verilog FPGA Compiler Tutorial" section. By default, this tutorial uses the XC4000E family as a target design, but you can use other FPGA families supported under A1.5i XSI as well. If you want to target a device other than the XC4000E, reference the directories that apply to that family (for example, the XDW libraries have separate directories for XC4000XL, Spartan, and Virtex). Use synlibs with the exact die-speed target desired.

Setting Up for the Spartan/XC4000 Verilog FPGA Compiler Tutorial

To use this tutorial, ensure installation of your Xilinx and Synopsys software and know where the software resides on your system. If you use a version of Synopsys newer than v1997.01, you must re-compile the XSI XDW and simulation libraries. Refer to the setup instructions in the "Setting up for FPGA Compiler" section.

For this tutorial, recompile only the libraries related to synthesizing a XC4000E device (if using a version of Synopsys newer than v1997.01), and ensure compilation of the XC4000E XDW DesignWare libraries. To compile the XC4000E device libraries, complete the following steps.

1. Change directories to the $XILINX/synopsys/libraries/dw/src/xc4000e directory.

2. At the UNIX prompt type the following.

   ```
   dc_shell –f install_dw.dc
   ```

3. Copy the file 4kxsiverilog.tar.Z into the empty directory you competed in the “Setting up for FPGA Compiler” section. Uncompress and untar this file.

4. Use synlibs to add the missing lines to the .synopsys_dc.setup file you renamed earlier in the “Setting up for FPGA Compiler” section. In this tutorial you synthesizing the design in a 4003Epc84-3 device, so to add the correct information into the .synopsys_dc.setup file for the s05pc84-3, type the following in the same directory as the .synopsys_dc.setup file.

   ```
   synlibs -fc 4003E-3 >> .synopsys_dc.setup
   ```

   This appends the output of synlibs into the .synopsys_dc.setup file. If you compiled the Spartan XDW libraries in the $XILINX tree, you can proceed to checking the .synopsys_vss.setup file. If you compiled the XDW libraries in the $XILINX area, then you need not modify the define_design_lib line in the .synopsys_dc.setup file.

5. If you copied the $XILINX/synopsys/libraries directory locally, change the path in the .synopsys_dc.setup file to reflect the copied directory path. For example, if you copied $XILINX/libraries to /home/data, edit the define_design_lib setting made by synlibs for the 4003E-3 shown previously to reflect the /home/data location.

   ```
   define_design_lib xdw_xc4000e /home/data/ \
   libraries/dw/lib/xc4000e
   ```

Conducting Functional Simulation

You finished most of the setup when you created the WORK directory and made the .synopsys_vss.setup file. Now you compile the design files and testbench and run the VSS simulation tool.

In the A1.5i XSI VSS functional simulation flow you can simulate instantiated XSI cells such as FDCE, or OSC4. Additionally, by using the UNISIM simulation libraries, you can simulate and implement the GSR without impact to the design or testbench.

1. Change to the /home/user/tutorial directory (where the .synopsys_dc.setup, .synopsys_vss.setup file, and WORK directory reside).
2. Check to make sure the files listed in the "Checking the Common Tutorial Files" section also reside in this directory.
3. Perform functional simulation by running the VerilogXL command verilog with the data file filesf.f. At the UNIX prompt enter the following.

   verilog –f filesf.f

   VerilogXL issues error messages when you attempt to simulate, three of which appear in the following example.

```
Compiling source file "testbenchf.v"
Compiling source file "stopwatch.v"
Compiling source file "stmchine.v"
Compiling source file "hex2led.v"
Compiling source file "cnt60.v"
Compiling source file "smallcntr.v"
Compiling source file "tenths.v"

Error!   Module or primitive (OSC4) not defined         [Verilog-MOPND]
        "stopwatch.v", 22: OSC4 OSCILLATOR(.F500K(oscout)
          );

Error!   Module or primitive (BUFG) not defined         [Verilog-MOPND]
        "stopwatch.v", 24: BUFG CLOCKBUF(.I(oscout), .O(
          clkint));

Error!   Module or primitive (X_FF) not defined         [Verilog-MOPND]
        "tenths.v", 46: X_FF FLOP0(.IN(Q_OUT[9]), .CLK(
        CLOCK), .CE(CLK_EN), .SET(FLOP0_GSR_OR), .RST(
          GND), .OUT(Q_OUT[0]));
```

4. Place the ‘uselib directive in the top-level file of this design, stopwatch.v by adding this line to the top of your stopwatch.v file.

   `uselib dir=$XILINX/verilog/src/UNI4000E libext=.v

   You do this because the Verilog code you tried to simulate contains instantiations of LogiBLOX and instantiated XSI synthesis library cells (such as OSC4, FDCE, and BUFG). You use the ‘uselib directive to tell the Verilog simulator where to find the models for these cells.

5. Replace the $XILINX text with the explicit path in your environment. If you set $XILINX to /home/software/xilinx, than place the ‘uselib line in the top of the stopwatch.v file as follows.

   **`uselib dir=/home/software/xilinx/verilog/src/ **
   UNI4000E libext=.v

6. Similarly, to the top of the .v file created for the LogiBLOX module in this design, tenths.v, add the following ‘uselib line.

   `uselib dir=$XILINX/verilog/data libext=.vmd

7. Replace $XILINX with the explicit path in your setup. So, using the previous example of $XILINX set to /home/software/xilinx, the ‘uselib line in the tenths.v file appears as follows.

   ```
   `uselib dir=/home/software/xilinx/verilog/data \
   libext=.vmd
   ```

8. After making these two changes, re-run Verilog.

   verilog –f filesf.f

Synthesizing Your Design

In this section of the tutorial, you synthesize the design and create a place and routed NCD file. After creating the place and routed NCD file, you can optionally create a BIT file for downloading to the demo board using bitgen and promgne, or the Hardware Debugger.

To synthesize a design with FPGA Compiler, you need to create a compile script. You can find a default compile script for your modification in the A1.5i software. Copy the file $XILINX/template.fpga.script.4kex into your /home/user/tutorial directory, which contains the .synopsys_dc.setup and .synopsys_vss.setup files you created earlier.

1. Rename the file template.synopsys.dc.setup.4kes to run.script.

   The file template.synopsys_dc.setup.4kex is a template for a compile script for XC4000 families. Spartan can use this file as a template for synthesizing with FPGA Express.

2. Open the file run.script in a text editor.

   The file run.script is setup only to compile one VHDL file and you must comment out several lines not relevant to the synthesis of this particular design.

   The following example shows an unmodified run.script file (using the file template.synopsys_dc.setup.4kex).

```
/* ================================================*/
/*    Sample Script for Synopsys to Xilinx Using     */
/*                  FPGA Compiler                    */
/*                                                   */
/*  Targets the Xilinx XC4028EX-3 and assumes a VHDL */
/*         source file by way of an example.         */
/*                                                   */
/*   For general use with XC4000E/EX architectures.  */
/*      Not suitable for use with XC3000A/XC5200     */
/*                  architectures.                   */
/* ================================================*/

/* ============================================== */
/* Set the name of the design's top-level module.    */
/* (Makes the script more readable and portable.)    */
/* Also set some useful variables to record the      */
/* designer and company name.                        */
/* ============================================== */
   TOP = <design_name>
                       /* ========================== */
                       /* Note: Assumes design file- */
```

```
                        /* name and entity name are   */
                        /* the same (minus extension) */
                        /* ========================== */
   designer = “XSI Team”
   company  = “Xilinx, Inc”
   part     = “4028expg299-3”
/* ================================================ */
/* Analyze and Elaborate the design file and specify */
/* the design file format.     */
/* ================================================ */
   analyze -format vhdl TOP + “.vhd”
                      /* ============================ */
                      /* You must analyze lower-level */
                      /* hierarchy modules here       */
                      /* ============================ */
   elaborate TOP
/* ================================================ */
/* Set the current design to the top level.         */
/* ================================================ */
   current_design TOP
/* ================================================ */
/* Set the synthesis design constraints.            */
/* ================================================ */
   remove_constraint –all
/* Some example constraints */
   create_clock <clock_port_name> -period 50
   set_input_delay 5 -clock <clock_port_name> \
     { <a_list_of_input_ports> }
   set_output_delay 5 -clock <clock_port_name> \
     { <a_list_of_output_ports> }
```

```
   set_max_delay 100 -from <source> -to <destination>
   set_false_path -from <source> -to <destination>
/* ================================================ */
/* Indicate those ports on the top-level module that */
/* should become chip-level I/O pads. Assign any I/O */
/* attributes or parameters and perform the I/O      */
/* synthesis.                                       */
/* ================================================ */
   set_port_is_pad "*"
/* Some example I/O parameters */
   set_pad_type –pullup <port_name>
   set_pad_type –no_clock all_inputs()
   set_pad_type –clock <clock_port_name>
   set_pad_type –exact BUFGS_F <hi_fanout_port_name>
   set_pad_type –slewrate HIGH all_outputs()
                  /* ============================== */
                  /* Note: Synopsys slew-control=  */
                  /* HIGH is the same as Xilinx's  */
                  /* slewrate=SLOW. Synopsys slew- */
                  /* control=LOW is same as Xilinx */
                  /* slewrate=FAST.                */
                  /* ============================== */
   insert_pads
/* ================================================ */
/* Synthesize and optimize the design               */
/* ================================================ */
   compile –boundary_optimization
/* ================================================ */
/* Write the design report files.                   */
/* ================================================ */
```

```
   report_fpga > TOP + ".fpga"
   report_timing > TOP + ".timing"
/* ================================================ */
/* Write out the design to a DB file. (Post compile) */
/* ================================================ */
   write –format db –hierarchy –output TOP + "_compiled.db"
/* ================================================ */
/* Replace CLBs and IOBs with gates.                */
/* ================================================ */
   replace_fpga
/* ================================================ */
/* Set the part type for the output netlist.     */
/* ================================================ */
   set_attribute TOP "part" –type string part
/* ================================================ */
/* Optional attribute to remove the FPGA Compiler's  */
/* mapping structures from the design. This permits  */
/* The Xilinx design implementation tools to map the */
/* design instead.                                  */
/* ================================================ */

/* set_attribute find(design,"*") "xnfout_write_map_symbols" \
     -type boolean FALSE  */

/* ================================================ */
/* Add any I/O constraints to the design.           */
/* ================================================ */
   set_attribute <port_name> "pad_location" \
     -type string "<pad_location>"
/* ================================================ */
```

```
/* Save design in XNF format as <design>.sxnf        */
/* ================================================= */
   ungroup –all –flatten
   write –format xnf –hierarchy –output TOP + “.sxnf”
/* ================================================= */
/* Write out the design to a DB. (Post replace_fpga) */
/* ================================================= */
   write –format db –hierarchy –output TOP + “.db”
/* ================================================= */
/* Write-out the timing constraints that were        */
/* applied earlier. (Note that any design hierarchy  */
/* needs to be flattened before the constraints are  */
/* written-out.)                                     */
/* ================================================= */
   write_script > TOP + “.dc”
/* ================================================= */
/* Call the Synopsys-to-Xilinx constraints translator*/
/* utility DC2NCF to convert the Synopsys constraints*/
/* to a Xilinx NCF file. You may like to view        */
/* dc2ncf.log to review the translation process.     */
/* ================================================= */
   sh dc2ncf TOP + “.dc”
/* ================================================= */
/* Exit the Compiler.                                */
/* ================================================= */
   exit
/* ================================================= */
/* Now run the Xilinx design implementation tools.   */
/* ================================================= */
```

The modified run.script file for the files in this tutorial (using the file run.script) follows. Before using this script, notice the following items.

- Compilation occurs from the bottom up.
- The VHDL file for the LogiBLOX counter tenths.vhd is not compiled.

A file called “tenths.v” reads into the compile script (not the same file as the ‘tenths.v’ file used in functional simulation). In this case, tenths.v is just a place holder.

```
/* ================================================*/
/*    Sample Script for Synopsys to Xilinx Using     */
/*                  FPGA Compiler                    */
/*                                                   */
/*  Targets the Xilinx XC4028EX-3 and assumes a VHDL */
/*         source file by way of an example.         */
/*                                                   */
/*   For general use with XC4000E/EX architectures.  */
/*      Not suitable for use with XC3000A/XC5200     */
/*                  architectures.                   */
/* ================================================*/

/* ============================================== */
/* Set the name of the design’s top-level module.    */
/* (Makes the script more readable and portable.)    */
/* Also set some useful variables to record the      */
/* designer and company name.                        */
/* ============================================== */
   TOP = stopwatch
                       /* ========================== */
                       /* Note: Assumes design file- */
                       /* name and entity name are   */
```

```
                        /* the same (minus extension) */
                        /* ========================== */
   designer = “XSI Team”
   company  = “Xilinx, Inc”
   part     = “4003Epc84-3”
/* ================================================ */
/* Analyze and Elaborate the design file and specify */
/* the design file format.     */
/* ================================================ */
read -format verilog “tenths.v”
read -format verilog “smallcntr.v”
read -format verilog “cnt60.v”
read -format verilog “hex2led.v”
read -format verilog “stmchine.v”
read -format verilog TOP + “.v”
                      /* ============================ */
                      /* You must analyze lower-level */
                      /* hierarchy modules here       */
                      /* ============================ */
set_dont_touch “OSCILLATOR”
   uniquify
/* ================================================ */
/* Set the current design to the top level.         */
/* ================================================ */
   current_design TOP
/* ================================================ */
/* Set the synthesis design constraints.            */
/* ================================================ */
   remove_constraint –all
/* Some example constraints */
```

```
/*   create_clock <clock_port_name> -period 50
   set_input_delay 5 -clock <clock_port_name> \
     { <a_list_of_input_ports> }
   set_output_delay 5 -clock <clock_port_name> \
     { <a_list_of_output_ports> }
   set_max_delay 100 -from <source> -to <destination>
   set_false_path -from <source> -to <destination> */
/* ================================================ */
/* Indicate those ports on the top-level module that */
/* should become chip-level I/O pads. Assign any I/O */
/* attributes or parameters and perform the I/O      */
/* synthesis.                                        */
/* ================================================ */
   set_port_is_pad “*”
/* Some example I/O parameters */
/*   set_pad_type –pullup <port_name>
   set_pad_type –no_clock all_inputs()
   set_pad_type –clock <clock_port_name>
   set_pad_type –exact BUFGS_F <hi_fanout_port_name>
   set_pad_type –slewrate HIGH all_outputs() */
                  /* ============================== */
                  /* Note: Synopsys slew-control=  */
                  /* HIGH is the same as Xilinx’s  */
                  /* slewrate=SLOW. Synopsys slew- */
                  /* control=LOW is same as Xilinx */
                  /* slewrate=FAST.                */
                  /* ============================== */
   insert_pads
/* ================================================ */
/* Synthesize and optimize the design               */
```

```
/* ================================================ */
   compile –boundary_optimization
/* ================================================ */
/* Write the design report files.                   */
/* ================================================ */
/*   report_fpga > TOP + “.fpga”
   report_timing > TOP + “.timing” */
/* ================================================ */
/* Write out the design to a DB file. (Post compile) */
/* ================================================ */
   write –format db –hierarchy –output TOP + “_compiled.db”
/* ================================================ */
/* Replace CLBs and IOBs with gates.                */
/* ================================================ */
   replace_fpga
/* ================================================ */
/* Set the part type for the output netlist.     */
/* ================================================ */
   set_attribute TOP “part” –type string part
/* ================================================ */
/* Optional attribute to remove the FPGA Compiler’s  */
/* mapping structures from the design. This permits  */
/* The Xilinx design implementation tools to map the */
/* design instead.                                  */
/* ================================================ */
/* set_attribute find(design,”*”) “xnfout_write_map_symbols” \
     -type boolean FALSE  */
/* ================================================ */
/* Add any I/O constraints to the design.           */
/* ================================================ */
```

```
/*   set_attribute <port_name> “pad_location” \
     -type string “<pad_location>” */
/* ================================================ */
/* Save design in XNF format as <design>.sxnf       */
/* ================================================ */
   ungroup –all –flatten
   remove_design “tenths”
   write –format xnf –hierarchy –output “watch.sxnf”
/* ================================================ */
/* Write out the design to a DB. (Post replace_fpga) */
/* ================================================ */
   write –format db –hierarchy –output TOP + “.db”
/* ================================================ */
/* Write-out the timing constraints that were       */
/* applied earlier. (Note that any design hierarchy  */
/* needs to be flattened before the constraints are  */
/* written-out.)                                    */
/* ================================================ */
/*   write_script > TOP + “.dc” */
/* ================================================ */
/* Call the Synopsys-to-Xilinx constraints translator*/
/* utility DC2NCF to convert the Synopsys constraints*/
/* to a Xilinx NCF file. You may like to view       */
/* dc2ncf.log to review the translation process.    */
/* ================================================ */
/*   sh dc2ncf TOP + “.dc” */
/* ================================================ */
/* Exit the Compiler.                               */
/* ================================================ */
/*   exit */
```

```
/* ================================================ */
/* Now run the Xilinx design implementation tools.   */
/* ================================================ */
```

You need a file which indicates the pin directions of the module when instantiating a LogiBLOX component in FPGA Compiler. The tenths.v file referenced in the run.script file was created by renaming and editing the tenths.vei file. The new tenths.v file contains the following.

```
module tenths(CLK_EN, CLOCK, ASYNC_CTRL, Q_OUT, TERM_CNT);
input CLK_EN;
input CLOCK;
input ASYNC_CTRL;
output [9:0] Q_OUT;
output TERM_CNT;
endmodule
```

You can create the same file by deleting the first 13 lines in the tenths.vei file and renaming the tenths.vei file to tenths.v.

When compiling a XC4000 design in FPGA Compiler, the proper type of output for place and route in A1.5i is an SXNF file.

Dont_touch commands appear in the script. Whenever you instantiate a component from the XSI synthesis library, you must place a Dont_touch on the instance to prevent Synopsys from deleting or modifying the library cell.

Using Design Analyzer

Start Design Analyzer by typing the following command in the directory that contains the .synopsys_dc.setup file for this design.

```
design_analyzer &
```

This launches the Design Analyzer GUI. When the GUI appears, run the script run.script by selecting Execute Script from the Setup pull-down menu. A pop-up window appears where you can select run.script.

If the script runs successfully, the script stops and a SXNF file generates. If an error occurs, check the following.

- Make sure that the .synopsys_dc.setup file is set up correctly.
- Make sure that the paths referenced in the .synopsys_dc.setup file exist.
- Make sure that the XDW synthesis libraries are compiled for the version of Synopsys you are using.
- Make sure you issue the Design_analyzer & command in the directory that contains the .synopsys_dc.setup file when you start Design Analyzer.

Placing and Routing the SXNF File

Use the SXNF file produced by Design Analyzer to place and route the design for timing simulation using the following script.

```
#!/bin/csh -f

ngdbuild –p 4003EPC84-4 watch.sxnf

map watch.ngd

par watch.ncd stopwatch_r.ncd

ngdanno stopwatch_r.ncd

ngd2ver –ul stopwatch_r.nga
```

Optionally, you can place and route the SXNF files by using the A1.5i GUI's. Please refer to the *Quick Start Guide Tutorial* for more information about using the GUI for place and route.

Conducting Timing Simulation

To perform timing simulation, you must create an SDF file and structural Verilog produced by NGD2VER, along with a testbench. The following steps show how to conduct timing simulation.

Note: Run NGD2VER with the –ul option before running timing simulation.

Run the following command at the command-line.

```
verilog –f filest.f
```

filest.f contains the names of the two files used in timing simulation. By default, the Verilog file produced by NGD2VER uses the $sdf_annotate directive which annotates the SDF file with the Verilog file from NGD2VER.

Spartan/XC4000 Verilog Alliance FPGA Express v2.1 Tutorial

This tutorial familiarizes you with the A1.5i XSI Verilog FPGA Express v2.1 / VerilogXL v2.5 design flow and includes a VHDL design that you can optionally download to the demo board, if you choose a XC4003EPC84-3 as a target device.

This tutorial presents common tasks to accomplish in this design flow such as locking pins and setting slew rate.

Getting Ready for This Tutorial

To use this tutorial, you need the software described in the "Using Common Setup Procedures" section. By default, this tutorial uses the XC4000E family as a target design, but you can use other FPGA families supported under A1.5i XSI as well. If you want to target a device other than the XC4000E, reference the directories that apply to that family (for example, the XDW libraries have separate directories for XC4000XL, Spartan, and Virtex). Use synlibs with the exact die-speed target desired.

Setting Up for the Spartan/XC4000 Verilog Alliance FPGA Express v2.1 Tutorial

To use this tutorial, ensure installation of your Xilinx and Synopsys software and know where the software resides on your system. Download the files for this tutorial, 4kxsiverexp.tar.Z, from the Xilinx web site. Untar and uncompress this file in a directory of your choosing, but this tutorial assumes that you place the files in the /home/user/tutorial directory.

Conducting Functional Simulation

Compile the design files and testbench with VerilogXL. If you untarred and uncompressed files in a directory other than /home/user/tutorial, replace your path appropriately in the following instructions. In the A1.5i XSI VerilogXL functional simulation flow you can simulate instantiated XSI cells such as FDCE and OSC4.

1. Change directories to the /home/user/tutorial directory (or where you installed the tutorial files). Make sure you previously

set up a .synopsys_dc.setup and .synopsys_vss.setup file in this directory, along with a WORK directory, as described in the "Using Common Setup Procedures" section.

2. Run the VerilogXL command verilog with the data file filesf.f to perform functional simulation. At the UNIX prompt enter the following.

   verilog –f filesf.f

   VerilogXL issues many error messages when you attempt to simulate, including the following three.

```
Compiling source file "testbenchf.v"
Compiling source file "stopwatch.v"
Compiling source file "stmchine.v"
Compiling source file "hex2led.v"
Compiling source file "cnt60.v"
Compiling source file "smallcntr.v"
Compiling source file "tenths.v"

Error!   Module or primitive (OSC4) not defined          [Verilog-MOPND]
        "stopwatch.v", 22: OSC4 OSCILLATOR(.F500K(oscout)
          );

Error!   Module or primitive (BUFG) not defined          [Verilog-MOPND]
        "stopwatch.v", 24: BUFG CLOCKBUF(.I(oscout), .O(
          clkint));

Error!   Module or primitive (X_FF) not defined          [Verilog-MOPND]
        "tenths.v", 46: X_FF FLOP0(.IN(Q_OUT[9]), .CLK(
        CLOCK), .CE(CLK_EN), .SET(FLOP0_GSR_OR), .RST(
          GND), .OUT(Q_OUT[0]));
```

3. Because the Verilog code you tried to simulate contains instantiations of LogiBLOX and instantiated XSI synthesis library cells

(such OSC4, FDCE, and BUFG, for example), you must tell the Verilog simulator where to find the models for these cells. You do this by placing the ‘uselib directive in the top-level file of this design, stopwatch.v. Add the following line to the top of your stopwatch.v file.

```
`uselib dir=$XILINX/verilog/src/UNI4000E libext=.v
```

4. Replace the $XILINX text with the explicit path in your environment. If you set $XILINX to /home/software/xilinx, enter the following ‘uselib line in the top of the stopwatch.v file.

```
`uselib dir=/home/software/xilinx/verilog/ \
src/UNI4000E libext=.v
```

5. Similarly, to the top of the .v file created for the LogiBLOX module in this design, tenths.v, add the following ‘uselib line.

```
`uselib dir=$XILINX/verilog/data libext=.vmd
```

6. Replace $XILINX with the explicit path in your setup. Using the previous example of $XILINX set to /home/software/xilinx, you find the following ‘uselib line in the tenths.v file.

```
`uselib dir=/home/software/xilinx/verilog \
/data libext=.vmd
```

7. After making these two changes, re-run functional simulation by typing the following command.

```
verilog –f filesf.f
```

Synthesizing Your Design

In this section of the tutorial, you synthesize the design and create a place and routed NCD file. After creating the place and routed NCD file, you can optionally proceed to create a BIT file for downloading to the demo board, using bitgen and promgne, or the Hardware Debugger. For more information about using FPGA Express’s GUI, please refer to the FPGA Express on-line help.

The following steps show you how to create and synthesize a project in FPGA Express.

1. Create an FPGA Express Project, enter source files, and specify a target device(4003EPC84-3), as shown in the following figure.

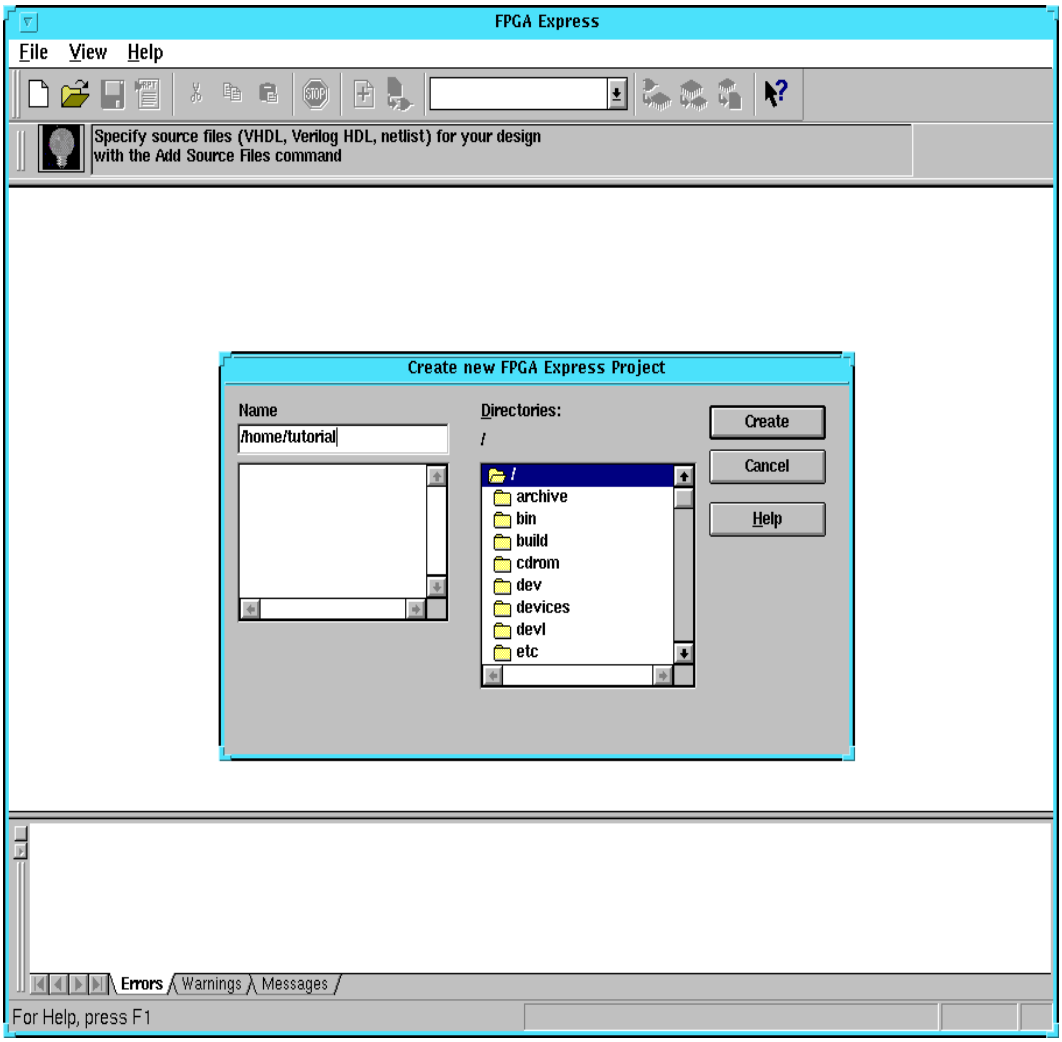

Figure 1-1 FPGA Express Create Project Window

2. Select the top level entity and select a target device, as shown in the next figure.

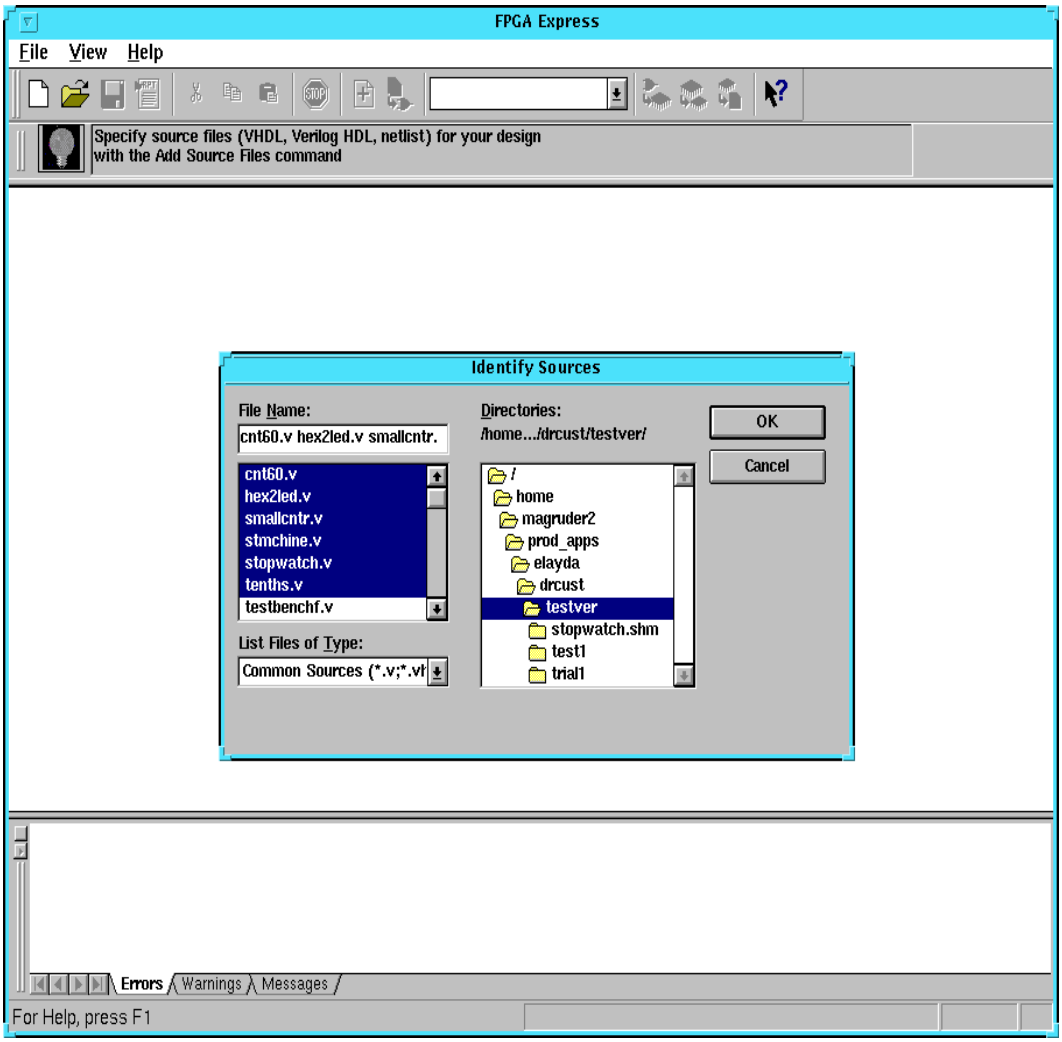

Figure 1-2 FPGA Express Add Design Files Dialog

3. An implementation appears in the right-hand window. Select the implementation and than clock on the Optimize button on the tool bar, as shown in the following illustration.

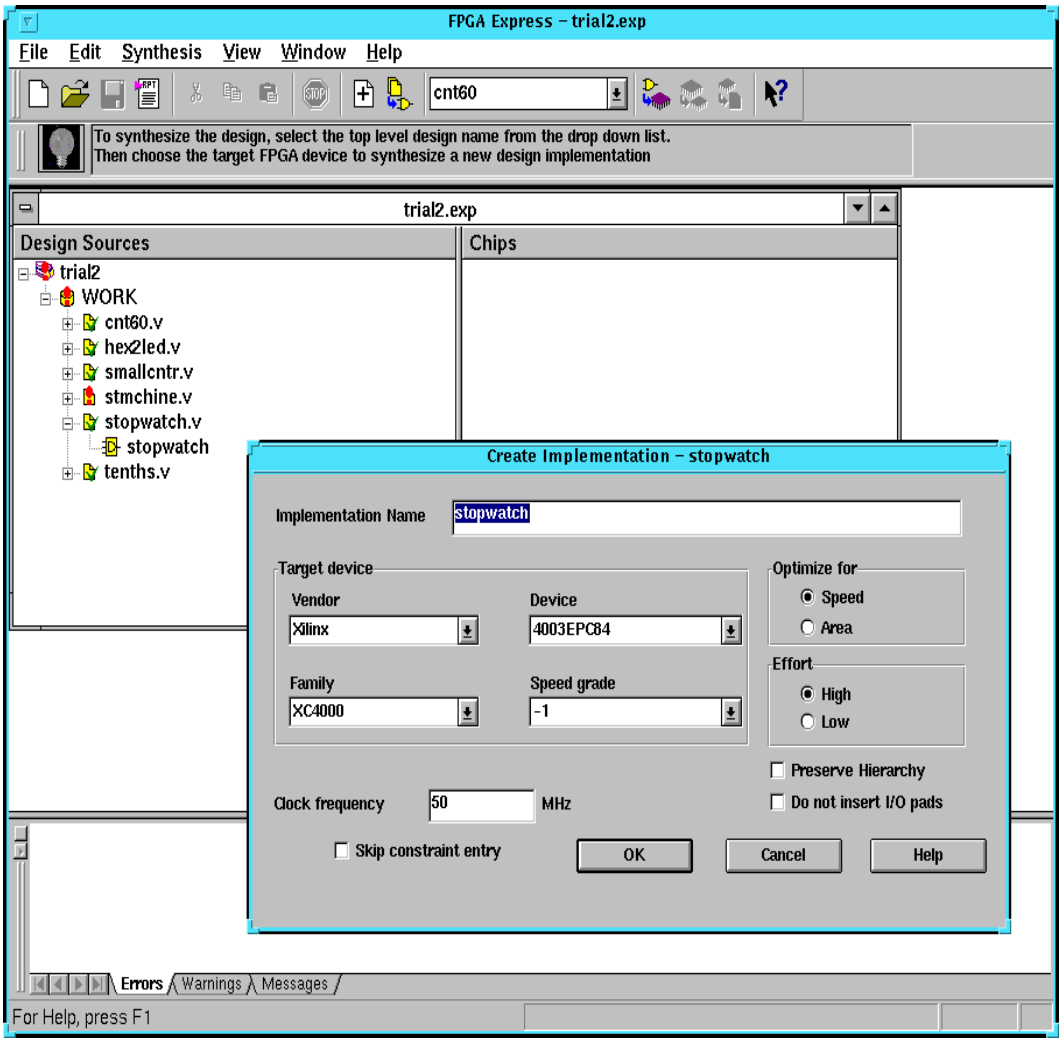

Figure 1-3 FPGA Express Create Implementation Dialog

4. Select the optimized design and write out the netlist by clicking on the Export Netlist button on the toolbar, as the next figure shows.

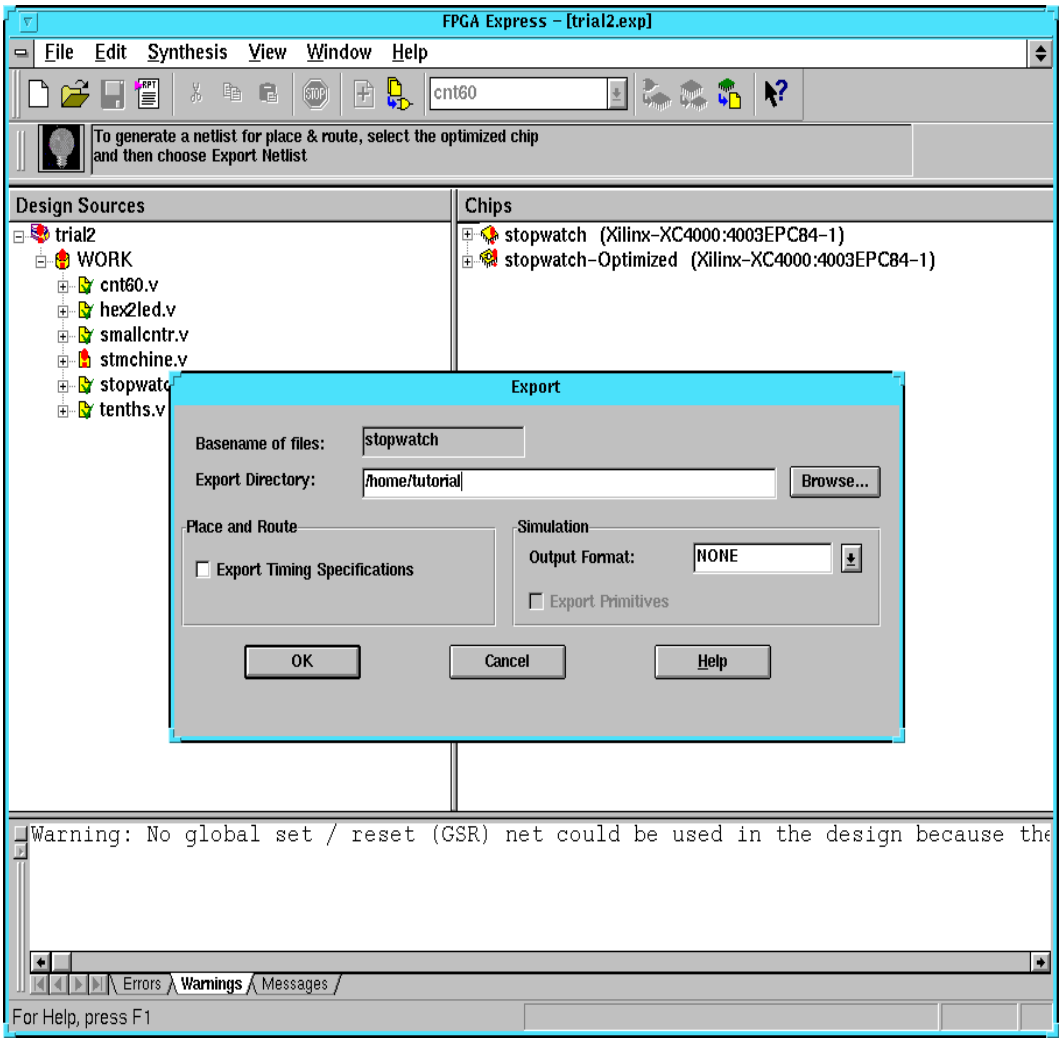

Figure 1-4 FPGA Express Netlist

5. Take the SXNF file produced by Design Analyzer and place and route the design for timing simulation using the following script.

```
#!/bin/csh -f

ngdbuild –p 4003EPC84-4 watch.sxnf

map watch.ngd
```

```
par watch.ncd stopwatch_r.ncd

ngdanno stopwatch_r.ncd

ngd2ver –ul stopwatch_r.nga
```

6. Optionally, you can place and route the SXNF files using the A1.5i GUI, as shown in the following figure. Refer the *Quick Start Guide Tutorial* for more information for using the GUI for place and route.

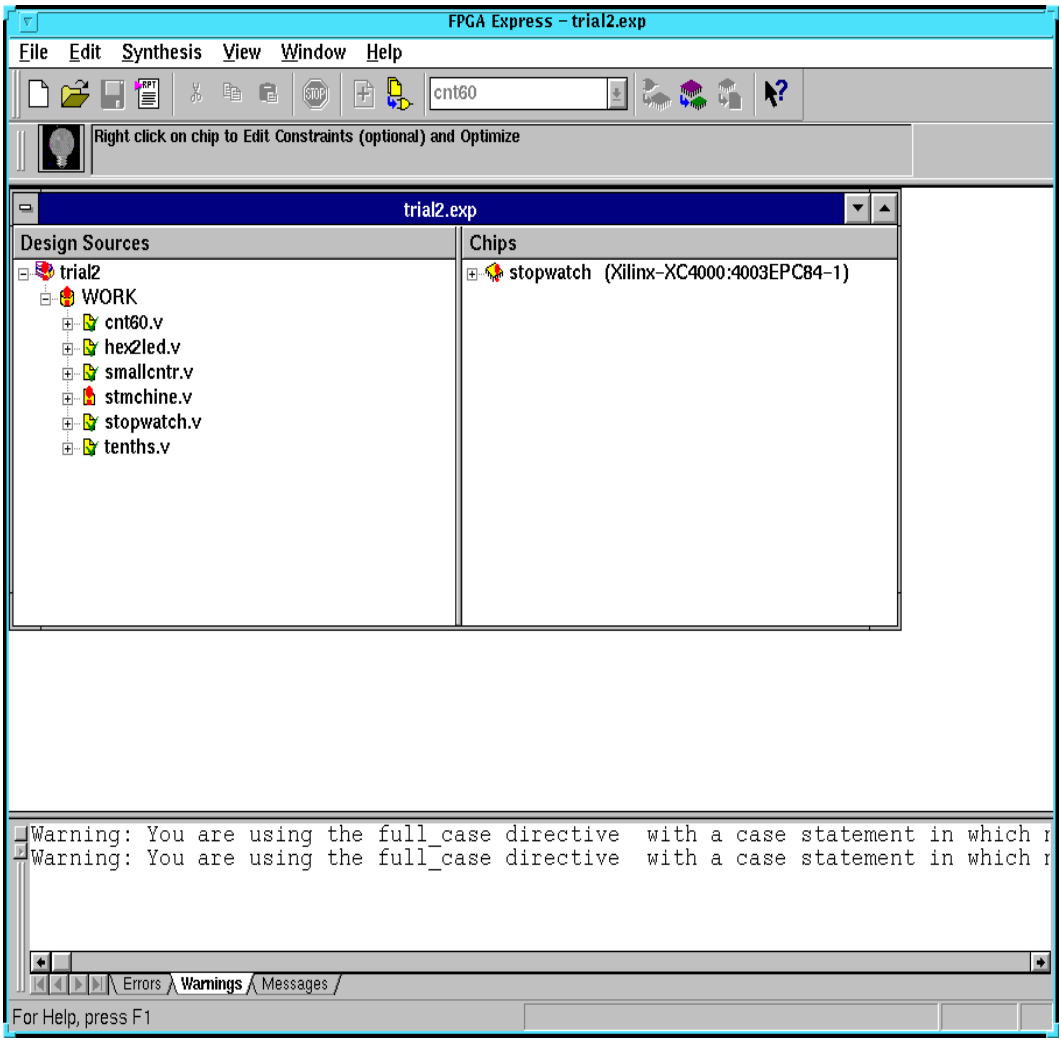

Figure 1-5 Placing and Routing SXNF Files

Conducting Timing Simulation

To perform timing simulation, you must create an SDF file and structural Verilog file using NGD2VER, along with a testbench.

To perform timing simulation, enter the following command at the command line.

```
verilog –f filest.f
```

filest.f contains the names of the two files used in timing simulation. By default, the Verilog file produced by NGD2VER uses the $sdf_annotate directive, which annotates the SDF file with the Verilog file from NGD2VER.

Note: Run NGD2VER with the –ul option before conducting timing simulation.

Spartan/XC4000 VHDL FPGA Compiler VSS Tutorial

This tutorial familiarizes you with the A1.5i FPGA Compiler/VSS design flow and includes a VHDL design you can optionally downloaded to the demo board, if you choose a XC4003EPC84-3 as a target device.

This tutorial takes you from functional simulation to timing simulation, presenting common tasks such as locking pins and setting slew rate. The tutorial has three major parts, functional simulation, synthesis, and timing simulation.

The functional simulation flow with FPGA Compiler and VSS has four possible flows, including the traditional pre-synthesis flow known as functional simulation. This tutorial uses the pre-synthesis simulation.

This tutorial assumes you copied the $XILINX/synopsys/libraries directory to /home/data, which the path used as an example in the "Using Common Setup Procedures" section.

Getting Ready for this Tutorial

To use this tutorial, you need the software described in the "Using Common Setup Procedures" section. If you use a version of Synopsys newer than v1997.01, you must follow the additional steps in the "Setting Up for the Spartan/XC4000 Verilog FPGA Compiler Tutorial" section. By default, this tutorial uses the XC4000E family as a target design, but you can use other FPGA families supported under A1.5i XSI as well. If you want to target a device other than the XC4000E, reference the directories that apply to that family (for example, the XDW libraries have separate directories for XC4000XL, Spartan, and Virtex). Use synlibs with the exact die-speed target desired.

Setting Up for the Spartan/XC4000 VHDL FPGA Compiler VSS Tutorial

To use this tutorial, ensure installation of your Xilinx and Synopsys software and know where the software resides on your system. If you use a version of Synopsys newer than v1997.01, you must re-compile the XSI XDW and simulation libraries. Refer to the setup instructions in the "Setting up for FPGA Compiler" section.

The A1.5i simulation libraries come in two parts, a functional simulation part and a timing simulation part. The A1.5i XSI VHDL functional simulation libraries are called UNISIM and the A1.5i XSI VHDL timing simulation libraries are called SIMPRIM libraries. The SIMPRIM library is a VITAL simulation library.

Additionally, A1.5i XSI provides two other functional simulation libraries, the LOGIBLOX and XDW simulation libraries. You use the LOGIBLOX when a design uses instantiated LogiBLOX components, such as a RAM. You use the XDW simulation library in addition to the UNISIM library when performing post-synthesis pre-M1 simulation.

For this tutorial, recompile only the libraries related to synthesizing a XC4000E device (if using a version of Synopsys newer than v1997.01), and ensure compilation of the XC4000E XDW DesignWare libraries.

Make sure you compile the Spartan XDW DesignWare libraries, XDW simulation libraries, LogiBLOX simulation libraries, UNISIM simulation libraries, and SIMPRIM libraries. All of these libraries contain compile scripts which let you compile these libraries in the $XILINX area. However, you need write permission to the $XILINX area to use these scripts.

If you do not have write permissions, make a local copy of the $XILINX/synopsys/libraries directory and use the following instructions. Instead of changing directories to $XILINX/synopsys/libraries, however, change directories to your local copy of that area.

1. Compile the libraries.

   a) Change directories to the $XILINX/synopsys/libraries/dw/src/spartan directory.

   b) Type the following command at the UNIX prompt.

   ```
   dc_shell -f install_dw.dc
   ```

c) Change directories to the $XILINX/synopsys/libraries/sim/src/logiblox directory.

d) Type the following command at the UNIX prompt.

```
./analyze.csh
```

e) Change directories to the $XILINX/synopsys/libraries/sim/src/simprims directory.

f) Type the following command at the UNIX prompt.

```
./analyze.csh
```

g) Change directories to the $XILINX/synopsys/libraries/sim/src/unisims directory.

h) Type the following command at the UNIX prompt.

```
./analyze.csh
```

i) Change directories to the $XILINX/synopsys/libraries/sim/src/xdw directory.

j) Type the following command at the UNIX prompt.

```
./analyze.csh
```

After compiling the libraries, you must create a directory where you run the tutorial. This directory contains the tutorial HDL files, along with the .synopsys_dc.setup and .synopsys_vss.setup you created in the "Setting up for FPGA Compiler" section.

2. In this empty directory, copy the file spartanxsivhdl.tar.Z. Uncompress and untar this file.

3. Before starting on the tutorial, in the directory where you created your .synopsys_dc.setup, .synopsys_vss.setup, and where you uncompressed and untarred the file spartanxsivhdl.tar.Z file, type the following command.

   ```
   ls -l
   ```

   Make sure that directory has the following items.

   - .synopsys_dc.setup
   - .synopsys_vss.setup
   - WORK
   - func

- synth
- time

The func directory contains all files and scripts needed for functional simulation. The synth directory contains all files and scripts needed for synthesis. The time directory contains files.

Conducting Functional Simulation

You finished most of the setup when you created the WORK directory and made the .synopsys_vss.setup file. Now, you need to compile the design files and testbench and run the VSS simulation tool. This tutorial assumes that you uncompressed and untarred the tutorial files and placed your setup files in /home/user/tutorial. If you did not, replace your path appropriately in the following instructions.

In the A1.5i XSI VSS functional simulation flow you can simulate instantiated XSI cells such as FDCE and OSC4. Additionally, by using the UNISIM simulation libraries, you can simulate and implement the GSR without impact to the design or testbench.

To conduct functional simulation, use the following steps.

1. Change directories to the /home/user/tutorial directory. and ensure all the appropriate tutorial files listed in the “Checking the Common Tutorial Files” section reside there.
2. You perform functional simulation by running the script func.script. But before running the script, open the file func.script in a text editor and note the following items in this file.
   - Simulation files read in from the bottom up.
   - The testbench reads in last.
   - vhdlan uses the –i option. Always use vhdlan with the –i option. By default, vhdlan uses the –c option, which only works if your system uses a certain type of C compiler. If you want to use the –c option with vhdlan, refer to the Synopsys web site for the proper setup.
   - The contents of the func.sim file.

```
#!/bin/csh -f

rm –r WORK
```

```
mkdir WORK
vhdlan –i tenths.vhd
vhdlan –i smallcntr.vhd
vhdlan –i cnt60.vhd
vhdlan –i hex2led.vhd
vhdlan –i stmchine.vhd
vhdlan –i stopwatch.vhd
vhdlan –i testbenchf.vhd
vhdlsim –e commandf.txt overall
```

Close the file after examining it.

3. Also open the file tenths.vhd, created by the LogiBLOX tool, in a text editor, and examine it. This file is a VHDL simulation model for the LogiBLOX counter called "tenths."

   Close the file after examining it.

4. Run the functional simulation by typing the following at the UNIX prompt.

   `./func.sim`

   This starts the functional simulation process.

   The functional simulation process ends with errors because the VHDL code contains an instantiated component (OSC4) that does not have an underlying RTL behavioral description. You must create an RTL model for functional simulation. For more information about OSC4, refer to the *Libraries Guide* and the *Databook*.

   The UNISIM libraries allow functional simulation of library cells instantiated from the XSI libraries. The VHDL code that instantiates any XSI library cell must contain the following two lines.

   ```
   library UNISIM;
   use UNISIM.all;
   ```

   In general, place these two lines only in the files that contain instantiated XSI library cells (for example, FDCE, RAM32X1S, and BUFG), but you can also place the above two lines in every VHDL file in your design.

5. Open a text editor and correct the stopwatch.vhd file as shown in the following example.

```
library IEEE;
use IEEE.std_logic_1164.all;
library UNISIM;
use UNISIM.all;

entity stopwatch is

    port (    RESET : in STD_LOGIC;
   STRTSTOP : in STD_LOGIC;
  TENTHSOUT : out STD_LOGIC_VECTOR(9 downto 0);
    ONESOUT : out STD_LOGIC_VECTOR(6 downto 0);
    TENSOUT : out STD_LOGIC_VECTOR(6 downto 0)
 );
end stopwatch;

architecture inside of stopwatch is

component OSC4
    port (F500K : out STD_LOGIC);
end component;

component BUFG
    port (I : in STD_LOGIC;
  O : out STD_LOGIC);
end component;

component stmchine
    port (     CLK : in STD_LOGIC;
```

```
     RESET : in STD_LOGIC;
  STRTSTOP : in STD_LOGIC;
     CLKEN : out STD_LOGIC;
       RST : out STD_LOGIC
 );
end component;

component tenths
    port (     CLOCK : in STD_LOGIC;
      CLK_EN : in STD_LOGIC;
  ASYNC_CTRL : in STD_LOGIC;
    TERM_CNT : out STD_LOGIC;
       Q_OUT : out STD_LOGIC_VECTOR(9 downto 0));
end component;

component cnt60
    port (    CE : in STD_LOGIC;
     CLK : in STD_LOGIC;
     CLR : in STD_LOGIC;
  LSBSEC : out STD_LOGIC_VECTOR(3 downto 0);
  MSBSEC : out STD_LOGIC_VECTOR(3 downto 0));
end component;

component hex2led
    port (HEX : in STD_LOGIC_VECTOR(3 downto 0);
  LED : out STD_LOGIC_VECTOR(6 downto 0));
end component;

signal strtstopinv : STD_LOGIC;
signal oscout : STD_LOGIC;
```

```
signal clkint : STD_LOGIC;
signal clkenable : STD_LOGIC;
signal rstint : STD_LOGIC;
signal xcountout : STD_LOGIC_VECTOR(9 downto 0);
signal xtermcnt : STD_LOGIC;
signal cnt60enable : STD_LOGIC;
signal lsbcnt : STD_LOGIC_VECTOR(3 downto 0);
signal msbcnt : STD_LOGIC_VECTOR(3 downto 0);

begin

OSCILLATOR:OSC4 port map(F500K=>oscout);

CLOCKBUF:BUFG port map(I=>oscout,O=>clkint);

MACHINE:stmchine port map(CLK=>clkint,
                  RESET=>RESET,
                  STRTSTOP=>strtstopinv,
                  CLKEN=>clkenable,
                  RST=>rstint
 );

XCOUNTER:tenths port map(CLOCK=>clkint,
                 CLK_EN=>clkenable,
                 ASYNC_CTRL=>rstint,
                 TERM_CNT=>xtermcnt,
                 Q_OUT=>xcountout
                        );

sixty: cnt60 port map(CE=>cnt60enable,
```

```
              CLK=>clkint,
              CLR=>rstint,
              LSBSEC=>lsbcnt,
              MSBSEC=>msbcnt
                     );

lsbled:hex2led port map(HEX=>lsbcnt,
LED=>ONESOUT
                       );

msbled:hex2led port map(HEX=>msbcnt,
LED=>TENSOUT
                       );
cnt60enable<=xtermcnt and clkenable;
TENTHSOUT<=not(xcountout);
strtstopinv<=not(STRTSTOP);

end inside;
```

6. Recompile the func.sim file after you have corrected the stopwatch.vhd file. Simulation starts and displays the waveform viewer; the UNIX shell displays the vhdlsim prompt.

Synthesizing Your Design

You conduct functional simulation to make sure that the desired RTL behavior implements. After confirming the desired RTL behavior, you can synthesize the design.To synthesize a design with FPGA Compiler, you need to create a compile script. A default compile script is provided for your modification in the A1.5i software.

In this section of the tutorial, you synthesize the design and create a place and routed NCD file using the following instructions. After creating the place and routed NCD file, you can optionally proceed to create a BIT file for downloading to the demo board, using bitgen and promgne, or the Hardware Debugger.

1. Copy the file $XILINX/template.fpga.script.4kex into your /home/user/tutorial directory, which contains the .synopsys_dc.setup and .synopsys_vss.setup files you created earlier.
2. Rename the file template.synopsys.dc.setup.4kex to run.script.

   The file template.synopsys_dc.setup.4kex is a template for a compile script you can use for XC4000 families. Spartan can use this file as a template for synthesizing with FPGA Express.
3. Open the file run.script in a text editor. The unmodified file run.script compiles only one VHDL file and you need to comment out several lines not relevant to the synthesis of this particular design.

   The unmodified run.script file appears in the following example (using the file template.synopsys_dc.setup.4kex).

```
/* ================================================*/
/*    Sample Script for Synopsys to Xilinx Using     */
/*                  FPGA Compiler                    */
/*                                                   */
/*  Targets the Xilinx XC4028EX-3 and assumes a VHDL */
/*         source file by way of an example.         */
/*                                                   */
/*   For general use with XC4000E/EX architectures.  */
/*      Not suitable for use with XC3000A/XC5200     */
/*                  architectures.                   */
/* ================================================*/

/* ================================================ */
/* Set the name of the design's top-level module.    */
/* (Makes the script more readable and portable.)    */
/* Also set some useful variables to record the      */
/* designer and company name.                        */
/* ================================================ */
```

```
   TOP = <design_name>
                        /* ========================== */
                        /* Note: Assumes design file- */
                        /* name and entity name are   */
                        /* the same (minus extension) */
                        /* ========================== */
   designer = “XSI Team”
   company  = “Xilinx, Inc”
   part     = “4028expg299-3”
/* ================================================ */
/* Analyze and Elaborate the design file and specify */
/* the design file format.     */
/* ================================================ */
   analyze -format vhdl TOP + “.vhd”
                      /* ============================ */
                      /* You must analyze lower-level */
                      /* hierarchy modules here       */
                      /* ============================ */
   elaborate TOP
/* ================================================ */
/* Set the current design to the top level.         */
/* ================================================ */
   current_design TOP
/* ================================================ */
/* Set the synthesis design constraints.            */
/* ================================================ */
   remove_constraint -all
   /* Some example constraints */
   create_clock <clock_port_name> -period 50
   set_input_delay 5 -clock <clock_port_name> \
```

```
     { <a_list_of_input_ports> }
   set_output_delay 5 -clock <clock_port_name> \
     { <a_list_of_output_ports> }
   set_max_delay 100 -from <source> -to <destination>
   set_false_path -from <source> -to <destination>
/* ================================================ */
/* Indicate those ports on the top-level module that */
/* should become chip-level I/O pads. Assign any I/O */
/* attributes or parameters and perform the I/O      */
/* synthesis.                                       */
/* ================================================ */
   set_port_is_pad “*”
   /* Some example I/O parameters */
   set_pad_type -pullup <port_name>
   set_pad_type -no_clock all_inputs()
   set_pad_type -clock <clock_port_name>
   set_pad_type -exact BUFGS_F <hi_fanout_port_name>
   set_pad_type -slewrate HIGH all_outputs()
                   /* ============================== */
                   /* Note: Synopsys slew-control=  */
                   /* HIGH is the same as Xilinx’s  */
                   /* slewrate=SLOW. Synopsys slew- */
                   /* control=LOW is same as Xilinx */
                   /* slewrate=FAST.                */
                   /* ============================== */
   insert_pads
/* ================================================ */
/* Synthesize and optimize the design               */
/* ================================================ */
   compile -boundary_optimization
```

```
/* ================================================ */
/* Write the design report files.                   */
/* ================================================ */
   report_fpga > TOP + ".fpga"
   report_timing > TOP + ".timing"
/* ================================================ */
/* Write out the design to a DB file. (Post compile) */
/* ================================================ */
   write -format db -hierarchy -output TOP + "_compiled.db"
/* ================================================ */
/* Replace CLBs and IOBs with gates.                */
/* ================================================ */
   replace_fpga
/* ================================================ */
/* Set the part type for the output netlist.     */
/* ================================================ */
   set_attribute TOP "part" -type string part
/* ================================================ */
/* Optional attribute to remove the FPGA Compiler's  */
/* mapping structures from the design. This permits  */
/* The Xilinx design implementation tools to map the */
/* design instead.                                  */
/* ================================================ */
/* set_attribute find(design,"*") "xnfout_write_map_symbols" \
     -type boolean FALSE  */
/* ================================================ */
/* Add any I/O constraints to the design.           */
/* ================================================ */
   set_attribute <port_name> "pad_location" \
     -type string "<pad_location>"
```

```
/* ================================================ */
/* Save design in XNF format as <design>.sxnf       */
/* ================================================ */
   ungroup -all -flatten
   write -format xnf -hierarchy -output TOP + ".sxnf"
/* ================================================ */
/* Write out the design to a DB. (Post replace_fpga) */
/* ================================================ */
   write -format db -hierarchy -output TOP + ".db"
/* ================================================ */
/* Write-out the timing constraints that were       */
/* applied earlier. (Note that any design hierarchy  */
/* needs to be flattened before the constraints are  */
/* written-out.)                                    */
/* ================================================ */
   write_script > TOP + ".dc"
/* ================================================ */
/* Call the Synopsys-to-Xilinx constraints translator*/
/* utility DC2NCF to convert the Synopsys constraints*/
/* to a Xilinx NCF file. You may like to view       */
/* dc2ncf.log to review the translation process.    */
/* ================================================ */
   sh dc2ncf TOP + ".dc"
/* ================================================ */
/* Exit the Compiler.                               */
/* ================================================ */
   exit
/* ================================================ */
/* Now run the Xilinx design implementation tools.  */
/* ================================================ */
```

4. Modify the run.script file for the files in this tutorial as shown in the following example.

   Before using this script, notice the following items.

   - The design compiles from the bottom up.
   - The VHDL file for the LogiBLOX counter tenths.vhd is not compiled. The VHDL file from the LogiBLOX tools is a simulation model.
   - The proper type of output for place and route in A1.5i is an SXNF file when compiling a XC4000 design in FPGA Compiler.

   Note the Dont_touch commands added to the script. Whenever you instantiate a component from the XSI synthesis library, you must place a Dont_touch on the instance to prevent Synopsys from deleting or modifying the library cell.

```
/* ================================================*/
/*    Sample Script for Synopsys to Xilinx Using     */
/*                  FPGA Compiler                    */
/*                                                   */
/*  Targets the Xilinx XC4028EX-3 and assumes a VHDL */
/*         source file by way of an example.         */
/*                                                   */
/*   For general use with XC4000E/EX architectures.  */
/*      Not suitable for use with XC3000A/XC5200     */
/*                  architectures.                   */
/* ================================================*/


/* ================================================ */
/* Set the name of the design's top-level module.    */
/* (Makes the script more readable and portable.)    */
/* Also set some useful variables to record the      */
/* designer and company name.                        */
/* ================================================ */
```

```
   TOP = stopwatch
                        /* ========================== */
                        /* Note: Assumes design file- */
                        /* name and entity name are   */
                        /* the same (minus extension) */
                        /* ========================== */
   designer = “XSI Team”
   company  = “Xilinx, Inc”
   part     = “s05pc84-3”
/* ================================================ */
/* Analyze and Elaborate the design file and specify */
/* the design file format.     */
/* ================================================ */
analyze -format vhdl “smallcntr.vhd”
analyze -format vhdl “cnt60.vhd”
analyze -format vhdl “hex2led.vhd”
analyze -format vhdl “stmchine.vhd”
   analyze -format vhdl TOP + “.vhd”
                      /* ============================ */
                      /* You must analyze lower-level */
                      /* hierarchy modules here       */
                      /* ============================ */
   elaborate TOP
   set_dont_touch “OSCILLATOR”
   uniquify
/* ================================================ */
/* Set the current design to the top level.         */
/* ================================================ */
   current_design TOP
/* ================================================ */
```

```
/* Set the synthesis design constraints.             */
/* ================================================= */
   remove_constraint -all
   /* Some example constraints */
/*   create_clock <clock_port_name> -period 50
   set_input_delay 5 -clock <clock_port_name> \
     { <a_list_of_input_ports> }
   set_output_delay 5 -clock <clock_port_name> \
     { <a_list_of_output_ports> }
   set_max_delay 100 -from <source> -to <destination>
   set_false_path -from <source> -to <destination> */
/* ================================================= */
/* Indicate those ports on the top-level module that */
/* should become chip-level I/O pads. Assign any I/O */
/* attributes or parameters and perform the I/O      */
/* synthesis.                                        */
/* ================================================= */
   set_port_is_pad “*”
   /* Some example I/O parameters */
/*   set_pad_type -pullup <port_name>
   set_pad_type -no_clock all_inputs()
   set_pad_type -clock <clock_port_name>
   set_pad_type -exact BUFGS_F <hi_fanout_port_name>
   set_pad_type -slewrate HIGH all_outputs() */
                  /* ============================== */
                  /* Note: Synopsys slew-control=  */
                  /* HIGH is the same as Xilinx’s  */
                  /* slewrate=SLOW. Synopsys slew- */
                  /* control=LOW is same as Xilinx */
                  /* slewrate=FAST.                */
```

```
                    /* ============================== */
   insert_pads
/* ================================================= */
/* Synthesize and optimize the design                */
/* ================================================= */

   compile -boundary_optimization
/* ================================================= */
/* Write the design report files.                    */
/* ================================================= */
/*   report_fpga > TOP + ".fpga"
   report_timing > TOP + ".timing" */
/* ================================================= */
/* Write out the design to a DB file. (Post compile) */
/* ================================================= */
   write -format db -hierarchy -output TOP + "_compiled.db"
/* ================================================= */
/* Replace CLBs and IOBs with gates.                 */
/* ================================================= */
   replace_fpga
/* ================================================= */
/* Set the part type for the output netlist.     */
/* ================================================= */
   set_attribute TOP "part" -type string part
/* ================================================= */
/* Optional attribute to remove the FPGA Compiler's  */
/* mapping structures from the design. This permits  */
/* The Xilinx design implementation tools to map the */
/* design instead.                                   */
/* ================================================= */
```

```
/* set_attribute find(design,"*") "xnfout_write_map_symbols" \
     -type boolean FALSE  */
/* ================================================ */
/* Add any I/O constraints to the design.           */
/* ================================================ */
/*   set_attribute <port_name> "pad_location" \
     -type string "<pad_location>" */
/* ================================================ */
/* Save design in XNF format as <design>.sxnf       */
/* ================================================ */
   ungroup -all -flatten
   write -format xnf -hierarchy -output TOP + ".sxnf"
/* ================================================ */
/* Write out the design to a DB. (Post replace_fpga) */
/* ================================================ */
   write -format db -hierarchy -output TOP + ".db"
/* ================================================ */
/* Write-out the timing constraints that were       */
/* applied earlier. (Note that any design hierarchy  */
/* needs to be flattened before the constraints are  */
/* written-out.)                                    */
/* ================================================ */
/*   write_script > TOP + ".dc" */
/* ================================================ */
/* Call the Synopsys-to-Xilinx constraints translator*/
/* utility DC2NCF to convert the Synopsys constraints*/
/* to a Xilinx NCF file. You may like to view       */
/* dc2ncf.log to review the translation process.    */
/* ================================================ */
/*   sh dc2ncf TOP + ".dc" */
```

```
/* ================================================ */
/* Exit the Compiler.                               */
/* ================================================ */
/*   exit */
/* ================================================ */
/* Now run the Xilinx design implementation tools.   */
/* ================================================ */
```

5. Synthesize the design, starting Design Analyzer by typing the following command in the directory that contains the .synopsys_dc.setup file for this design.

   design_analyzer &

   This launches the Design Analyzer GUI.

6. When the GUI appears, run the script run.script by selecting Execute Script from the Setup menu. A pop-up window appears where you can select the script 'run.script' to run.

   If the script runs successfully, the script stops and creates a SXNF file. If an error occurs, check the following.

   - Make sure you set up the .synopsys_dc.setup file correctly.
   - Make sure you compiled the XDW synthesis libraries for the version of Synopsys you use.
   - Make sure that the paths referenced in the .synopsys_dc.setup file exist. Refer to the "Using Common Setup Procedures" section for more information.
   - Make sure you run the command 'design_analyzer &' in the directory that contains the .synopsys_dc.setup file when you start Design Analyzer.

7. Take the SXNF file produced by Design Analyzer and place and route the design for timing simulation using the following script.

   ```
   #!/bin/csh –f

   ngdbuild –p 4003EPC84-4 stopwatch.sxnf

   map stopwatch.ngd

   par stopwatch.ncd stopwatch_r.ncd
   ```

```
ngdanno stopwatch_r.ncd

ngd2vhdl stopwatch_r.nga
```

Optionally, you can place and route the SXNF files using the A1.5i GUI. Please refer to the *Quick Start Guide Tutorial* for more information for using the GUI for place and route.

Conducting Timing Simulation

To perform timing simulation, you must create an SDF file and structural VHDL file using NGD2VHDL, along with a testbench. The script file timing.sim performs the timing simulation.

Run timing.sim in the directory that contains the .synopsys_vss.setup file you created.

Before conducting timing simulation, open the script file timing.sim in a text editor and notice that vhdlan uses the –i option. Always use vhdlan with the –i option. By default, vhdlan uses the –c option, which only works if your system is setup to use a certain type of C compiler. If you want to use the –c option with vhdlan, please refer to the Synopsys web site for the proper setup.

Because you perform a timing simulation, when invoking vhdldbx or vhdlsim, specify the –sdf_top option first.

Use the file timing.sim shown in the following example.

```
#!/bin/csh -f

rm –r WORK

mkdir WORK

vhdlan –i stopwatch_r.vhd

vhdlan –i testbencht.vhd

vhdlsim –sdf_top /testbenchf/uut \
–sdf stopwatch_r.sdf –e commandt.txt overall
```

Close the timing.sim file in the text editor. Run the timing simulation, which produces a waveform view like the functional simulation, by typing the following at the UNIX prompt.

```
./timing.sim
```

If you get error or warning messages about “Bad Regions” or undefined libraries when the script runs, make sure you set up the simula-

tion libraries for A1.5i XSI VSS. Make sure the paths in the .synopsys_vss.setup file point to paths which exist in your setup. For further information on setup, refer to the “Using Common Setup Procedures” section.

Spartan/XC4000 VHDL Alliance FPGA Express v2.1 Tutorial

This tutorial familiarizes you with the A1.5i with Alliance FPGA Express v2.1 flow, presenting common tasks such as locking pins and setting slew rate.

Getting Ready for This Tutorial

To use this tutorial, you need the software described in the “Using Common Setup Procedures” section. By default, this tutorial uses the XC4000E family as a target design, but you can use other FPGA families supported under A1.5i XSI as well. If you want to target a device other than the XC4000E, reference the directories that apply to that family (for example, the XDW libraries have separate directories for XC4000XL, Spartan, and Virtex). Use synlibs with the exact die-speed target desired.

Setting Up for the Spartan/XC4000 VHDL Alliance FPGA Express v2.1 Tutorial

To use this tutorial, ensure installation of your Xilinx and Synopsys software and know where the software resides on your system. Download the files for this tutorial, spartanxsivhdl.tar.Z, from the Xilinx web site. Untar and uncompress this file in a directory of your choosing, but this tutorial assumes that you place the files in the /home/user/tutorial directory.

You must set up the VSS simulation libraries for this tutorial. The A1.5i simulation libraries come in two parts, a functional simulation part and a timing simulation part. The A1.5i XSI VHDL functional simulation libraries are called UNISIM. The A1.5i XSI VHDL timing simulation libraries are called SIMPRIM libraries. The SIMPRIM library is a VITAL simulation library.

Additionally, there is another functional simulation library called LOGIBLOX, used for simulating simulation models created by Logi-

BLOX. You use the LOGIBLOX library when your design contains an instantiated LOGIBLOX component such as a RAM.

For this tutorial, compile only the libraries related to simulating, the LogiBLOX simulation libraries, UNISIM simulation libraries, and SIMPRIM libraries. All of these libraries have compile scripts which let you compile these libraries in the $XILINX area, if you have write permissions to the $XILINX area.

If you do not have write permissions, make a local copy of the $XILINX/synopsys/libraries directory and use the following instructions. But instead of changing directories to $XILINX/synopsys/libraries, change directories to your local copy of that area where appropriate.

To compile the libraries, use the following steps.

1. Change directories to the $XILINX/synopsys/libraries/dw/src/xc4000e directory.
2. Type the following command at the UNIX prompt.

   `dc_shell -f install_dw.dc`
3. Change directories to the $XILINX/synopsys/libraries/sim/src/logiblox directory.
4. (4) Type the following command at the UNIX prompt.

   `./analyze.csh`
5. Change directories to the $XILINX/synopsys/libraries/sim/src/simprims directory.
6. Type the following command at the UNIX prompt.

   `./analyze.csh`
7. Change directories to the $XILINX/synopsys/libraries/sim/src/unisims directory
8. Type the following command at the UNIX prompt.

   `./analyze.csh`
9. Change directories to the $XILINX/synopsys/libraries/sim/src/xdw directory.
10. Type the following command at the UNIX prompt.

    `./analyze.csh`

After compiling the libraries, you must create a directory where you run the tutorial, as described in the "Using Common Setup Procedures" section. This directory contains the tutorial HDL files. If you copied the $XILINX/synopsys/libraries directory locally, then modify the paths for the various simulation libraries as you use the instructions in this tutorial.

The functional simulation flow with FPGA Express and VSS has four possible flows, including the traditional pre-synthesis flow known as functional simulation. This tutorial uses pre-synthesis simulation as the functional simulation flow.

Conducting Functional Simulation

In this part of the tutorial you compile the design files and testbench, and run the VSS simulation tool. This tutorial assumes that you uncompressed, untarred, and placed your setup files in /home/user/tutorial. If you used a different location, replace your path appropriately in the instructions in this tutorial.

A feature of the A1.5i XSI VSS functional simulation flow allows you to simulate instantiated XSI cells such as FDCE and OSC4. Additionally, by using the UNISIM simulation libraries, you can simulate and implement the GSR without impact to the design or testbench.

Use the following instructions to conduct functional simulation.

1. Change directories to the /home/user/tutorial directory. and ensure all the appropriate tutorial files listed in the "Checking the Common Tutorial Files" section reside there.
2. You perform functional simulation by running the script func.script. But before running the script, open the file func.script in a text editor and note the following items in this file.
   - Simulation files read in from the bottom up.
   - The testbench reads in last.
   - vhdlan uses the –i option. Always use vhdlan with the –i option. By default, vhdlan uses the –c option, which only works if your system uses a certain type of C compiler. If you want to use the –c option with vhdlan, refer to the Synopsys web site for the proper setup.
   - The contents of the func.sim file.

```
#!/bin/csh -f
rm –r WORK
mkdir WORK
vhdlan –i tenths.vhd
vhdlan –i smallcntr.vhd
vhdlan –i cnt60.vhd
vhdlan –i hex2led.vhd
vhdlan –i stmchine.vhd
vhdlan –i stopwatch.vhd
vhdlan –i testbenchf.vhd
vhdlsim –e commandf.txt overall
```

Close the file after examining it.

3. Also open the file tenths.vhd, created by the LogiBLOX tool, in a text editor, and examine it. This file is a VHDL simulation model for the LogiBLOX counter called “tenths.”

   Close the file after examining it.

4. Run the functional simulation by typing the following at the UNIX prompt.

   `./func.sim`

   This starts the functional simulation process.

   You receive error messages because the VHDL code contains an instantiated component (OSC4) that does not have an underlying RTL behavioral description. You must make an RTL model for functional simulation. For more information on the OSC4, refer to the *Libraries Guide* and the *Databook*.

   The UNISIM libraries allow functional simulation of library cells instantiated from the XSI libraries. The VHDL code that instantiates any XSI library cell must contain the following two lines.

   ```
   library UNISIM;
   use UNISIM.all;
   ```

   In general, you need to add these lines only in the files that contain instantiated XSI library cells (such as FDCE, RAM32X1S

and BUFG, for example). You can also place the above two lines in every VHDL file in your design.

Add the two lines of code in the stopwatch.vhd file, as shown in the following example.

```
library IEEE;
use IEEE.std_logic_1164.all;
library UNISIM;
use UNISIM.all;

entity stopwatch is

    port (    RESET : in STD_LOGIC;
   STRTSTOP : in STD_LOGIC;
  TENTHSOUT : out STD_LOGIC_VECTOR(9 downto 0);
    ONESOUT : out STD_LOGIC_VECTOR(6 downto 0);
    TENSOUT : out STD_LOGIC_VECTOR(6 downto 0)
 );
end stopwatch;

architecture inside of stopwatch is

component OSC4
    port (F500K : out STD_LOGIC);
end component;

component BUFG
    port (I : in STD_LOGIC;
  O : out STD_LOGIC);
end component;
```

```
component stmchine
    port (     CLK : in STD_LOGIC;
     RESET : in STD_LOGIC;
  STRTSTOP : in STD_LOGIC;
     CLKEN : out STD_LOGIC;
       RST : out STD_LOGIC
 );
end component;

component tenths
    port (     CLOCK : in STD_LOGIC;
      CLK_EN : in STD_LOGIC;
  ASYNC_CTRL : in STD_LOGIC;
    TERM_CNT : out STD_LOGIC;
       Q_OUT : out STD_LOGIC_VECTOR(9 downto 0));
end component;

component cnt60
    port (    CE : in STD_LOGIC;
     CLK : in STD_LOGIC;
     CLR : in STD_LOGIC;
  LSBSEC : out STD_LOGIC_VECTOR(3 downto 0);
  MSBSEC : out STD_LOGIC_VECTOR(3 downto 0));
end component;

component hex2led
    port (HEX : in STD_LOGIC_VECTOR(3 downto 0);
  LED : out STD_LOGIC_VECTOR(6 downto 0));
end component;
```

```
signal strtstopinv : STD_LOGIC;
signal oscout : STD_LOGIC;
signal clkint : STD_LOGIC;
signal clkenable : STD_LOGIC;
signal rstint : STD_LOGIC;
signal xcountout : STD_LOGIC_VECTOR(9 downto 0);
signal xtermcnt : STD_LOGIC;
signal cnt60enable : STD_LOGIC;
signal lsbcnt : STD_LOGIC_VECTOR(3 downto 0);
signal msbcnt : STD_LOGIC_VECTOR(3 downto 0);

begin

OSCILLATOR:OSC4 port map(F500K=>oscout);

CLOCKBUF:BUFG port map(I=>oscout,O=>clkint);

MACHINE:stmchine port map(CLK=>clkint,
                  RESET=>RESET,
                  STRTSTOP=>strtstopinv,
                  CLKEN=>clkenable,
                  RST=>rstint
 );

XCOUNTER:tenths port map(CLOCK=>clkint,
                 CLK_EN=>clkenable,
                 ASYNC_CTRL=>rstint,
                 TERM_CNT=>xtermcnt,
                 Q_OUT=>xcountout
                        );
```

```
sixty: cnt60 port map(CE=>cnt60enable,
              CLK=>clkint,
              CLR=>rstint,
              LSBSEC=>lsbcnt,
              MSBSEC=>msbcnt
                     );

lsbled:hex2led port map(HEX=>lsbcnt,
LED=>ONESOUT
                       );

msbled:hex2led port map(HEX=>msbcnt,
LED=>TENSOUT
                       );
cnt60enable<=xtermcnt and clkenable;
TENTHSOUT<=not(xcountout);
strtstopinv<=not(STRTSTOP);

end inside;
```

5. Restart functional simulation after you correct the stopwatch.vhd file. The simulation starts and displays the waveform viewer. The UNIX shell displays the vhdlsim prompt.

Synthesizing Your Design

In this section of the tutorial, you synthesize the design and create a place and routed NCD file. After creating the place and routed NCD file, you can optionally create a BIT file for downloading to the demo board, using bitgen and promgne, or the Hardware Debugger. For more information on using the FPGA Express GUI, please refer to the FPGA Express on-line help.

Create an FPGA Express Project, enter source files, and Specify a Target device(4003EPC84-3), as shown in the following illustration.

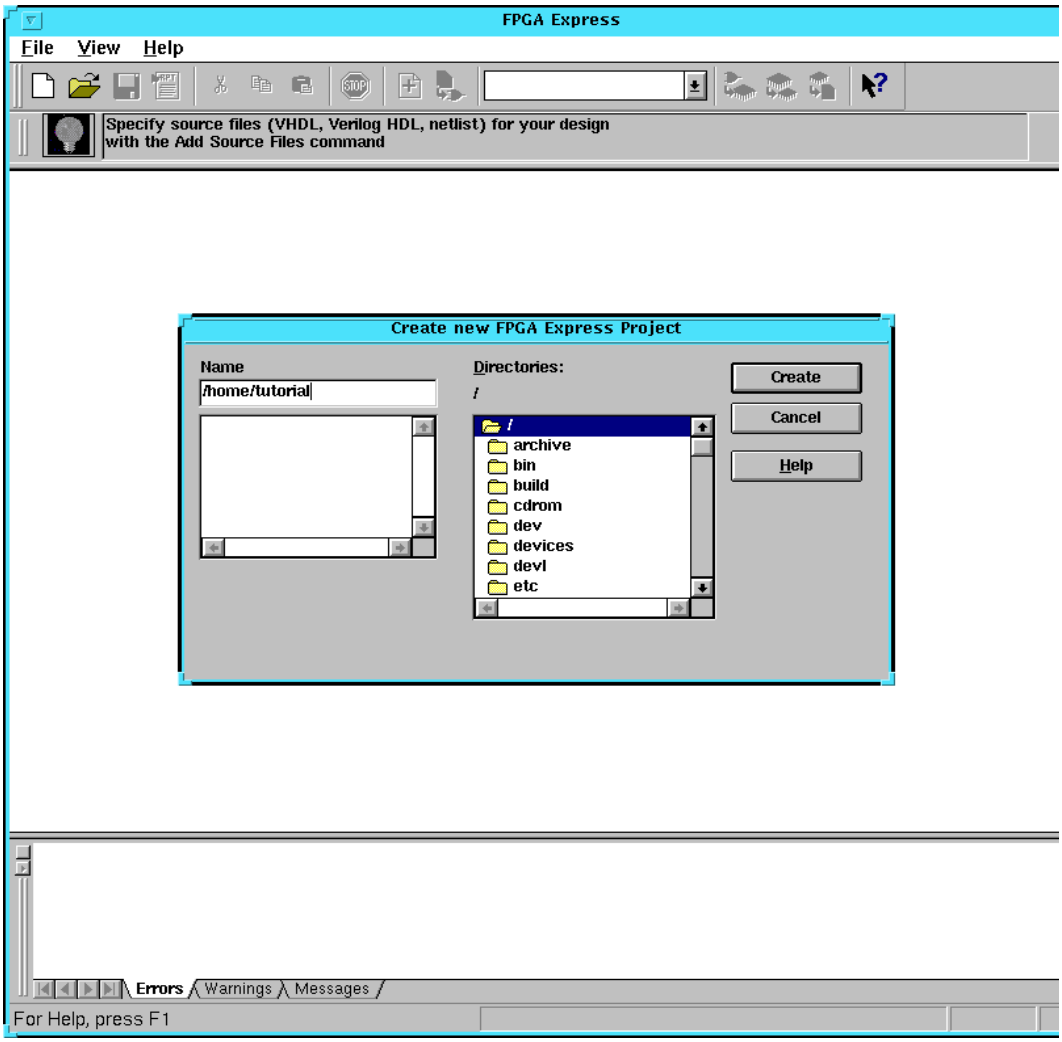

Figure 1-6 Creating an FPGA Express Project

Add the design files to the project, as shown in the next figure.

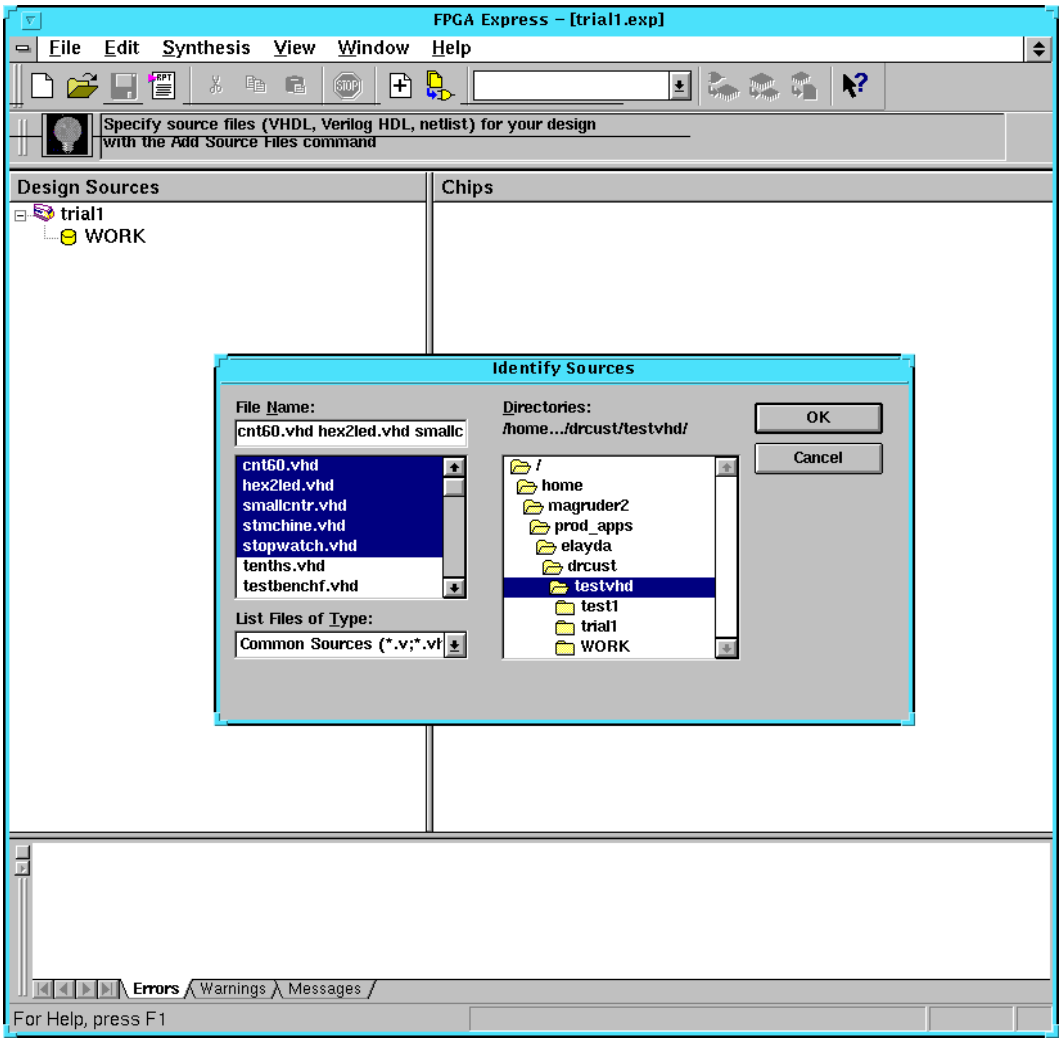

Figure 1-7 Adding Design Files to an FPGA Express Project

Select the top level entity and select a target device, as shown in the following figure.

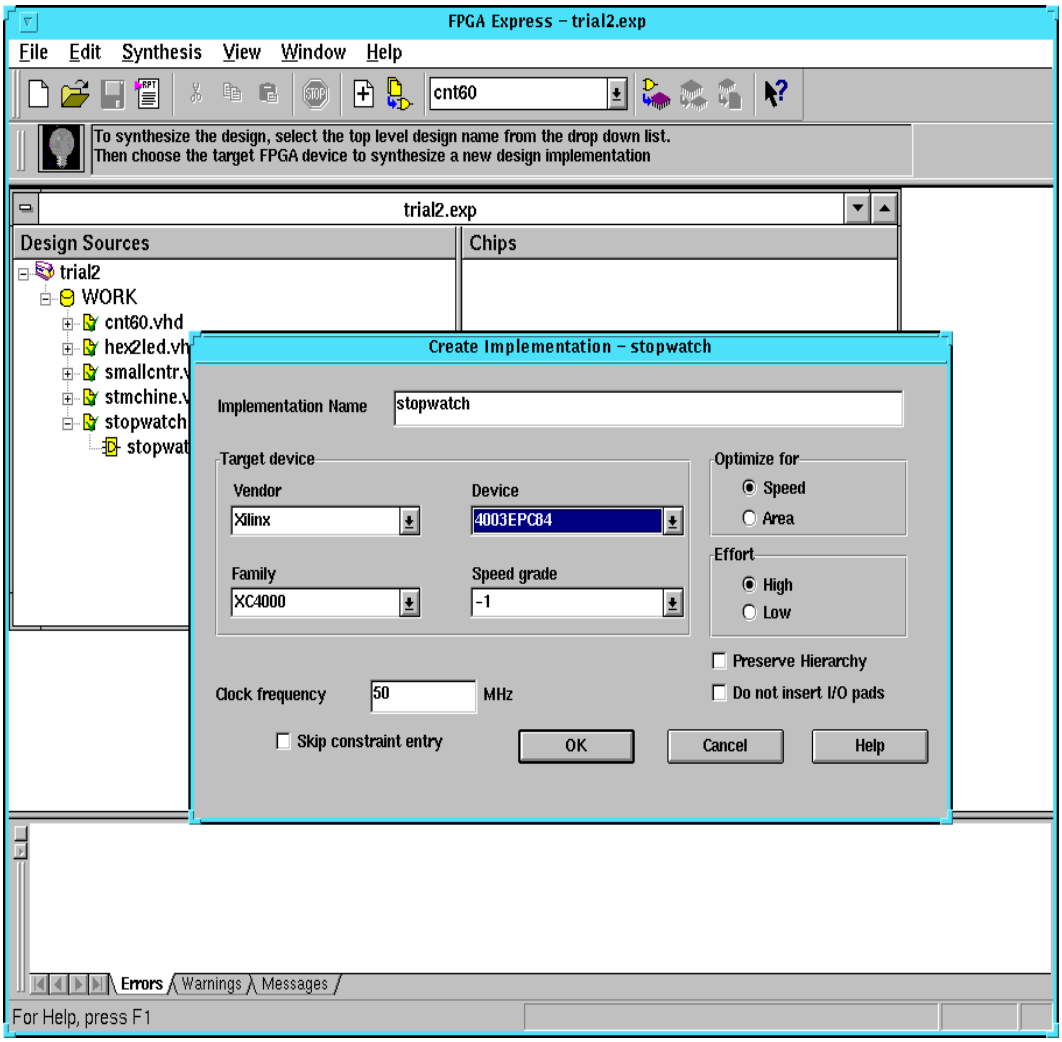

Figure 1-8 Selecting a Top Entity and Target Device

An implementation appears in the right-hand window. Select the implementation and press the Optimize button on the tool bar as shown in the next illustration.

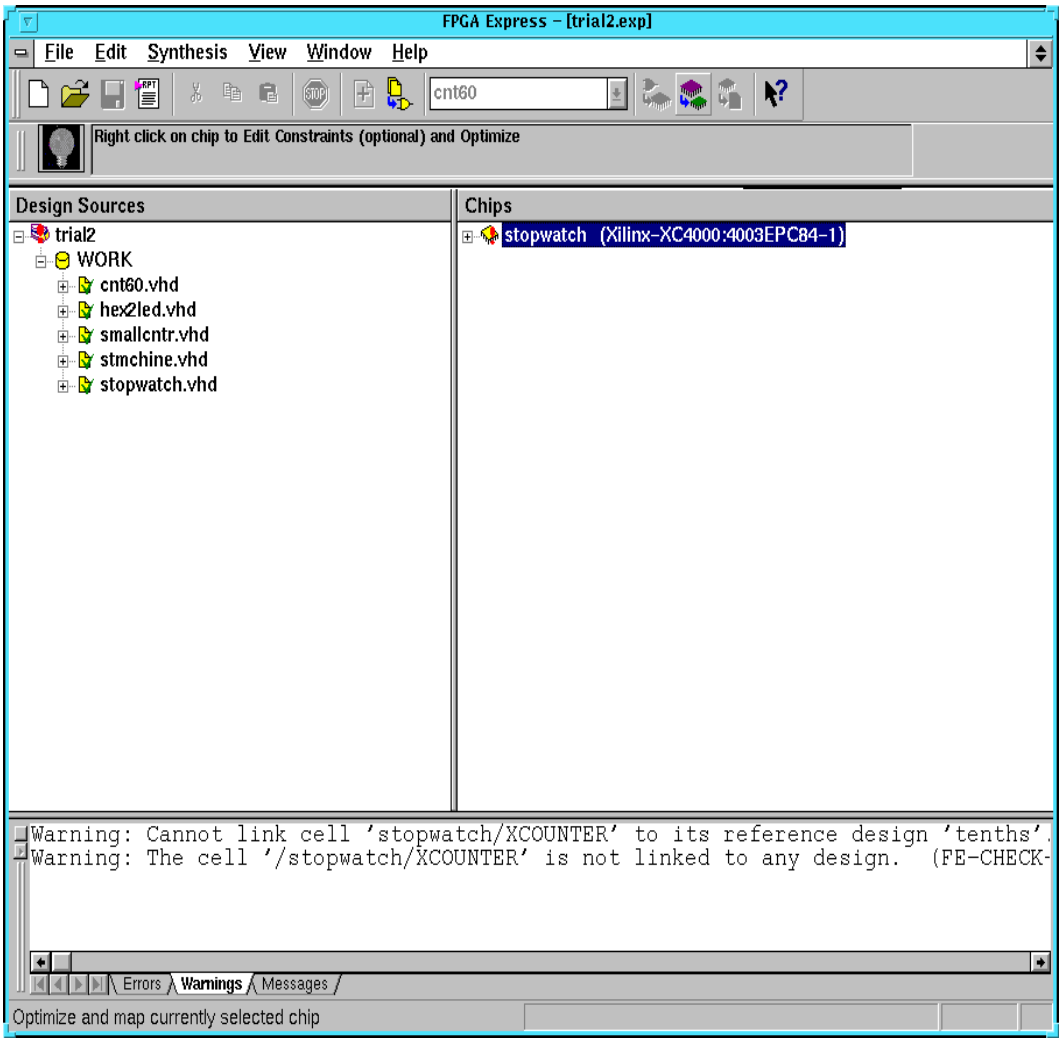

Figure 1-9 Optimizing an Implementation

Select the optimized design and write out the netlist by pressing the Export Netlist button on the toolbar, as shown in the following illustration.

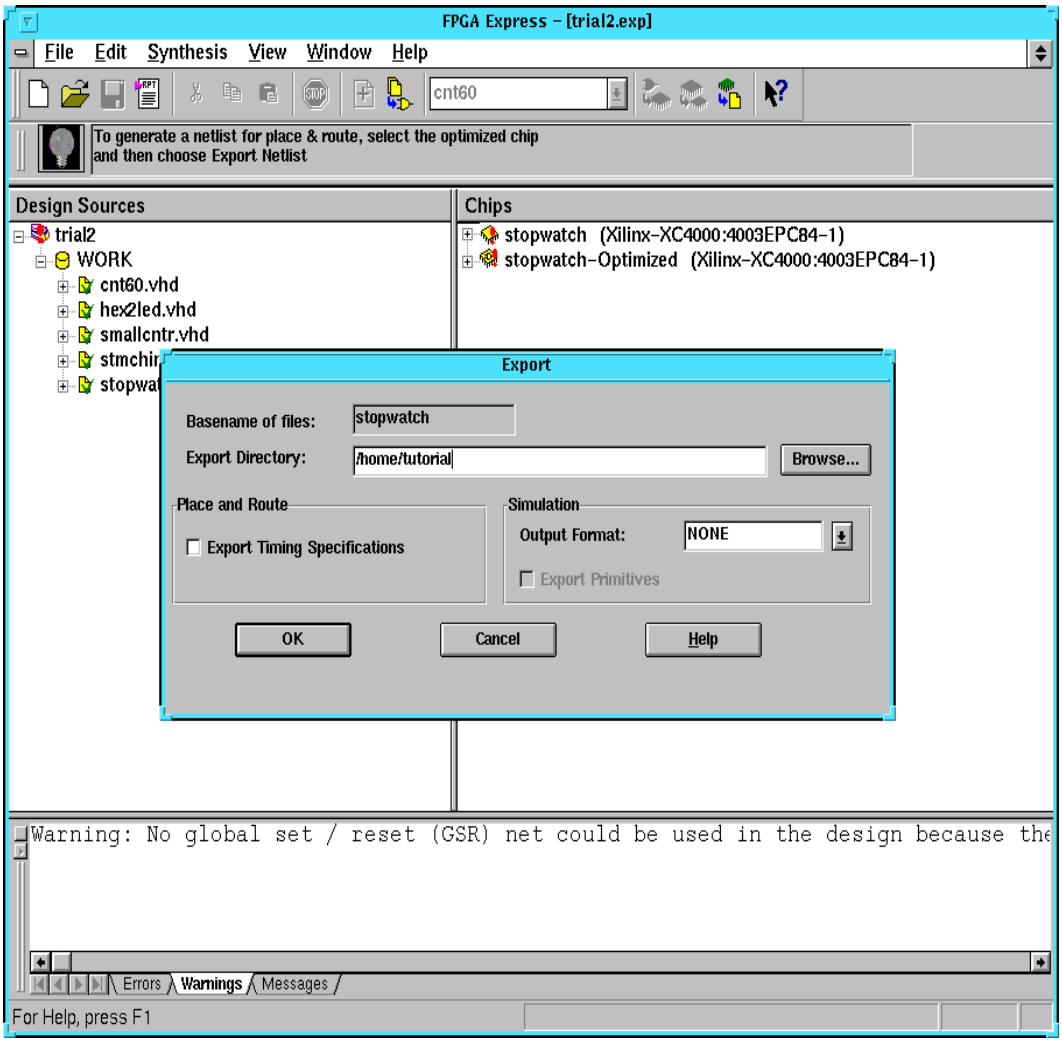

Figure 1-10 Writing Out the Netlist

Take the XNF file produced by FPGA Express and place and route the design for timing simulation using the following script.

```
#!/bin/csh –f

ngdbuild –p 4003EPC84-4 stopwatch.xnf

map stopwatch.ngd
```

```
par stopwatch.ncd stopwatch_r.ncd

ngdanno stopwatch_r.ncd

ngd2vhdl stopwatch_r.nga
```

Optionally, you can place and route the XNF file using the A1.5i GUI. Refer to the *Quick Start Guide Tutorial* for more information about using the GUI for place and route.

Conducting Timing Simulation

To perform timing simulation, use NGD2VHDL to create an SDF file and structural VHDL file, along with a testbench. The script file timing.sim performs the timing simulation.

Run timing.sim in the directory that contains the .synopsys_vss.setup file you created.

Open the script file timing.sim in a text editor and note the following.

- vhdlan uses the –i option. Always use vhdlan with the –i option. By default, vhdlan uses the –c option, which only works if your system uses a certain type of C compiler. If you want to use the –c option with vhdlan, please refer to the Synopsys web site for more information.
- Specify the –sdf_top option first when invoking vhdldbx or vhdlsim for a timing simulation.
- The file timing.sim contents

```
#!/bin/csh –f

rm –r WORK

mkdir WORK

vhdlan –i stopwatch_r.vhd

vhdlan –i testbencht.vhd

vhdlsim –sdf_top /testbenchf/uut –sdf \
stopwatch_r.sdf –e commandt.txt overall
```

Close the timing.sim file in the text editor. Run the timing simulation, which produces a waveform view of the functional simulation, by typing the following command at the UNIX prompt.

```
./timing.sim
```

When the script runs, if you get errors and warning messages about "Bad Regions" or undefined libraries, make sure you set up the simulation libraries for A1.5i XSI VSS correctly. Make sure the paths in the .synopsys_vss.setup file point to paths which exist in your setup. For further information on setup, refer to the "Using Common Setup Procedures" section.

Virtex Verilog FPGA Compiler VerilogXL Tutorial

This tutorial familiarizes you with the A1.5i FPGA Compiler / VerilogXL design flow and includes a VHDL design you can optionally download to the demo board. This tutorial presents common tasks such as locking pins and setting slew rate.

Getting Ready for this Tutorial

To use this tutorial, you need the software described in the "Using Common Setup Procedures" section. If you use a version of Synopsys newer than v1997.01, you must follow the additional steps in the "Setting Up for the Spartan / XC4000 Verilog FPGA Compiler Tutorial" section. If you want to target a device other than that specified in the tutorial, reference the directories that apply to that family (for example, the XDW libraries have separate directories for XC4000XL, Spartan, and Virtex). Use synlibs with the exact die-speed target desired.

Setting Up for the Virtex FPGA Compiler VerilogXL Tutorial

To use this tutorial, ensure installation of your Xilinx and Synopsys software and know where the software resides on your system. If you use a version of Synopsys newer than v1997.01, you must re-compile the XSI XDW and simulation libraries. Refer to the setup instructions in the "Setting up for FPGA Compiler" section.

For this tutorial, recompile only the libraries related to synthesizing a Virtex device (if using a version of Synopsys newer than v1997.01), and ensure compilation of the Virtex XDW DesignWare libraries.

All A1.5i XSI of these libraries contain compile scripts which let you compile these libraries in the $XILINX area. However, to use these scripts, you need write permissions to the $XILINX area. If you do not have write permissions, make a local copy of the $XILINX /

synopsys/libraries directory and follow the instructions below, but instead of changing directories to $XILINX/synopsys/libraries, change directories to your local copy of that area.

To compile the Virtex device libraries, complete the following steps.

1. Change directories to the $XILINX/synopsys/libraries/dw/src/virtex directory.
2. Type the following command at the UNIX prompt.

   ```
   dc_shell –f install_dw.dc
   ```

   Make sure you have completed the instructions in the "Using Common Setup Procedures" section before continuing.
3. Because you synthesize a Virtex device in this tutorial, use synlibs to add the correct information into the .synopsys_dc.setup file. Type the following command at the UNIX prompt.

   ```
   synlibs –fc  v50pq240-4 >> .synopsys_dc.setup
   ```

   This appends the output of synlibs into the .synopsys_dc.setup file. If you compiled the Spartan XDW libraries in the $XILINX tree, you can proceed to checking the .synopsys_vss.setup file. If you compiled the XDW libraries in the $XILINX area, you need not modification of the define_design_lib line in the .synopsys_dc.setup file. If you copied the $XILINX/synopsys/libraries directory locally, you must change the path in the .synopsys_dc.setup file to reflect the copied directory path. For example, if you copied the $XILINX/libraries to /home/data, edit the `define_design_lib' setting made by synlibs for the v50-4 above to reflect the /home/data location.

   ```
   define_design_lib xdw_virtex /home/data/libraries \
   /dw/lib/virtex
   ```
4. In the directory where you created your .synopsys_dc.setup, and where you uncompressed and untarred the file spartanx-sivhdl.tar.Z file, type the following command at the UNIX prompt.

   ```
   ls -l
   ```
5. Make sure that directory contains at a minimum the following items.

   - .synopsys_dc.setup

- WORK

Conducting Functional Simulation

In this section of the tutorial you compile the design files and test-bench, and run the VSS simulation tool. This tutorial assumes that you uncompressed, untarred, and placed your setup files in /home/user/tutorial. If you placed these files elsewhere, replace your path appropriately in the following instructions.

Make sure you follow the directions provided in the "Using Common Setup Procedures" section before conducting functional simulation.

The A1.5i XSI VSS functional simulation flow allows you to simulate instantiated XSI cells such as FDCE and OSC4. Additionally, by using the UNISIM simulation libraries, the you can simulate and implement the GSR without impact to the design or testbench.

Use the following steps to conduct functional simulation.

1. Change directories to the /home/user/tutorial directory.
2. Conduct functional simulation is performed by running the VerilogXL command verilog with the data file filesf.f. Enter the following at the UNIX prompt

   verilog –f filesf.f

   Running this command produces errors, including the following.

```
Compiling source file "testbenchf.v"
Compiling source file "stopwatch.v"
Compiling source file "stmchine.v"
Compiling source file "hex2led.v"
Compiling source file "cnt60.v"
Compiling source file "smallcntr.v"
Compiling source file "tenths.v"

Error!   Module or primitive (BUFG) not defined         [Verilog-MOPND]
        "stopwatch.v", 24: BUFG CLOCKBUF(.I(oscout), .O(
          clkint));
```

3. You get error messages because the tutorial design contains instantiations of LogiBLOX and instantiated XSI synthesis library cells (such as OSC4, FDCE, and BUFG, for example). You need to tell the Verilog simulator where to find the models for these cells by placing the ‘uselib directive in the top-level file of this design, stopwatch.v. Add the following line to the top of your stopwatch.v file.

   ```
   `uselib dir=$XILINX/verilog/src/UNIVIRTEX libext=.v
   ```

4. Replace the $XILINX text with the explicit path in your environment. If you set $XILINX to /home/software/xilinx, put the following ‘uselib line in the top of the stopwatch.v file.

   ```
   `uselib dir=/home/software/xilinx/verilog \
   /src/UNIVIRTEX libext=.v
   ```

5. After making the changes, re-run the simulation command.

   ```
   verilog –f filesf.f
   ```

Synthesizing Your Design

In this section of the tutorial, you synthesize the design and create a place and routed NCD file. After creating the place and routed NCD file, you can optionally create a BIT file for downloading to the demo board, using bitgen and promgne, or the Hardware Debugger.

You conduct functional simulation to make sure that the design implements the desired RTL behavior. To synthesize a design with FPGA Compiler after achieving the desired RTL behavior, you need to create a compile script. This tutorial provides a default compile script you can modify in the A1.5i software. Use the following instructions to copy and modify this compile script.

1. Copy the file $XILINX/template.fpga.script.4kex into your /home/user/tutorial directory, which contains the .synopsys_dc.setup and .synopsys_vss.setup files you created earlier in the “Using Common Setup Procedures” section.
2. Rename the file template.fpga.script.virtex to run.script.
3. Open the file run.script in a text editor.

   The file run.script compiles only one VHDL file. You need to comment out lines not relevant to the synthesis of this particular

design. The following example shows the unmodified run.script (using the file template.fpga.script.virtex).

```
/* ================================================*/
/*    Sample Script for Synopsys to Xilinx Using     */
/*                  FPGA Compiler                    */
/*                                                   */
/*  Targets the Xilinx XCV150PQ240-3 and assumes a   */
/*   VHDL source file by way of an example.          */
/*                                                   */
/*   For general use with VIRTEX architectures.      */
/* ================================================*/

/* ================================================ */
/* Set the name of the design's top-level module.    */
/* (Makes the script more readable and portable.)    */
/* Also set some useful variables to record the      */
/* designer and company name.                        */
/* ================================================ */
   TOP = <design_name>
                       /* ========================== */
                       /* Note: Assumes design file- */
                       /* name and entity name are   */
                       /* the same (minus extension) */
                       /* ========================== */
   designer = "XSI Team"
   company  = "Xilinx, Inc"
   part     = "XCV150PQ240-3"
/* ================================================ */
/* Analyze and Elaborate the design file and specify */
/* the design file format.     */
```

```
/* ================================================ */
   analyze -format vhdl TOP + ".vhd"
                    /* ============================ */
                    /* You must analyze lower-level */
                    /* hierarchy modules here       */
                    /* ============================ */
   elaborate TOP
/* ================================================ */
/* Set the current design to the top level.         */
/* ================================================ */
   current_design TOP
/* ================================================ */
/* Set the synthesis design constraints.            */
/* ================================================ */
   remove_constraint –all
/* If setting timing constraints, do it here.
   For example:                                     */
/*
   create_clock <clock_pad_name> –period 50
*/
/* ================================================ */
/* Indicate those ports on the top-level module that */
/* should become chip-level I/O pads. Assign any I/O */
/* attributes or parameters and perform the I/O     */
/* synthesis.                                       */
/* ================================================ */
   set_port_is_pad "*"
   set_pad_type –slewrate HIGH all_outputs()
   insert_pads
/* ++++++++++++++++++++++++++++++++++++++++++++++++ */
```

```
/*               Compile the design                  */
/* ++++++++++++++++++++++++++++++++++++++++++++++++ */
   compile –map_effort med
/* ================================================ */
/* Write the design report files.                    */
/* ================================================ */
   report_fpga > TOP + ".fpga"
   report_timing > TOP + ".timing"
/* ================================================ */
/* Set the part type for the output netlist.         */
/* ================================================ */
   set_attribute TOP "part" –type string part
/* ================================================ */
/* Save design in EDIF format as <design>.sedif      */
/* ================================================ */
   write -format edif –hierarchy –output TOP + ".sedif"
/* ================================================ */
/* Write out the design to a DB.                     */
/* ================================================ */
   write -format db –hierarchy –output TOP + ".db"
/* ================================================ */
/* Write-out the timing constraints that were        */
/* applied earlier. (Note that any design hierarchy  */
/* needs to be flattened before the constraints are  */
/* written-out.)                                     */
/* ================================================ */
   write_script > TOP + ".dc"
/* ================================================ */
/* Call the Synopsys-to-Xilinx constraints translator*/
/* utility DC2NCF to convert the Synopsys constraints*/
```

```
/* to a Xilinx NCF file. You may like to view        */
/* dc2ncf.log to review the translation process.     */
/* ================================================= */
   sh dc2ncf -w TOP + ".dc"
/* ================================================= */
/* Exit the Compiler.                                */
/* ================================================= */
   exit
/* ================================================= */
/* Now run the Xilinx design implementation tools.   */
/* ================================================= */
```

The next example shows the modified run.script file (using the file run.script). Before using this script, notice the following items.

- The design compiles from the bottom up.
- The proper type of output for place and route in A1.5i is an SEDIF file when compiling a Virtex design in FPGA Compiler.
- Dont_touch commands appear in the script. Whenever you instantiate a component from the XSI synthesis library, you must place a Dont_touch on the instance to prevent Synopsys from deleting or modifying the library cell.

```
/* ================================================*/
/*    Sample Script for Synopsys to Xilinx Using     */
/*                   FPGA Compiler                   */
/*                                                   */
/*  Targets the Xilinx XCV150PQ240-3 and assumes a   */
/*   VHDL source file by way of an example.          */
/*                                                   */
/*   For general use with VIRTEX architectures.      */
/* ================================================*/
```

```
/* ================================================ */
/* Set the name of the design's top-level module.    */
/* (Makes the script more readable and portable.)    */
/* Also set some useful variables to record the      */
/* designer and company name.                        */
/* ================================================ */
   TOP = stopwatch
                        /* ========================== */
                        /* Note: Assumes design file- */
                        /* name and entity name are   */
                        /* the same (minus extension) */
                        /* ========================== */
   designer = "XSI Team"
   company  = "Xilinx, Inc"
   part     = "XCV150PQ240-3"
/* ================================================ */
/* Analyze and Elaborate the design file and specify */
/* the design file format.     */
/* ================================================ */
read -format verilog "smallcntr.v"
compile
read -format verilog "hex2led.v"
compile
read -format verilog "cnt60.v"
read -format verilog "stmchine.v"
read -format verilog "tenths.v"
read -format verilog TOP + ".v"
                      /* ============================ */
                      /* You must analyze lower-level */
```

```
                    /* hierarchy modules here       */
                    /* ============================ */
   elaborate TOP
/* ================================================= */
/* Set the current design to the top level.          */
/* ================================================= */
   current_design TOP
   set_dont_touch “DLL”
/* ================================================= */
/* Set the synthesis design constraints.             */
/* ================================================= */
   remove_constraint –all
/* If setting timing constraints, do it here.
   For example:                                       */
/*
   create_clock <clock_pad_name> –period 50
*/
/* ================================================= */
/* Indicate those ports on the top-level module that */
/* should become chip-level I/O pads. Assign any I/O */
/* attributes or parameters and perform the I/O      */
/* synthesis.                                        */
/* ================================================= */
   set_port_is_pad “*”
   set_pad_type –slewrate HIGH all_outputs()
   insert_pads
/* +++++++++++++++++++++++++++++++++++++++++++++++++ */
/*               Compile the design                  */
/* +++++++++++++++++++++++++++++++++++++++++++++++++ */
   compile –map_effort med
```

```
/* ================================================ */
/* Write the design report files.                   */
/* ================================================ */
/*   report_fpga > TOP + ".fpga"
   report_timing > TOP + ".timing" */
/* ================================================ */
/* Set the part type for the output netlist.        */
/* ================================================ */
   set_attribute TOP "part" -type string part
/* ================================================ */
/* Save design in EDIF format as <design>.sedif     */
/* ================================================ */
   write -format edif -hierarchy -output TOP + ".sedif"
/* ================================================ */
/* Write out the design to a DB.                    */
/* ================================================ */
/*   write -format db -hierarchy -output TOP + ".db" */
/* ================================================ */
/* Write-out the timing constraints that were       */
/* applied earlier. (Note that any design hierarchy  */
/* needs to be flattened before the constraints are  */
/* written-out.)                                    */
/* ================================================ */
/*   write_script > TOP + ".dc" */
/* ================================================ */
/* Call the Synopsys-to-Xilinx constraints translator*/
/* utility DC2NCF to convert the Synopsys constraints*/
/* to a Xilinx NCF file. You may like to view       */
/* dc2ncf.log to review the translation process.    */
/* ================================================ */
```

```
/*   sh dc2ncf -w TOP + ".dc" */
/* ================================================ */
/* Exit the Compiler.                               */
/* ================================================ */
/*    exit */
/* ================================================ */
/* Now run the Xilinx design implementation tools.   */
/* ================================================ */
```

4. Synthesize the design by first starting Design Analyzer. Type the following command in the directory that contains the .synopsys_dc.setup file for this design.

   `design_analyzer &`

   This brings up the Design Analyzer GUI.

5. When the GUI appears, run the script run.script by selecting Execute Script from the Setup pull-down menu. A pop-up window appears where you can select the script 'run.script' to run.

   If the script runs successfully, the script stops and creates an SXNF file. If you get errors, check the following.

   - Make sure you set up the .synopsys_dc.setup file correctly. Refer to the "Using Common Setup Procedures" section for more information.
   - Make sure you compiled the XDW synthesis libraries for the version of Synopsys you use. Refer to the "Using Common Setup Procedures" section for more information.
   - Make sure that the paths referenced in the .synopsys_dc.setup file exist. Refer to the "Using Common Setup Procedures" section for more information.
   - Make sure you run the design_analyzer & command in the directory that contains the .synopsys_dc.setup file when you start Design Analyzer.

6. Take the SXNF file produced by Design Analyzer and place and route the design for timing simulation using the following script.

   ```
   #!/bin/csh –f
   ```

```
ngdbuild –p v50PC84-4 watch.sxnf
map watch.ngd
par watch.ncd stopwatch_r.ncd
ngdanno stopwatch_r.ncd
ngd2ver –ul stopwatch_r.nga
```

Optionally, you can place and route the SXNF files using the A1.5i GUI. Refer to the *Quick Start Guide Tutorial* for more information about using the GUI for place and route.

Conducting Timing Simulation

To conduct timing simulation, use NGD2VER to create an SDF file and structural Verilog file, along with a testbench. Run the following command at the command-line.

```
verilog –f filest.f
```

filest.f contains the names of the two files used in timing simulation. By default, the Verilog file produced by NGD2VER uses the $sdf_annotate directive, which annotates the SDF file with the Verilog file from NGD2VER.

Note: Before running timing simulation make sure you run NGD2VER with the –ul option.

Virtex Verilog Alliance FPGA Express v2.1/VerilogXL v2.5 Tutorial

This tutorial familiarizes you with the A1.5i XSI Verilog FPGA Express v2.1 / VerilogXL v2.5 design flow, presenting common tasks such as locking pins and setting slew rate.

Getting Ready for this Tutorial

To use this A1.5i XSI tutorial, you must be using A1.5i XSI, FPGA Express v2.1 or better, and VerilogXL v2.5 or better. Refer to the instructions listed in the “Using Common Setup Procedures” section for detailed information. This design uses Virtex features.

Download the file 4vrtxsiverexp.tar.Z from the Xilinx web site. Untar and uncompress this file in the directory of your choice, but this tutorial assumes you use the /home/user/tutorial directory.

Conducting Functional Simulation

To run a functional simulation, you need to compile the design files and testbench with VerilogXL. This tutorial assumes that you uncompressed, untarred, and placed your setup files in /home/user/tutorial. If you placed the files elsewhere, replace your path appropriately in the following instructions.

The A1.5i XSI VerilogXL functional simulation flow allows you to simulate instantiated XSI cells such as FDCE and OSC4.

To conduct functional simulation, use the following steps.

1. Change directories to the /home/user/tutorial directory.
2. You perform functional simulation by running the VerilogXL command verilog with the data file filesf.f. Type the following command at the UNIX prompt.

   `verilog –f filesf.f`

   VerilogXL issues error messages when you attempt to simulate, including the following.

```
Compiling source file “testbenchf.v”
Compiling source file “stopwatch.v”
Compiling source file “stmchine.v”
Compiling source file “hex2led.v”
Compiling source file “cnt60.v”
Compiling source file “smallcntr.v”
Compiling source file “tenths.v”

Error!   Module or primitive (BUFG) not defined          [Verilog-MOPND]
         “stopwatch.v”, 24: BUFG CLOCKBUF(.I(oscout), .O(
          clkint));
```

3. You get the error messages because the tutorial design contains instantiations of LogiBLOX and instantiated XSI synthesis library cells (such as OSC4, FDCE, and BUFG for example). You must tell the Verilog simulator where to find the models for these cells by placing the ‘uselib directive in the top-level file of this design,

stopwatch.v. Add the following line to the top of your stopwatch.v file.

```
`uselib dir=$XILINX/verilog/src/UNIVIRTEX libext=.v
```

4. Replace the $XILINX text with the explicit path in your environment. If you set $XILINX to /home/software/xilinx, the ‘uselib line appears in the top of the stopwatch.v file as the following.

```
`uselib dir=/home/software/xilinx/ \
verilog/src/UNIVIRTEX libext=.v
```

5. After making the change, re-run the functional simulation command.

```
verilog –f filesf.f
```

Synthesizing Your Design

In this section of the tutorial, you synthesize the design and create a place and routed NCD file. After creating the place and routed NCD file, you can optionally proceed to create a BIT file for downloading to the demo board, using bitgen and promgne, or the Hardware Debugger. For more information about using the FPGA Express GUI, refer to the FPGA Express on-line help.

Create an FPGA Express project, enter source files, and specify a target device(4003EPC84-3) as shown in the following figure.

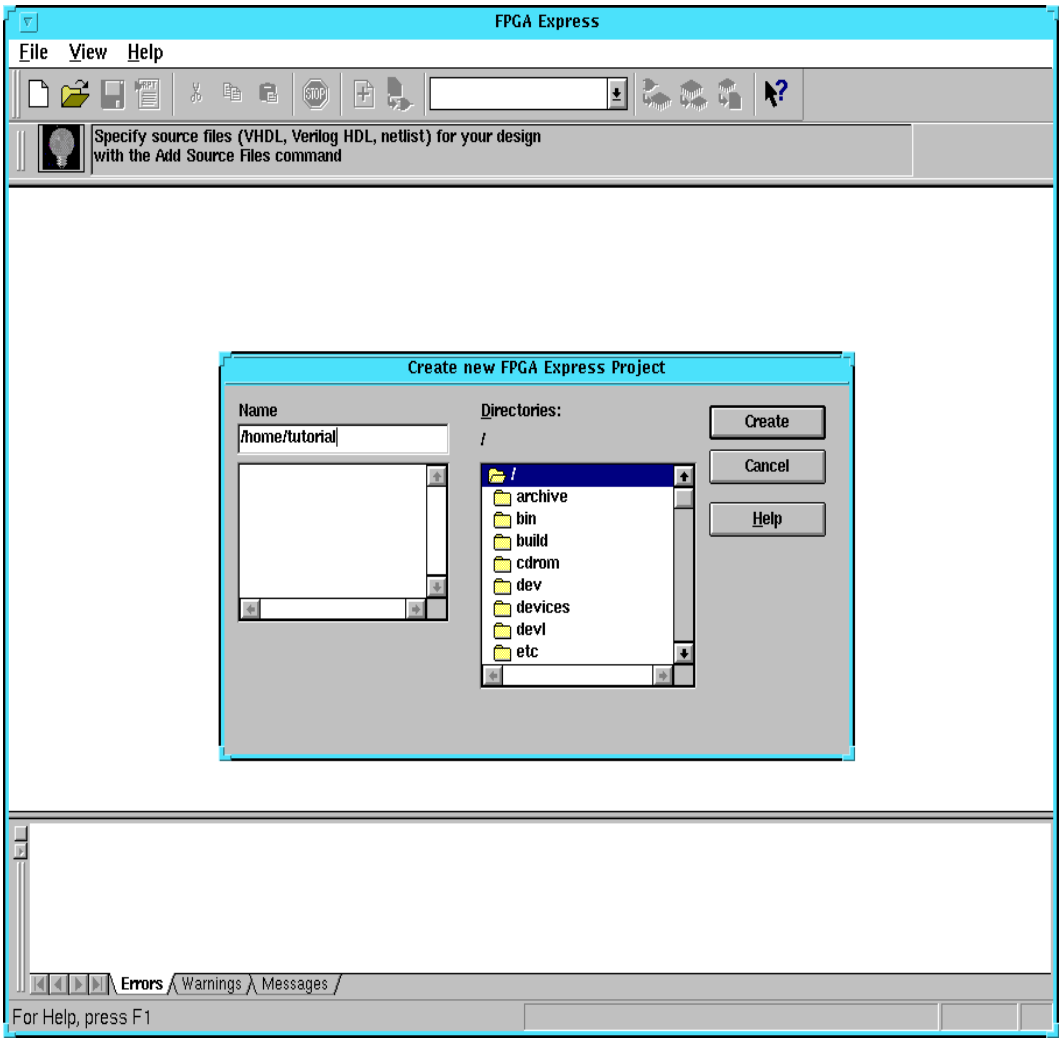

Figure 1-11 Creating an FPGA Express Verilog Project File

Add the design files, as shown in the following figure.

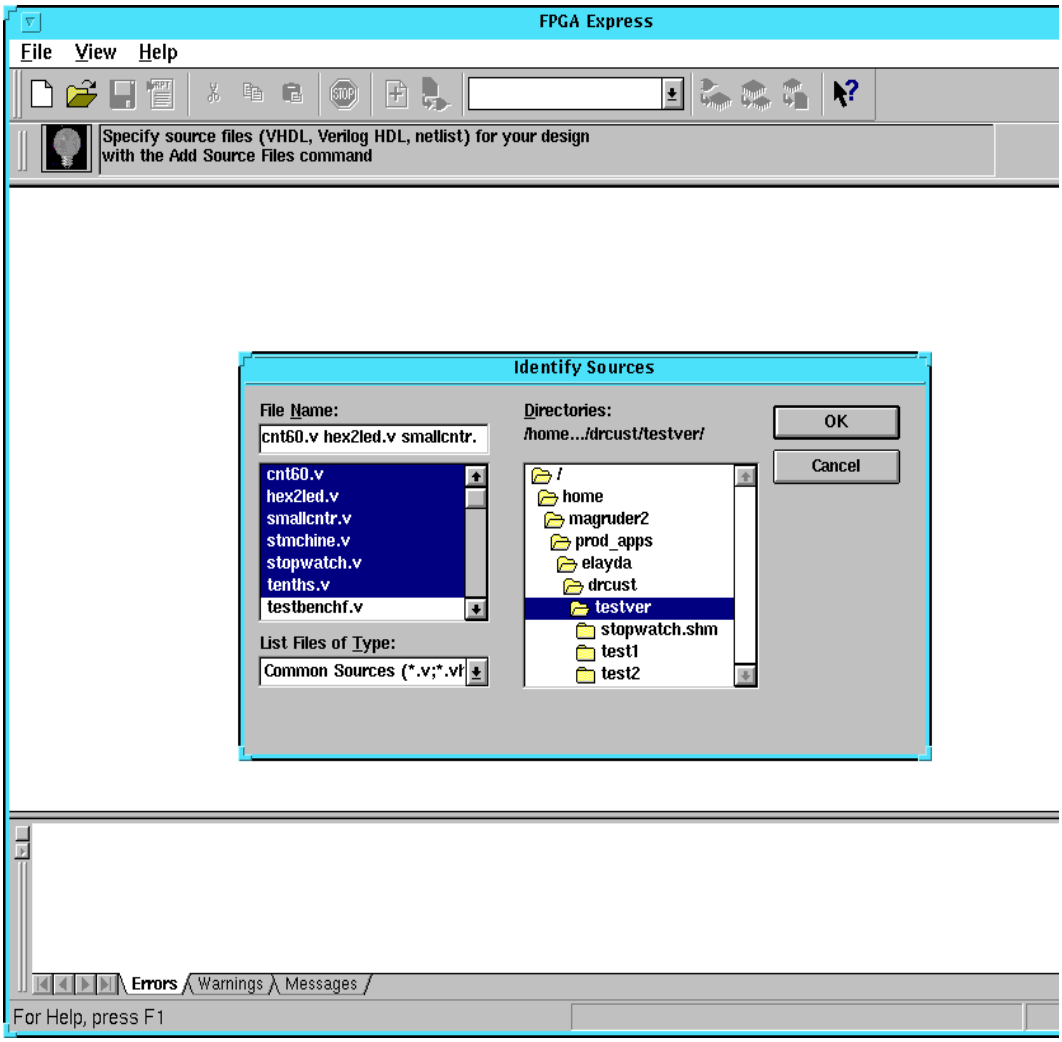

Figure 1-12 Adding Design Files for an FPGA Express Design

Select the top level entity and select a target device as shown in the following illustration.

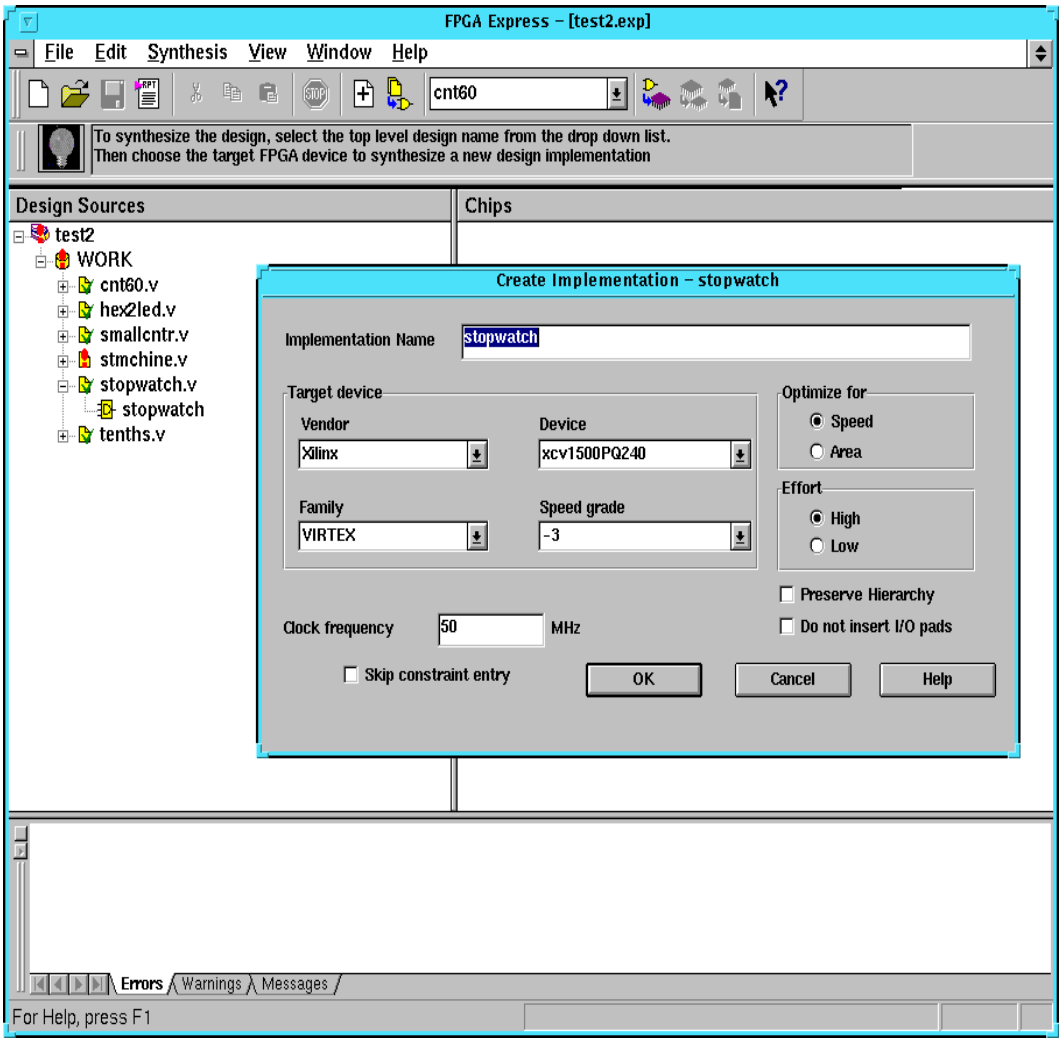

Figure 1-13 Selecting the Target Device (Virtex)

An implementation appears in the right-hand window. Select the implementation and press the Optimize button on the tool bar, as the following figure shows.

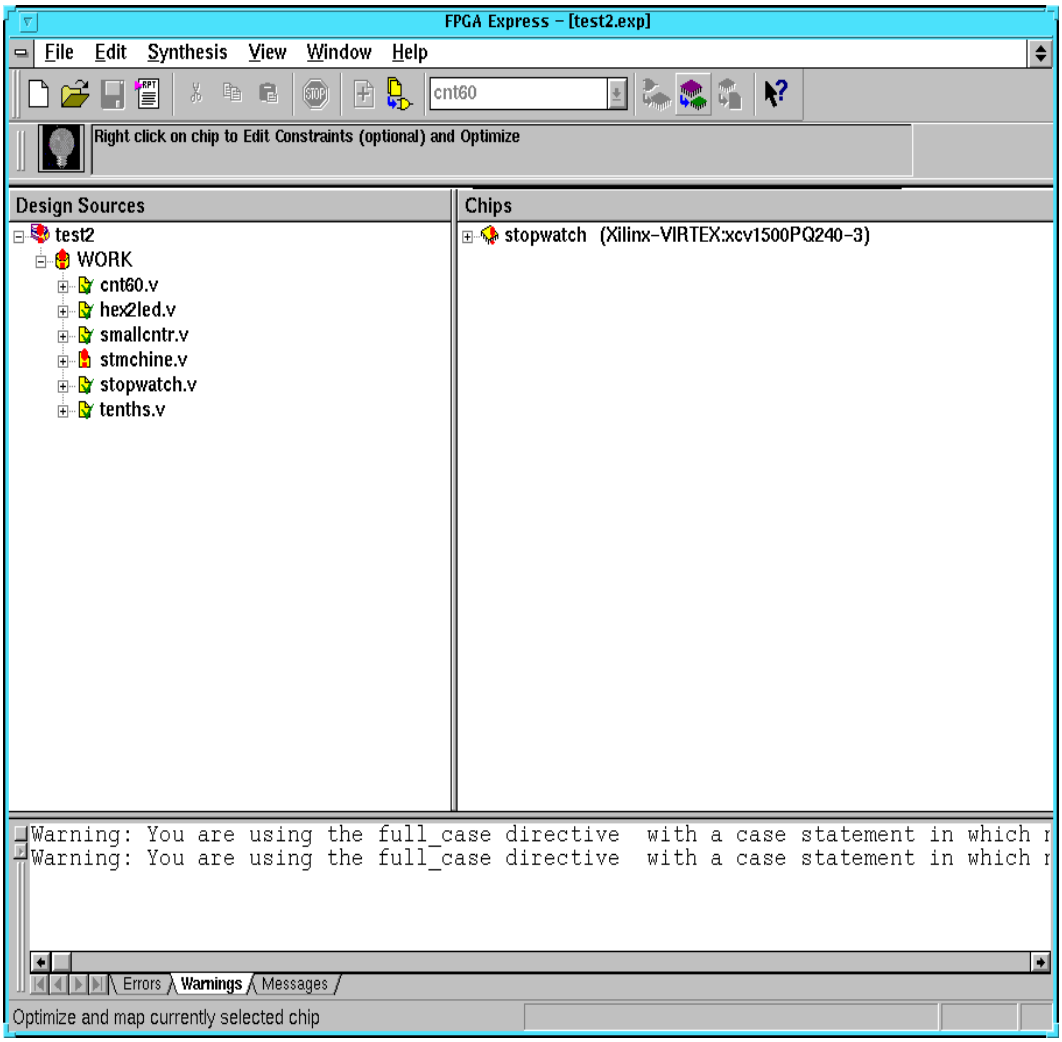

Figure 1-14 Optimizing an Implementation (Virtex)

Select the optimized design and write out the netlist by pressing the Export Netlist button on the toolbar, illustrated in the following figure.

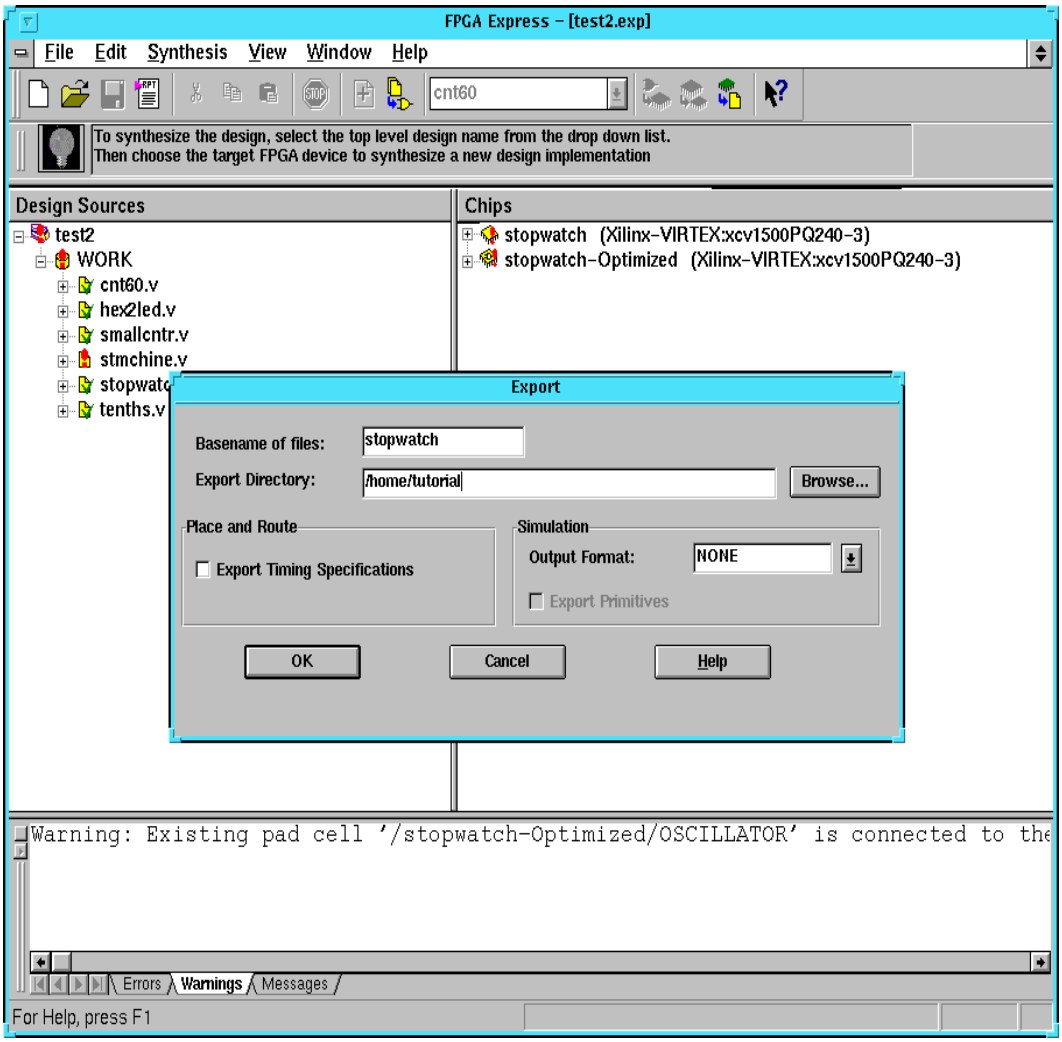

Figure 1-15 Writing Out an Optimized Netlist (Virtex)

Take the EDIF file produced by FPGA Express and place and route the design for timing simulation using the following script.

```
#!/bin/csh –f
ngdbuild –p v50pq240-4 watch.edf
map watch.ngd
```

```
par watch.ncd stopwatch_r.ncd

ngdanno stopwatch_r.ncd

ngd2ver –ul stopwatch_r.nga
```

Optionally, you can place and route the design using the A1.5i GUI. Refer to the *Quick Start Guide Tutorial* for more information about using the GUI for place and route.

Conducting Timing Simulation

To perform timing simulation, use NGD2VER to create an SDF file and structural Verilog file, along with a testbench.

To perform timing simulation, run the following command at the command-line

```
verilog –f filest.f
```

filest.f contains the names of the two files used in timing simulation. By default, the Verilog file produced by NGD2VER uses the $sdf_annotate directive, which annotates the SDF file with the Verilog file from NGD2VER.

Note: Run NGD2VER with the –ul option before running timing simulation.

The timing simulation testbench contains a definition for the Verilog macro GSR_SIGNAL. You must toggle this macro at the start of timing simulation. To decide on the proper pulse width, consult the *Databook* for the die-speed grade appropriate to your simulation.

Virtex VHDL FPGA Compiler VSS Tutorial

This tutorial familiarizes you with the A1.5i FPGA Compiler / VSS design flow and includes a VHDL design you can optionally download to the demo board. This tutorial presents common tasks such as locking pins and setting slew rate.

Getting Ready for this Tutorial

To use this tutorial, you need the software described in the “Using Common Setup Procedures” section. If you use a version of Synopsys newer than v1997.01, you must follow the additional steps in the “Setting Up for the Spartan / XC4000 Verilog FPGA Compiler Tuto-

rial" section. If you want to target a device other than Virtex, reference the directories that apply to that family (for example, the XDW libraries have separate directories for XC4000XL, Spartan, and Virtex). Use synlibs with the exact die-speed target desired.

Setting Up for the Virtex VHDL FPGA Compiler Tutorial

To use this tutorial, ensure installation of your Xilinx and Synopsys software and know where the software resides on your system. If you use a version of Synopsys newer than v1997.01, you must re-compile the XSI XDW and simulation libraries. Refer to the setup instructions in the "Setting up for FPGA Compiler" section.

For this tutorial, recompile only the libraries related to synthesizing a Virtex device (if using a version of Synopsys newer than v1997.01), and ensure compilation of the Virtex XDW DesignWare libraries. The A1.5i simulation libraries come in two parts, a functional simulation part and a timing simulation part. The A1.5i XSI VHDL functional simulation libraries are called UNISIM. The A1.5i XSI VHDL timing simulation libraries are called SIMPRIM libraries. The SIMPRIM library is a VITAL simulation library.

Compile the Spartan XDW DesignWare libraries, XDW simulation libraries, LogiBLOX simulation libraries, UNISIM simulation libraries, and SIMPRIM libraries. All of these libraries contain compile scripts which let you compile these libraries in the $XILINX area. However, to use these scripts, you need write permissions to the $XILINX area. If you do not have write permissions, make a local copy of the $XILINX/synopsys/libraries directory and use the following instructions, but instead of changing directories to $XILINX/synopsys/libraries, change directories to your local copy of that area.

Use the following instructions to compile the libraries Maker sure you have followed the directions listed in the "Using Common Setup Procedures" section.

1. Change directories to the $XILINX/synopsys/libraries/dw/src/virtex directory.
2. Type the following command at the UNIX prompt

   ```
   dc_shell -f install_dw.dc
   ```

3. Change directories to the $XILINX/synopsys/libraries/sim/src/simprims directory.
4. Type the following command at the UNIX prompt.

   `./analyze.csh`

5. Change directories to the $XILINX/synopsys/libraries/sim/src/unisims directory.
6. Type the following command at the UNIX prompt.

   `./analyze.csh`

7. In the empty directory you created earlier in the "Using Common Setup Procedures" section, copy the file virtxxsvhdlc.tar.Z. Uncompress and untar this file.
8. Create a .synospys_dc.setup file and a .synopsys_vss.setup using the templates provided in the $XILINX/.synopsys/examples area and the instructions listed in the "Using Common Setup Procedures" section.
9. You must add to the .synopsys_dc.setup file several lines related to synthesis libraries. Use the A1.5i XSI tool synlibs, which displays the synthesis library information needed for a given die-speed combination. To add the correct information into the .synopsys_dc.setup file for a Virtex, type the following command in the same directory as the .synopsys_dc.setup file.

   `synlibs -fc  v50pq240-4 >> .synopsys_dc.setup`

   This appends the output of synlibs into the .synopsys_dc.setup file. If you compiled the Virtex XDW libraries in the $XILINX tree, you can proceed to checking the .synopsys_vss.setup file. If you compiled the XDW libraries in the $XILINX area, then you need not modify the define_design_lib line in the .synopsys_dc.setup file.

10. If you copied the $XILINX/synopsys/libraries directory locally, change the path in the .synopsys_dc.setup file to reflect the copied directory path. For example, if you copied $XILINX/libraries to /home/data, then the `define_design_lib' setting made by synlibs, edit the previous v50-4 examplle to reflect the /home/data location.

    `define_design_lib xdw_virtex /home/data/libraries/ \`
    `dw/lib/virtex`

11. In the directory where you created your .synopsys_dc.setup, .synopsys_vss.setup and where you uncompressed and untarred the file virtexxsivhdle.tar.Z file, type the following command.

    `ls -l`

12. Make sure the directory contains the following items.

    - .synopsys_dc.setup
    - .synopsys_vss.setup
    - WORK
    - func
    - synth
    - time

    The func directory contains all files and scripts needed for functional simulation. The synth directory contains all files and scripts needed for synthesis. The time directory contains files needed for timing simulation.

Conducting Functional Simulation

To conduct functional simulation, you must first compile the design files and testbench, then run the VSS simulation tool. This tutorial assumes that you uncompressed, untarred, and placed your setup files in /home/user/tutorial. If you placed these files elsewhere, replace your path appropriately in the following instructions.

The A1.5i XSI VSS functional simulation flow allows you to simulate instantiated XSI cells such as FDCE and CLKDLL. Additionally, by using the UNISIM simulation libraries, you can simulate and implement the GSR without impact to the design or testbench.

To conduct functional simulation, use the following instructions.

1. Change directories to the /home/user/tutorial directory.
2. You conduct functional simulation by running the script func.script. Before running the script, open the file func.script in a text editor. Notice the following items in this file.

    - The files for simulation read in from the bottom up.
    - The testbench reads in last.

- vhdlan uses the –i option. Always use vhdlan with the –i option. By default, vhdlan uses the –c option, which works only if your system uses a certain type of C compiler. If you want to use the –c option with vhdlan, refer to the Synopsys web site for the proper setup.
- The contents of the func.sim file.

```
#!/bin/csh –f
rm –r WORK
mkdir WORK
vhdlan –i tenths.vhd
vhdlan –i smallcntr.vhd
vhdlan –i cnt60.vhd
vhdlan –i hex2led.vhd
vhdlan –i stmchine.vhd
vhdlan –i stopwatch.vhd
vhdlan –i testbenchf.vhd
vhdlsim –e commandf.txt overall
```

3. Run the functional simulation by typing the following command at the UNIX prompt.

   `./func.sim`

4. You get error messages because the VHDL code contains an instantiated component (OSC4) that does not have an underlying RTL behavioral description; you must make an RTL model for functional simulation. For more information on OSC4, refer to the *Libraries Guide* and the *Databook*. Use the UNISIM libraries to functionally simulate library cells instantiated from the XSI libraries. VHDL code that instantiates any XSI library cell must contain the following two lines.

   ```
   library UNISIM;
   use UNISIM.all;
   ```

   In general, you need these lines only in the files that contain instantiated XSI library cells (such as FDCE, RAM32X1S, and BUFG, for example). However, you can place the above two lines in every VHDL file in your design.

The following example shows the stopwatch.vhd file with the two lines added.

```
library IEEE;
use IEEE.std_logic_1164.all;
library UNISIM;
use UNISIM.all;

entity stopwatch is

    port (    RESET : in STD_LOGIC;
   STRTSTOP : in STD_LOGIC;
  TENTHSOUT : out STD_LOGIC_VECTOR(9 downto 0);
    ONESOUT : out STD_LOGIC_VECTOR(6 downto 0);
    TENSOUT : out STD_LOGIC_VECTOR(6 downto 0);
    CLOCK: in STD_LOGIC
 );
end stopwatch;

architecture inside of stopwatch is

component BUFGDLL
    port (I: in STD_LOGIC; O: out STD_LOGIC);
end component;

component BUFG
    port (I : in STD_LOGIC;
  O : out STD_LOGIC);
end component;

component stmchine
```

```
    port (     CLK : in STD_LOGIC;
     RESET : in STD_LOGIC;
  STRTSTOP : in STD_LOGIC;
     CLKEN : out STD_LOGIC;
       RST : out STD_LOGIC
 );
end component;

component tenths
    port (     CLOCK : in STD_LOGIC;
      CLK_EN : in STD_LOGIC;
  ASYNC_CTRL : in STD_LOGIC;
    TERM_CNT : out STD_LOGIC;
       Q_OUT : out STD_LOGIC_VECTOR(9 downto 0));
end component;

component cnt60
    port (    CE : in STD_LOGIC;
     CLK : in STD_LOGIC;
     CLR : in STD_LOGIC;
  LSBSEC : out STD_LOGIC_VECTOR(3 downto 0);
  MSBSEC : out STD_LOGIC_VECTOR(3 downto 0));
end component;

component hex2led
    port (HEX : in STD_LOGIC_VECTOR(3 downto 0);
  LED : out STD_LOGIC_VECTOR(6 downto 0));
end component;

signal strtstopinv : STD_LOGIC;
```

```
signal oscout : STD_LOGIC;
signal clkint : STD_LOGIC;
signal clkenable : STD_LOGIC;
signal rstint : STD_LOGIC;
signal xcountout : STD_LOGIC_VECTOR(9 downto 0);
signal xtermcnt : STD_LOGIC;
signal cnt60enable : STD_LOGIC;
signal lsbcnt : STD_LOGIC_VECTOR(3 downto 0);
signal msbcnt : STD_LOGIC_VECTOR(3 downto 0);

begin

DLL:BUFGDLL port map(I=>CLOCK,O=>oscout);

CLOCKBUF:BUFG port map(I=>oscout,O=>clkint);

MACHINE:stmchine port map(CLK=>clkint,
                  RESET=>RESET,
                  STRTSTOP=>strtstopinv,
                  CLKEN=>clkenable,
                  RST=>rstint
 );

XCOUNTER:tenths port map(CLOCK=>clkint,
                 CLK_EN=>clkenable,
                 ASYNC_CTRL=>rstint,
                 TERM_CNT=>xtermcnt,
                 Q_OUT=>xcountout
                        );
```

```
sixty: cnt60 port map(CE=>cnt60enable,
              CLK=>clkint,
              CLR=>rstint,
              LSBSEC=>lsbcnt,
              MSBSEC=>msbcnt
                     );

lsbled:hex2led port map(HEX=>lsbcnt,
LED=>ONESOUT
                       );

msbled:hex2led port map(HEX=>msbcnt,
LED=>TENSOUT
                       );
cnt60enable<=xtermcnt and clkenable;
TENTHSOUT<=not(xcountout);
strtstopinv<=not(STRTSTOP);

end inside;
```

5. After correcting the stopwatch.vhd file, recompile the func.sim file. The simulation starts and displays the waveform viewer. The UNIX shell displays the vhdlsim prompt.

Synthesizing Your Design

In this section of the tutorial, you synthesize the design and create a place and routed NCD file. After creating the place and routed NCD file, you can optionally create a BIT file for downloading to the demo board, using bitgen and promgne, or the Hardware Debugger.

Use the following instructions to synthesize your design.

1. Create a compile script. You can modify the default compile script located in the A1.5i software. Copy the file $XILINX/template.fpga.script.4kex into your /home/user/tutorial direc-

tory, which contains the .synopsys_dc.setup and .synopsys_vss.setup files you created earlier.

2. Rename the file template.fpga.script.virtex to run.script.

   Open the file run.script in a text editor and notice the following items. The file run.script compiles only one VHDL file. You must comment out several lines not relevant to synthesis of this particular design.

   The following example shows the unmodified run.script file (using the file template.fpga.script.virtex).

```
/* ================================================*/
/*    Sample Script for Synopsys to Xilinx Using     */
/*                   FPGA Compiler                   */
/*                                                   */
/*  Targets the Xilinx XCV150PQ240-3 and assumes a   */
/*   VHDL source file by way of an example.          */
/*                                                   */
/*   For general use with VIRTEX architectures.      */
/* ================================================*/

/* ================================================ */
/* Set the name of the design's top-level module.    */
/* (Makes the script more readable and portable.)    */
/* Also set some useful variables to record the      */
/* designer and company name.                        */
/* ================================================ */
   TOP = <design_name>
                       /* ========================== */
                       /* Note: Assumes design file- */
                       /* name and entity name are   */
                       /* the same (minus extension) */
                       /* ========================== */
```

```
   designer = “XSI Team”
   company  = “Xilinx, Inc”
   part     = “XCV150PQ240-3”
/* ================================================ */
/* Analyze and Elaborate the design file and specify */
/* the design file format.     */
/* ================================================ */
   analyze –format vhdl TOP + “.vhd”
                    /* ============================ */
                    /* You must analyze lower-level */
                    /* hierarchy modules here       */
                    /* ============================ */
   elaborate TOP
/* ================================================ */
/* Set the current design to the top level.         */
/* ================================================ */
   current_design TOP
/* ================================================ */
/* Set the synthesis design constraints.            */
/* ================================================ */
   remove_constraint –all
/* If setting timing constraints, do it here.
   For example:                                     */
/*
   create_clock <clock_pad_name> –period 50
*/
/* ================================================ */
/* Indicate those ports on the top-level module that */
/* should become chip-level I/O pads. Assign any I/O */
/* attributes or parameters and perform the I/O     */
```

```
/* synthesis.                                          */
/* ================================================== */
   set_port_is_pad “*”
   set_pad_type -slewrate HIGH all_outputs()
   insert_pads
/* ++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*               Compile the design                   */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++ */
   compile –map_effort med
/* ================================================== */
/* Write the design report files.                     */
/* ================================================== */
   report_fpga > TOP + “.fpga”
   report_timing > TOP + “.timing”
/* ================================================== */
/* Set the part type for the output netlist.          */
/* ================================================== */
   set_attribute TOP “part” –type string part
/* ================================================== */
/* Save design in EDIF format as <design>.sedif       */
/* ================================================== */
   write –format edif –hierarchy –output TOP + “.sedif”
/* ================================================== */
/* Write out the design to a DB.                      */
/* ================================================== */
   write –format db –hierarchy –output TOP + “.db”
/* ================================================== */
/* Write-out the timing constraints that were         */
/* applied earlier. (Note that any design hierarchy   */
/* needs to be flattened before the constraints are   */
```

```
/* written-out.)                                        */
/* ================================================== */
   write_script > TOP + “.dc”
/* ================================================== */
/* Call the Synopsys-to-Xilinx constraints translator*/
/* utility DC2NCF to convert the Synopsys constraints*/
/* to a Xilinx NCF file. You may like to view        */
/* dc2ncf.log to review the translation process.     */
/* ================================================== */
   sh dc2ncf –w TOP + “.dc”
/* ================================================== */
/* Exit the Compiler.                                 */
/* ================================================== */
   exit
/* ================================================== */
/* Now run the Xilinx design implementation tools.   */
/* ================================================== */
```

The following example shows the modified run.script file (using the file run.script).

Before using this script, notice the following items.

- The design compiles from the bottom up.
- The proper type of output for place and route in A1.5i is an SEDIF file when compiling a Virtex design in FPGA Compiler.
- Dont_touch commands appear in the script. Whenever you instantiate a component from the XSI synthesis library, you must place a Dont_touch on the instance to prevent Synopsys from deleting or modifying the library cell.

```
/* ================================================*/
/*    Sample Script for Synopsys to Xilinx Using     */
```

```
/*                  FPGA Compiler                    */
/*                                                   */
/*  Targets the Xilinx XCV150PQ240-3 and assumes a   */
/*   VHDL source file by way of an example.          */
/*                                                   */
/*   For general use with VIRTEX architectures.      */
/* ================================================*/
/* ================================================ */
/* Set the name of the design's top-level module.    */
/* (Makes the script more readable and portable.)    */
/* Also set some useful variables to record the      */
/* designer and company name.                        */
/* ================================================ */
   TOP = stopwatch
                       /* ========================== */
                       /* Note: Assumes design file- */
                       /* name and entity name are   */
                       /* the same (minus extension) */
                       /* ========================== */
   designer = "XSI Team"
   company  = "Xilinx, Inc"
   part     = "XCV150PQ240-3"
/* ================================================ */
/* Analyze and Elaborate the design file and specify */
/* the design file format.     */
/* ================================================ */
analyze -format vhdl "smallcntr.vhd"
elaborate smallcntr
compile
analyze -format vhdl "hex2led.vhd"
```

```
elaborate hex2led
compile
analyze –format vhdl “cnt60.vhd”
analyze –format vhdl “stmchine.vhd”
analyze –format vhdl “tenths.vhd”
analyze –format vhdl TOP + “.vhd”
                    /* ============================ */
                    /* You must analyze lower-level */
                    /* hierarchy modules here       */
                    /* ============================ */
   elaborate TOP
/* ================================================ */
/* Set the current design to the top level.         */
/* ================================================ */
   current_design TOP
   set_dont_touch “DLL”
/* ================================================ */
/* Set the synthesis design constraints.            */
/* ================================================ */
   remove_constraint –all
/* If setting timing constraints, do it here.
   For example:                                     */
/*
   create_clock <clock_pad_name> –period 50
*/
/* ================================================ */
/* Indicate those ports on the top-level module that */
/* should become chip-level I/O pads. Assign any I/O */
/* attributes or parameters and perform the I/O     */
/* synthesis.                                       */
```

```
/* ================================================ */
   set_port_is_pad “*”
   set_pad_type –slewrate HIGH all_outputs()
   insert_pads
/* ++++++++++++++++++++++++++++++++++++++++++++++++ */
/*               Compile the design                 */
/* ++++++++++++++++++++++++++++++++++++++++++++++++ */
   compile –map_effort med
/* ================================================ */
/* Write the design report files.                   */
/* ================================================ */
/*   report_fpga > TOP + “.fpga”
   report_timing > TOP + “.timing” */
/* ================================================ */
/* Set the part type for the output netlist.        */
/* ================================================ */
   set_attribute TOP “part” –type string part
/* ================================================ */
/* Save design in EDIF format as <design>.sedif     */
/* ================================================ */
   write –format edif –hierarchy –output TOP + “.sedif”
/* ================================================ */
/* Write out the design to a DB.                    */
/* ================================================ */
/*   write –format db –hierarchy –output TOP + “.db” */
/* ================================================ */
/* Write-out the timing constraints that were       */
/* applied earlier. (Note that any design hierarchy  */
/* needs to be flattened before the constraints are  */
/* written-out.)                                    */
```

```
/* ================================================ */
/*   write_script > TOP + ".dc" */
/* ================================================ */
/* Call the Synopsys-to-Xilinx constraints translator*/
/* utility DC2NCF to convert the Synopsys constraints*/
/* to a Xilinx NCF file. You may like to view       */
/* dc2ncf.log to review the translation process.    */
/* ================================================ */
/*   sh dc2ncf –w TOP + ".dc" */
/* ================================================ */
/* Exit the Compiler.                               */
/* ================================================ */
/*    exit */
/* ================================================ */
/* Now run the Xilinx design implementation tools.  */
/* ================================================ */
```

3. Synthesize the design by first starting Design Analyzer. Type the following command in the directory that contains the .synopsys_dc.setup file for this design.

   `design_analyzer &`

   This brings up the Design Analyzer GUI.

4. When the GUI appears, run the script run.script by selecting Execute Script from the Setup pull-down menu. A pop-up window appears where you can select the script run.script.

   If the script runs successfully, the script stops and creates an SXNF file. If an error occurs, check the following.

   - Make sure that you set up the .synopsys_dc.setup file correctly.
   - Make sure that you compiled the XDW synthesis libraries for the version of Synopsys you use.

- Make sure that the paths referenced in the .synopsys_dc.setup file exist. Refer to the "Using Common Setup Procedures" section for more information.
- Make sure you run the design_analyzer & command in the directory that contains the .synopsys_dc.setup file when you start Design Analyzer.

5. Take the SXNF file produced by Design Analyzer and place and route the design for timing simulation using the following script.

```
#!/bin/csh –f

ngdbuild –p v50PC84-4 stopwatch.sxnf

map stopwatch.ngd

par stopwatch.ncd stopwatch_r.ncd

ngdanno stopwatch_r.ncd

ngd2vhdl stopwatch_r.nga
```

   Optionally, you can place and route the EDIF files using the A1.5i GUI. Refer to the *Quick Start Guide Tutorial* for more information about using the GUI for place and route.

Conducting Timing Simulation

To perform timing simulation, use NGD2VHDL to create an SDF file and structural VHDL file, along with a testbench. Use the script file timing.sim to perform the timing simulation.

You run timing.sim in the directory that contains the .synopsys_vss.setup file you created earlier.

Open the script file timing.sim in a text editor and notice the following.

- vhdlan uses the –i option. Always use vhdlan with the –i option. By default, vhdlan uses the –c option, which works only if your system uses a certain type of C compiler. If you want to use the –c option with vhdlan, refer to the Synopsys web site for the proper setup.
- The -sdf_top option is specified first when invoking vhdldbx or vhdlsim.

The following example shows the contents of the file timing.sim.

```
#!/bin/csh -f

rm -r WORK

mkdir WORK

vhdlan –i stopwatch_r.vhd

vhdlan –i testbencht.vhd

vhdlsim –sdf_top /testbenchf/uut –sdf \
stopwatch_r.sdf –e commandt.txt overall
```

Close the timing.sim file in the text editor. Run the timing simulation, which produces a waveform view similar to the functional simulation, by typing the following command at the UNIX prompt.

```
./timing.sim
```

If you get error and warning about “Bad Regions” or undefined libraries, make sure you set up the simulation libraries for A1.5i XSI VSS correctly. Make sure that the paths in the .synopsys_vss.setup file point to paths which exist in your setup. For further information, refer to the “Using Common Setup Procedures” section.

Virtex VHDL FPGA Compiler VSS Tutorial

This tutorial familiarizes you with the A1.5i FPGA Compiler / VSS design flow and includes a VHDL design you can optionally downloaded to the demo board. This tutorial presents common tasks such as locking pins and setting slew rate.

Getting Ready for this Tutorial

To use this tutorial, you need the software described in the “Using Common Setup Procedures” section. If you want to target a device other than Virtex, reference the directories that apply to that family (for example, the XDW libraries have separate directories for XC4000XL, Spartan, and Virtex). Use synlibs with the exact die-speed target desired.

Setting Up for the Virtex VHDL FPGA Compiler VSS Tutorial

To use this tutorial, ensure installation of your Xilinx and Synopsys software and know where the software resides on your system. If you

use a version of Synopsys newer than v1997.01, you must re-compile the XSI XDW and simulation libraries. Refer to the setup instructions in the "Setting up for FPGA Compiler" section.

For this tutorial, recompile only the libraries related to synthesizing a Virtex device (if using a version of Synopsys newer than v1997.01), and ensure compilation of the Virtex XDW DesignWare libraries. The A1.5i simulation libraries come in two parts, a functional simulation part and a timing simulation part. The A1.5i XSI VHDL functional simulation libraries are called UNISIM. The A1.5i XSI VHDL timing simulation libraries are called SIMPRIM libraries. The SIMPRIM library is a VITAL simulation library.

Compile the Spartan XDW DesignWare libraries, XDW simulation libraries, LogiBLOX simulation libraries, UNISIM simulation libraries, and SIMPRIM libraries. All of these libraries contain compile scripts which let you compile these libraries in the $XILINX area. However, to use these scripts, you need write permissions to the $XILINX area. If you do not have write permissions, make a local copy of the $XILINX/synopsys/libraries directory and use the following instructions, but instead of changing directories to $XILINX/synopsys/libraries, change directories to your local copy of that area.

Use the following instructions to compile the libraries Maker sure you have followed the directions listed in the "Using Common Setup Procedures" section.

1. Change directories to the $XILINX/synopsys/libraries/dw/src/virtex directory.
2. Type the following command at the UNIX prompt

   ```
   dc_shell -f install_dw.dc
   ```

3. Change directories to the $XILINX/synopsys/libraries/sim/src/simprims directory.
4. Type the following command at the UNIX prompt.

   ```
   ./analyze.csh
   ```

5. Change directories to the $XILINX/synopsys/libraries/sim/src/unisims directory.
6. Type the following command at the UNIX prompt.

   ```
   ./analyze.csh
   ```

7. In the empty directory you created earlier in the “Using Common Setup Procedures” section, copy the file virtxxsvhdlc.tar.Z. Uncompress and untar this file.
8. Create a .synospys_dc.setup file and a .synopsys_vss.setup using the templates provided in the $XILINX/.synopsys/examples area and the instructions listed in the “Using Common Setup Procedures” section.
9. You must add to the .synopsys_dc.setup file several lines related to synthesis libraries. Use the A1.5i XSI tool synlibs, which displays the synthesis library information needed for a given die-speed combination. To add the correct information into the .synopsys_dc.setup file for a Virtex, type the following command in the same directory as the .synopsys_dc.setup file.

   ```
   synlibs -fc  v50pq240-4 >> .synopsys_dc.setup
   ```

   This appends the output of synlibs into the .synopsys_dc.setup file. If you compiled the Virtex XDW libraries in the $XILINX tree, you can proceed to checking the .synopsys_vss.setup file. If you compiled the XDW libraries in the $XILINX area, then you need not modify the define_design_lib line in the .synopsys_dc.setup file.
10. If you copied the $XILINX/synopsys/libraries directory locally, change the path in the .synopsys_dc.setup file to reflect the copied directory path. For example, if you copied $XILINX/libraries to /home/data, then the `define_design_lib’ setting made by synlibs, edit the previous v50-4 example to reflect the /home/data location.

    ```
    define_design_lib xdw_virtex /home/data/libraries/ \
    dw/lib/virtex
    ```

11. In the directory where you created your .synopsys_dc.setup, .synopsys_vss.setup and where you uncompressed and untarred the file virtexxsivhdle.tar.Z file, type the following command.

    ```
    ls -l
    ```

12. Make sure the directory contains the following items.
    - .synopsys_dc.setup
    - .synopsys_vss.setup
    - WORK

- func
- synth
- time

The func directory contains all files and scripts needed for functional simulation. The synth directory contains all files and scripts needed for synthesis. The time directory contains files needed for timing simulation.

Conducting Functional Simulation

To run a functional simulation, you need to compile the design files and testbench, before you run the VSS simulation tool. This tutorial assumes that you uncompressed, untarred, and placed your setup files in /home/user/tutorial. If you placed your files elsewhere, replace your path appropriately in the following instructions.

The A1.5i XSI VSS functional simulation flow allows you to simulate instantiated XSI cells such as FDCE and CLKDLL. Additionally, by using the UNISIM simulation libraries, you can simulate and implement the GSR without impact to the design or testbench.

To conduct functional simulation, use the following steps.

1. Change directories to the /home/user/tutorial directory.
2. You conduct functional simulation by running the script func.script. Before running the script, open the file func.script in a text editor. Notice the following items in this file.
   - The files for simulation read in from the bottom up.
   - The testbench reads in last.
   - vhdlan uses the –i option. Always use vhdlan with the –i option. By default, vhdlan uses the –c option, which works only if your system uses a certain type of C compiler. If you want to use the –c option with vhdlan, refer to the Synopsys web site for the proper setup.
   - The contents of the func.sim file.

```
#!/bin/csh –f

rm –r WORK

mkdir WORK
```

```
vhdlan –i tenths.vhd
vhdlan –i smallcntr.vhd
vhdlan –i cnt60.vhd
vhdlan –i hex2led.vhd
vhdlan –i stmchine.vhd
vhdlan –i stopwatch.vhd
vhdlan –i testbenchf.vhd
vhdlsim –e commandf.txt overall
```

3. Run the functional simulation by typing the following command at the UNIX prompt.

   `./func.sim`

4. You get error messages because the VHDL code contains an instantiated component (OSC4) that does not have an underlying RTL behavioral description; you must make an RTL model for functional simulation. For more information on OSC4, refer to the *Libraries Guide* and the *Databook*. Use the UNISIM libraries to functionally simulate library cells instantiated from the XSI libraries. VHDL code that instantiates any XSI library cell must contain the following two lines.

   `library UNISIM;`

   `use UNISIM.all;`

   In general, you need these lines only in the files that contain instantiated XSI library cells (such as FDCE, RAM32X1S, and BUFG, for example). However, you can place the above two lines in every VHDL file in your design.

   The following example shows the modified stopwatch.vhd file.

```
library IEEE;
use IEEE.std_logic_1164.all;
library UNISIM;
use UNISIM.all;

entity stopwatch is
```

```
    port (    RESET : in STD_LOGIC;
   STRTSTOP : in STD_LOGIC;
  TENTHSOUT : out STD_LOGIC_VECTOR(9 downto 0);
    ONESOUT : out STD_LOGIC_VECTOR(6 downto 0);
    TENSOUT : out STD_LOGIC_VECTOR(6 downto 0);
    CLOCK: in STD_LOGIC
 );
end stopwatch;

architecture inside of stopwatch is

component BUFGDLL
    port (I: in STD_LOGIC; O: out STD_LOGIC);
end component;

component BUFG
    port (I : in STD_LOGIC;
  O : out STD_LOGIC);
end component;

component stmchine
    port (     CLK : in STD_LOGIC;
     RESET : in STD_LOGIC;
  STRTSTOP : in STD_LOGIC;
     CLKEN : out STD_LOGIC;
       RST : out STD_LOGIC
 );
end component;

component tenths
```

```
    port (     CLOCK : in STD_LOGIC;
      CLK_EN : in STD_LOGIC;
  ASYNC_CTRL : in STD_LOGIC;
    TERM_CNT : out STD_LOGIC;
       Q_OUT : out STD_LOGIC_VECTOR(9 downto 0));
end component;

component cnt60
    port (    CE : in STD_LOGIC;
     CLK : in STD_LOGIC;
     CLR : in STD_LOGIC;
  LSBSEC : out STD_LOGIC_VECTOR(3 downto 0);
  MSBSEC : out STD_LOGIC_VECTOR(3 downto 0));
end component;

component hex2led
    port (HEX : in STD_LOGIC_VECTOR(3 downto 0);
  LED : out STD_LOGIC_VECTOR(6 downto 0));
end component;

signal strtstopinv : STD_LOGIC;
signal oscout : STD_LOGIC;
signal clkint : STD_LOGIC;
signal clkenable : STD_LOGIC;
signal rstint : STD_LOGIC;
signal xcountout : STD_LOGIC_VECTOR(9 downto 0);
signal xtermcnt : STD_LOGIC;
signal cnt60enable : STD_LOGIC;
signal lsbcnt : STD_LOGIC_VECTOR(3 downto 0);
signal msbcnt : STD_LOGIC_VECTOR(3 downto 0);
```

```
begin

DLL:BUFGDLL port map(I=>CLOCK,O=>oscout);

CLOCKBUF:BUFG port map(I=>oscout,O=>clkint);

MACHINE:stmchine port map(CLK=>clkint,
                  RESET=>RESET,
                  STRTSTOP=>strtstopinv,
                  CLKEN=>clkenable,
                  RST=>rstint
 );

XCOUNTER:tenths port map(CLOCK=>clkint,
                 CLK_EN=>clkenable,
                 ASYNC_CTRL=>rstint,
                 TERM_CNT=>xtermcnt,
                 Q_OUT=>xcountout
                        );

sixty: cnt60 port map(CE=>cnt60enable,
              CLK=>clkint,
              CLR=>rstint,
              LSBSEC=>lsbcnt,
              MSBSEC=>msbcnt
                     );

lsbled:hex2led port map(HEX=>lsbcnt,
LED=>ONESOUT
```

```
                        );

msbled:hex2led port map(HEX=>msbcnt,
LED=>TENSOUT
                        );
cnt60enable<=xtermcnt and clkenable;
TENTHSOUT<=not(xcountout);
strtstopinv<=not(STRTSTOP);

end inside;
```

5. After you modify the stopwatch.vhd file, the func.sim file compiles with no errors. The simulation starts and displays the waveform viewer. The UNIX shell displays the vhdlsim prompt.

Synthesizing Your Design

In this section of the tutorial, you synthesize the design and create a place and routed NCD file. After creating the place and routed NCD file, you can optionally create a BIT file for downloading to the demo board, using bitgen and promgne, or the Hardware Debugger.

To synthesize your design, use the following instructions.

1. Create a compile script. You can modify the default compile script located in the A1.5i software. Copy the file $XILINX/template.fpga.script.4kex into your /home/user/tutorial directory, which contains the .synopsys_dc.setup and .synopsys_vss.setup files you created earlier.
2. Rename the file template.synopsys.dc.setup.4kes to run.script.
3. Open the file run.script in a text editor. The file run.script compiles only one VHDL file. You must comment out several lines not relevant to synthesis of this particular design.

   The following example shows the unmodified run.script file (using the file template.synopsys_dc.setup.4kex).

```
/* ================================================*/
/*    Sample Script for Synopsys to Xilinx Using     */
```

```
/*                  FPGA Compiler                    */
/*                                                   */
/*  Targets the Xilinx XC4028EX-3 and assumes a VHDL */
/*         source file by way of an example.         */
/*                                                   */
/*   For general use with XC4000E/EX architectures.  */
/*      Not suitable for use with XC3000A/XC5200     */
/*                  architectures.                   */
/* ================================================*/

/* ================================================ */
/* Set the name of the design's top-level module.    */
/* (Makes the script more readable and portable.)    */
/* Also set some useful variables to record the      */
/* designer and company name.                        */
/* ================================================ */
   TOP = <design_name>
                      /* ========================== */
                      /* Note: Assumes design file- */
                      /* name and entity name are   */
                      /* the same (minus extension) */
                      /* ========================== */
   designer = "XSI Team"
   company  = "Xilinx, Inc"
   part     = "4028expg299-3"
/* ================================================ */
/* Analyze and Elaborate the design file and specify */
/* the design file format.     */
/* ================================================ */
   analyze –format vhdl TOP + ".vhd"
```

```
                    /* ============================ */
                    /* You must analyze lower-level */
                    /* hierarchy modules here       */
                    /* ============================ */
   elaborate TOP
/* ================================================ */
/* Set the current design to the top level.         */
/* ================================================ */
   current_design TOP
/* ================================================ */
/* Set the synthesis design constraints.            */
/* ================================================ */
   remove_constraint –all
   /* Some example constraints */
   create_clock <clock_port_name> –period 50
   set_input_delay 5 –clock <clock_port_name> \
     { <a_list_of_input_ports> }
   set_output_delay 5 –clock <clock_port_name> \
     { <a_list_of_output_ports> }
   set_max_delay 100 –from <source> –to <destination>
   set_false_path –from <source> –to <destination>
/* ================================================ */
/* Indicate those ports on the top-level module that */
/* should become chip-level I/O pads. Assign any I/O */
/* attributes or parameters and perform the I/O      */
/* synthesis.                                       */
/* ================================================ */
   set_port_is_pad “*”
   /* Some example I/O parameters */
   set_pad_type –pullup <port_name>
```

```
   set_pad_type –no_clock all_inputs()
   set_pad_type –clock <clock_port_name>
   set_pad_type –exact BUFGS_F <hi_fanout_port_name>
   set_pad_type –slewrate HIGH all_outputs()
                  /* ============================== */
                  /* Note: Synopsys slew-control=  */
                  /* HIGH is the same as Xilinx's  */
                  /* slewrate=SLOW. Synopsys slew- */
                  /* control=LOW is same as Xilinx */
                  /* slewrate=FAST.                */
                  /* ============================== */
   insert_pads
/* ================================================ */
/* Synthesize and optimize the design               */
/* ================================================ */
   compile –boundary_optimization
/* ================================================ */
/* Write the design report files.                   */
/* ================================================ */
   report_fpga > TOP + ".fpga"
   report_timing > TOP + ".timing"
/* ================================================ */
/* Write out the design to a DB file. (Post compile) */
/* ================================================ */
   write –format db –hierarchy –output TOP + "_compiled.db"
/* ================================================ */
/* Replace CLBs and IOBs with gates.                */
/* ================================================ */
   replace_fpga
/* ================================================ */
```

```
/* Set the part type for the output netlist.     */
/* ================================================ */
   set_attribute TOP “part” –type string part
/* ================================================ */
/* Optional attribute to remove the FPGA Compiler’s  */
/* mapping structures from the design. This permits  */
/* The Xilinx design implementation tools to map the */
/* design instead.                                   */
/* ================================================ */
/* set_attribute find(design,”*”) “xnfout_write_map_symbols” \
     –type boolean FALSE  */
/* ================================================ */
/* Add any I/O constraints to the design.            */
/* ================================================ */
   set_attribute <port_name> “pad_location” \
     –type string “<pad_location>”
/* ================================================ */
/* Save design in XNF format as <design>.sxnf        */
/* ================================================ */
     –ungroup all –flatten
   write –format xnf –hierarchy –output TOP + “.sxnf”
/* ================================================ */
/* Write out the design to a DB. (Post replace_fpga) */
/* ================================================ */
   write –format db –hierarchy –output TOP + “.db”
/* ================================================ */
/* Write-out the timing constraints that were        */
/* applied earlier. (Note that any design hierarchy  */
/* needs to be flattened before the constraints are  */
/* written-out.)                                     */
```

```
/* ================================================ */
   write_script > TOP + ".dc"
/* ================================================ */
/* Call the Synopsys-to-Xilinx constraints translator*/
/* utility DC2NCF to convert the Synopsys constraints*/
/* to a Xilinx NCF file. You may like to view       */
/* dc2ncf.log to review the translation process.    */
/* ================================================ */
   sh dc2ncf TOP + ".dc"
/* ================================================ */
/* Exit the Compiler.                               */
/* ================================================ */
   exit
/* ================================================ */
/* Now run the Xilinx design implementation tools.  */
/* ================================================ */
```

The following example shows the modified run.script file (using the file run.script).

Before using this script, notice the following items.

- The design compiles from the bottom up.
- The VHDL file for the LogiBLOX counter tenths.vhd is not compiled.
- The VHDL file from the LogiBLOX tools is a simulation model.
- The proper type of output for place and route in A1.5i is an SXNF file when compiling an XC4000 design in FPGA Compiler.
- Dont_touch commands appear in the script. Whenever you instantiate a component from the XSI synthesis library, you must place a Dont_touch on the instance to prevent Synopsys from deleting or modifying the library cell.

```
/* ================================================*/
```

```
/*    Sample Script for Synopsys to Xilinx Using     */
/*                  FPGA Compiler                    */
/*                                                   */
/*  Targets the Xilinx XC4028EX-3 and assumes a VHDL */
/*         source file by way of an example.         */
/*                                                   */
/*   For general use with XC4000E/EX architectures.  */
/*      Not suitable for use with XC3000A/XC5200     */
/*                  architectures.                   */
/* ================================================*/

/* ================================================ */
/* Set the name of the design's top-level module.    */
/* (Makes the script more readable and portable.)    */
/* Also set some useful variables to record the      */
/* designer and company name.                        */
/* ================================================ */
   TOP = stopwatch
                      /* ========================== */
                      /* Note: Assumes design file- */
                      /* name and entity name are   */
                      /* the same (minus extension) */
                      /* ========================== */
   designer = "XSI Team"
   company  = "Xilinx, Inc"
   part     = "s05pc84-3"
/* ================================================ */
/* Analyze and Elaborate the design file and specify */
/* the design file format.     */
/* ================================================ */
```

```
analyze –format vhdl “smallcntr.vhd”
analyze –format vhdl “cnt60.vhd”
analyze –format vhdl “hex2led.vhd”
analyze –format vhdl “stmchine.vhd”
analyze –format vhdl TOP + “.vhd”
                    /* ============================ */
                    /* You must analyze lower-level */
                    /* hierarchy modules here       */
                    /* ============================ */
   elaborate TOP
   set_dont_touch “OSCILLATOR”
   uniquify
/* ================================================ */
/* Set the current design to the top level.         */
/* ================================================ */
   current_design TOP
/* ================================================ */
/* Set the synthesis design constraints.            */
/* ================================================ */
   remove_constraint –all
   /* Some example constraints */
/*   create_clock <clock_port_name> –period 50
   set_input_delay 5 –clock <clock_port_name> \
     { <a_list_of_input_ports> }
   set_output_delay 5 –clock <clock_port_name> \
     { <a_list_of_output_ports> }
   set_max_delay 100 –from <source> –to <destination>
   set_false_path –from <source> –to <destination> */
/* ================================================ */
/* Indicate those ports on the top-level module that */
```

```
/* should become chip-level I/O pads. Assign any I/O */
/* attributes or parameters and perform the I/O      */
/* synthesis.                                        */
/* ================================================= */
   set_port_is_pad "*"
   /* Some example I/O parameters */
/*   set_pad_type –pullup <port_name>
   set_pad_type –no_clock all_inputs()
   set_pad_type –clock <clock_port_name>
   set_pad_type –exact BUFGS_F <hi_fanout_port_name>
   set_pad_type –slewrate HIGH all_outputs() */
                  /* ============================== */
                  /* Note: Synopsys slew-control=  */
                  /* HIGH is the same as Xilinx's  */
                  /* slewrate=SLOW. Synopsys slew- */
                  /* control=LOW is same as Xilinx */
                  /* slewrate=FAST.                */
                  /* ============================== */
   insert_pads
/* ================================================= */
/* Synthesize and optimize the design                */
/* ================================================= */
   compile –boundary_optimization
/* ================================================= */
/* Write the design report files.                    */
/* ================================================= */
/*   report_fpga > TOP + ".fpga"
   report_timing > TOP + ".timing" */
/* ================================================= */
/* Write out the design to a DB file. (Post compile) */
```

```
/* ================================================ */
   write –format db –hierarchy –output TOP + “_compiled.db”
/* ================================================ */
/* Replace CLBs and IOBs with gates.                */
/* ================================================ */
   replace_fpga
/* ================================================ */
/* Set the part type for the output netlist.     */
/* ================================================ */
   set_attribute TOP “part” –type string part
/* ================================================ */
/* Optional attribute to remove the FPGA Compiler’s  */
/* mapping structures from the design. This permits  */
/* The Xilinx design implementation tools to map the */
/* design instead.                                  */
/* ================================================ */
/* set_attribute find(design,”*”) “xnfout_write_map_symbols” \
     -type boolean FALSE  */
/* ================================================ */
/* Add any I/O constraints to the design.           */
/* ================================================ */
/*   set_attribute <port_name> “pad_location” \
   –type string “<pad_location>” */
/* ================================================ */
/* Save design in XNF format as <design>.sxnf       */
/* ================================================ */
   ungroup –all –flatten
   write -format xnf -hierarchy -output TOP + “.sxnf”
/* ================================================ */
/* Write out the design to a DB. (Post replace_fpga) */
```

```
/* ================================================ */
   write –format db –hierarchy –output TOP + ".db"
/* ================================================ */
/* Write-out the timing constraints that were       */
/* applied earlier. (Note that any design hierarchy  */
/* needs to be flattened before the constraints are  */
/* written-out.)                                    */
/* ================================================ */
/*   write_script > TOP + ".dc" */
/* ================================================ */
/* Call the Synopsys-to-Xilinx constraints translator*/
/* utility DC2NCF to convert the Synopsys constraints*/
/* to a Xilinx NCF file. You may like to view       */
/* dc2ncf.log to review the translation process.    */
/* ================================================ */
/*   sh dc2ncf TOP + ".dc" */
/* ================================================ */
/* Exit the Compiler.                               */
/* ================================================ */
/*   exit */
/* ================================================ */
/* Now run the Xilinx design implementation tools.  */
/* ================================================ */
```

4. Synthesize the design by first starting Design Analzyer. Type the following command in the directory that contains the .synopsys_dc.setup file for this design.

   design_analyzer &

   This brings up the Design Analyzer GUI.

5. When the GUI appears, run the script run.script by selecting Execute Script in the Setup menu. A pop-up window appears where you can select the script run.script to run.

If the script runs successfully, the script stops and creates an SXNF file. If an error occurs, check the following.

- Make sure you set up the .synopsys_dc.setup file correctly.
- Make sure you compiled the XDW synthesis libraries for the version of Synopsys you use.
- Make sure that the paths referenced in the .synopsys_dc.setup file exist. Refer to the "Using Common Setup Procedures" section for more information.
- Make sure you run the design_analyzer & command in the directory that contains the .synopsys_dc.setup file when you start Design Analyzer.

6. Take the SXNF file produced by Design Analyzer and place and route the design for timing simulation using the following script.

```
#!/bin/csh –f

ngdbuild –p 4003EPC84-4 stopwatch.sxnf

map stopwatch.ngd

par stopwatch.ncd stopwatch_r.ncd

ngdanno stopwatch_r.ncd

ngd2vhdl stopwatch_r.nga
```

Optionally, you can place and route the SXNF files using the A1.5i GUI. Refer to the *Quick Start Guide Tutorial* for more information about using the GUI for place and route.

Conducting Timing Simulation

To perform timing simulation, use NGD2VHDL to create an SDF file and structural VHDL file, along with a testbench. The script file timing.sim performs the timing simulation.

Run timing.sim in the directory that contains the .synopsys_vss.setup file you created earlier.

Open the script file timing.sim in a text editor and notice the following.

- vhdlan uses with the –i option. Always use vhdlan with the –i option. By default, vhdlan uses the –c option, which works only if your system uses a certain type of C compiler. If you want to use

the –c option with vhdlan, refer to the Synopsys web site for the proper setup.

- The -sdf_top option is specified first when invoking when vhdldbx or vhdlsim because.
- The contents of file timing.sim here.

```
#!/bin/csh –f

rm –r WORK

mkdir WORK

vhdlan –i stopwatch_r.vhd

vhdlan –i testbencht.vhd

vhdlsim –sdf_top /testbenchf/uut –sdf \
stopwatch_r.sdf –e commandt.txt overall
```

Close the timing.sim file in the text editor. Run the timing simulation, which produces a waveform view similar the functional simulation, by typing the following command at the UNIX prompt.

```
./timing.sim
```

When the script runs, if you get error and warning messages about "Bad Regions" or undefined libraries, make sure you set up the simulation libraries for A1.5i XSI VSS correctly. Make sure the paths in the .synopsys_vss.setup file point to paths which exist in your setup. For further information on setup, refer to the "Using Common Setup Procedures" section.

Virtex VHDL Alliance FPGA Express v2.1/VSS Tutorial

This tutorial familiarizes you with the Alliance FPGA Express v2.1/VSS flow, presenting common tasks such as locking pins and setting slew rate.

Getting Ready for this Tutorial

To use this tutorial, you need the software described in the "Using Common Setup Procedures" section. If you want to target a device other than Virtex, reference the directories that apply to that family (for example, the XDW libraries have separate directories for XC4000XL, Spartan, and Virtex). Use synlibs with the exact die-speed

target desired. You need FPGA Express and VSS v1997.01 or better for this tutorial.

Setting Up for the Virtex VHDL Alliance FPGA Express V2.1/VSS Tutorial

To use this tutorial, ensure installation of your Xilinx and Synopsys software and know where the software resides on your system. If you use a version of Synopsys newer than v1997.01, you must re-compile the XSI XDW and simulation libraries. Refer to the setup instructions in the "Setting up for FPGA Compiler" section.

For this tutorial, recompile only the libraries related to synthesizing a Virtex device (if using a version of Synopsys newer than v1997.01), and ensure compilation of the Virtex XDW DesignWare libraries. The A1.5i simulation libraries come in two parts, a functional simulation part and a timing simulation part. The A1.5i XSI VHDL functional simulation libraries are called UNISIM. The A1.5i XSI VHDL timing simulation libraries are called SIMPRIM libraries. The SIMPRIM library is a VITAL simulation library.

Compile the Spartan XDW DesignWare libraries, XDW simulation libraries, LogiBLOX simulation libraries, UNISIM simulation libraries, and SIMPRIM libraries. All of these libraries contain compile scripts which let you compile these libraries in the $XILINX area. However, to use these scripts, you need write permissions to the $XILINX area. If you do not have write permissions, make a local copy of the $XILINX/synopsys/libraries directory and use the following instructions, but instead of changing directories to $XILINX/synopsys/libraries, change directories to your local copy of that area.

Use the following instructions to compile the libraries Maker sure you have followed the directions listed in the "Using Common Setup Procedures" section.

1. Change directories to the $XILINX/synopsys/libraries/sim/src/simprims directory.
2. Type the following command at the UNIX prompt.

   ```
   ./analyze.csh
   ```

3. Change directories to the $XILINX/synopsys/libraries/sim/src/unisims directory.

4. Type the following command at the UNIX prompt.

   `./analyze.csh`

5. In the empty directory you created earlier in the "Using Common Setup Procedures" section, copy the file vrtxxsivhdle.tar.Z. Uncompress and untar this file.

6. Create a .synospys_dc.setup file and a .synopsys_vss.setup using the templates provided in the $XILINX/.synopsys/examples area and the instructions listed in the "Using Common Setup Procedures" section.

Conducting Functional Simulation

To run a functional simulation, you need to compile the design files and testbench, before you run the VSS simulation tool. This tutorial assumes that you uncompressed, untarred, and placed your setup files in /home/user/tutorial. If you placed your files elsewhere, replace your path appropriately in the following instructions.

The A1.5i XSI VSS functional simulation flow allows you to simulate instantiated XSI cells such as FDCE and CLKDLL. Additionally, by using the UNISIM simulation libraries, you can simulate and implement the GSR without impact to the design or testbench.

To conduct functional simulation, use the following steps.

1. Change directories to the /home/user/tutorial directory.

2. You conduct functional simulation by running the script func.script. Before running the script, open the file func.script in a text editor. Notice the following items in this file.

   - The files for simulation read in from the bottom up.
   - The testbench reads in last.
   - vhdlan uses the –i option. Always use vhdlan with the –i option. By default, vhdlan uses the –c option, which works only if your system uses a certain type of C compiler. If you want to use the –c option with vhdlan, refer to the Synopsys web site for the proper setup.
   - The contents of the func.sim file.

   ```
   #!/bin/csh –f

   rm –r WORK
   ```

```
mkdir WORK
vhdlan –i tenths.vhd
vhdlan –i smallcntr.vhd
vhdlan –i cnt60.vhd
vhdlan –i hex2led.vhd
vhdlan –i stmchine.vhd
vhdlan –i stopwatch.vhd
vhdlan –i testbenchf.vhd
vhdlsim –e commandf.txt overall
```

3. Run the functional simulation by typing the following command at the UNIX prompt.

   `./func.sim`

4. You get error messages because the VHDL code contains an instantiated component (OSC4) that does not have an underlying RTL behavioral description; you must make an RTL model for functional simulation. For more information on OSC4, refer to the *Libraries Guide* and the *Databook*. Use the UNISIM libraries to functionally simulate library cells instantiated from the XSI libraries. VHDL code that instantiates any XSI library cell must contain the following two lines.

   `library UNISIM;`

   `use UNISIM.all;`

   In general, you need these lines only in the files that contain instantiated XSI library cells (such as FDCE, RAM32X1S, and BUFG, for example). However, you can place the above two lines in every VHDL file in your design.

   The following example shows the modified stopwatch.vhd file.

```
library IEEE;
use IEEE.std_logic_1164.all;
library UNISIM;
use UNISIM.all;

entity stopwatch is
```

```
    port (    RESET : in STD_LOGIC;
   STRTSTOP : in STD_LOGIC;
  TENTHSOUT : out STD_LOGIC_VECTOR(9 downto 0);
    ONESOUT : out STD_LOGIC_VECTOR(6 downto 0);
    TENSOUT : out STD_LOGIC_VECTOR(6 downto 0);
   CLOCK : in STD_LOGIC
 );
end stopwatch;

architecture inside of stopwatch is

component BUFGDLL
port(I: in STD_LOGIC; O: out STD_LOGIC);
end component;

component BUFG
    port (I : in STD_LOGIC;
  O : out STD_LOGIC);
end component;

component stmchine
    port (     CLK : in STD_LOGIC;
     RESET : in STD_LOGIC;
  STRTSTOP : in STD_LOGIC;
     CLKEN : out STD_LOGIC;
       RST : out STD_LOGIC
 );
end component;
```

```
component tenths
    port (     CLOCK : in STD_LOGIC;
      CLK_EN : in STD_LOGIC;
  ASYNC_CTRL : in STD_LOGIC;
    TERM_CNT : out STD_LOGIC;
       Q_OUT : out STD_LOGIC_VECTOR(9 downto 0));
end component;

component cnt60
    port (    CE : in STD_LOGIC;
     CLK : in STD_LOGIC;
     CLR : in STD_LOGIC;
  LSBSEC : out STD_LOGIC_VECTOR(3 downto 0);
  MSBSEC : out STD_LOGIC_VECTOR(3 downto 0));
end component;

component hex2led
    port (HEX : in STD_LOGIC_VECTOR(3 downto 0);
  LED : out STD_LOGIC_VECTOR(6 downto 0));
end component;

signal strtstopinv : STD_LOGIC;
signal oscout : STD_LOGIC;
signal clkint : STD_LOGIC;
signal clkenable : STD_LOGIC;
signal rstint : STD_LOGIC;
signal xcountout : STD_LOGIC_VECTOR(9 downto 0);
signal xtermcnt : STD_LOGIC;
signal cnt60enable : STD_LOGIC;
signal lsbcnt : STD_LOGIC_VECTOR(3 downto 0);
```

```
signal msbcnt : STD_LOGIC_VECTOR(3 downto 0);

begin

DLL:BUFGDLL port map(I=>CLOCK, O=>oscout);

CLOCKBUF:BUFG port map(I=>oscout,O=>clkint);

MACHINE:stmchine port map(CLK=>clkint,
                  RESET=>RESET,
                  STRTSTOP=>strtstopinv,
                  CLKEN=>clkenable,
                  RST=>rstint
 );

XCOUNTER:tenths port map(CLOCK=>clkint,
                 CLK_EN=>clkenable,
                 ASYNC_CTRL=>rstint,
                 TERM_CNT=>xtermcnt,
                 Q_OUT=>xcountout
                        );

sixty: cnt60 port map(CE=>cnt60enable,
              CLK=>clkint,
              CLR=>rstint,
              LSBSEC=>lsbcnt,
              MSBSEC=>msbcnt
                     );

lsbled:hex2led port map(HEX=>lsbcnt,
```

```
LED=>ONESOUT
                        );

msbled:hex2led port map(HEX=>msbcnt,
LED=>TENSOUT
                        );
cnt60enable<=xtermcnt and clkenable;
TENTHSOUT<=not(xcountout);
strtstopinv<=not(STRTSTOP);

end inside;
```

5. After correcting the stopwatch.vhd file, recompile the func.sim file. The simulation starts and displays the waveform viewer. The UNIX shell displays the vhdlsim prompt.

Synthesizing Your Design

In this section of the tutorial, you synthesize the design and create a place and routed NCD file. After creating the place and routed NCD file, you can optionally create a BIT file for downloading to the demo board, using bitgen and promgne, or the Hardware Debugger. For more information about using the FPGA Express GUI, refer to the FPGA Express on-line help.

Use the following instructions to synthesize your design.

1. Create an FPGA Express Project, entering source files and specifying a target device (Virtex) as shown in the following two figures.

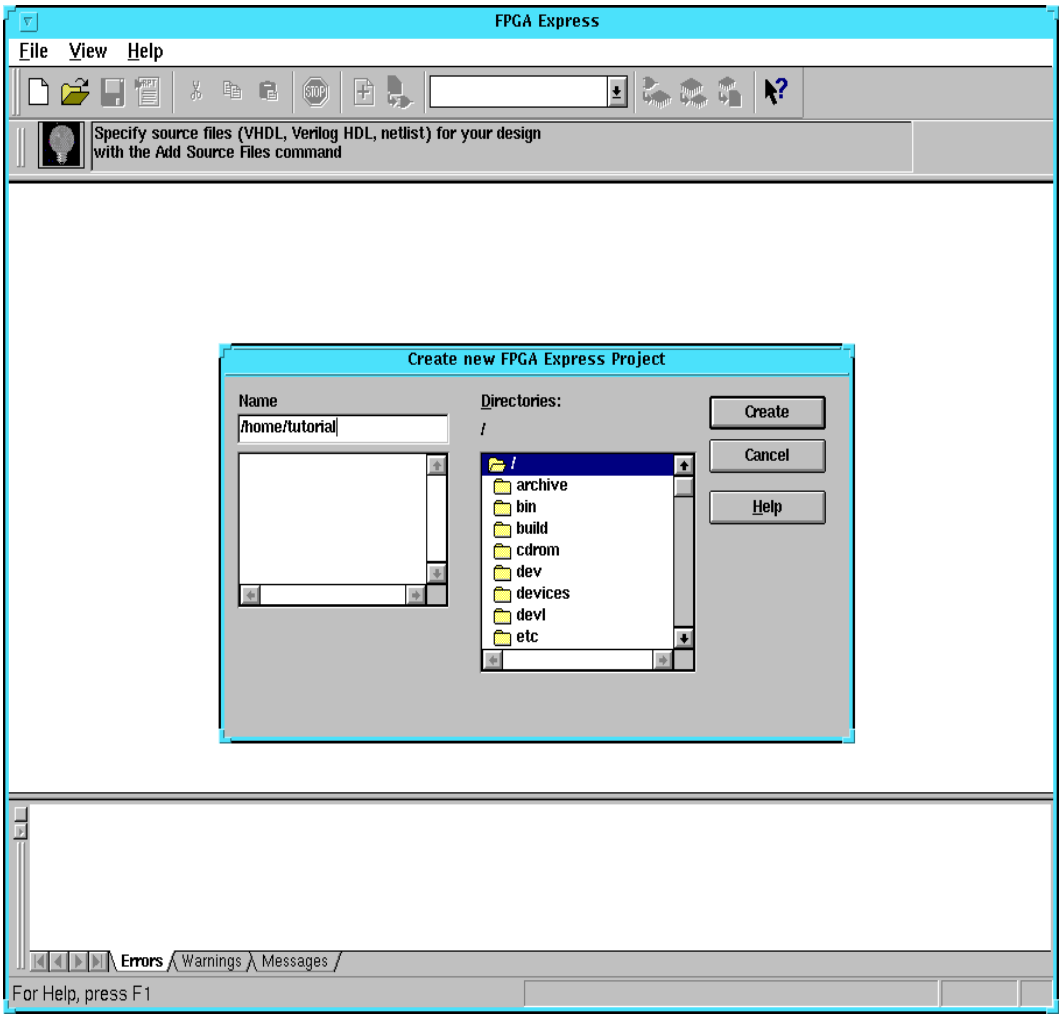

Figure 1-16 Create Project Window for FPGA Express VHDL (Virtex)

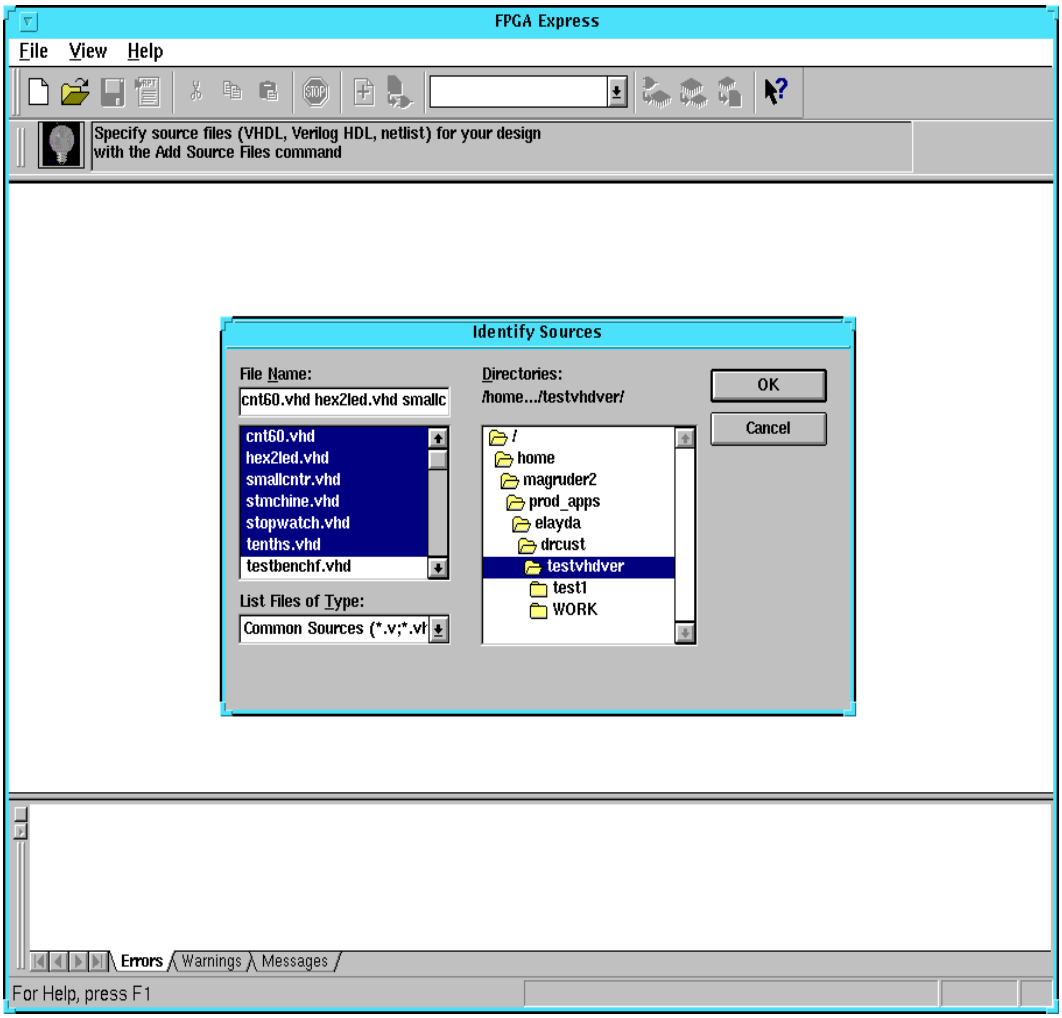

Figure 1-17 Identifying Sources for anFPGA Express Project (Virtex)

2. Select the top level entity and select a target device, as the following figure illustrates.

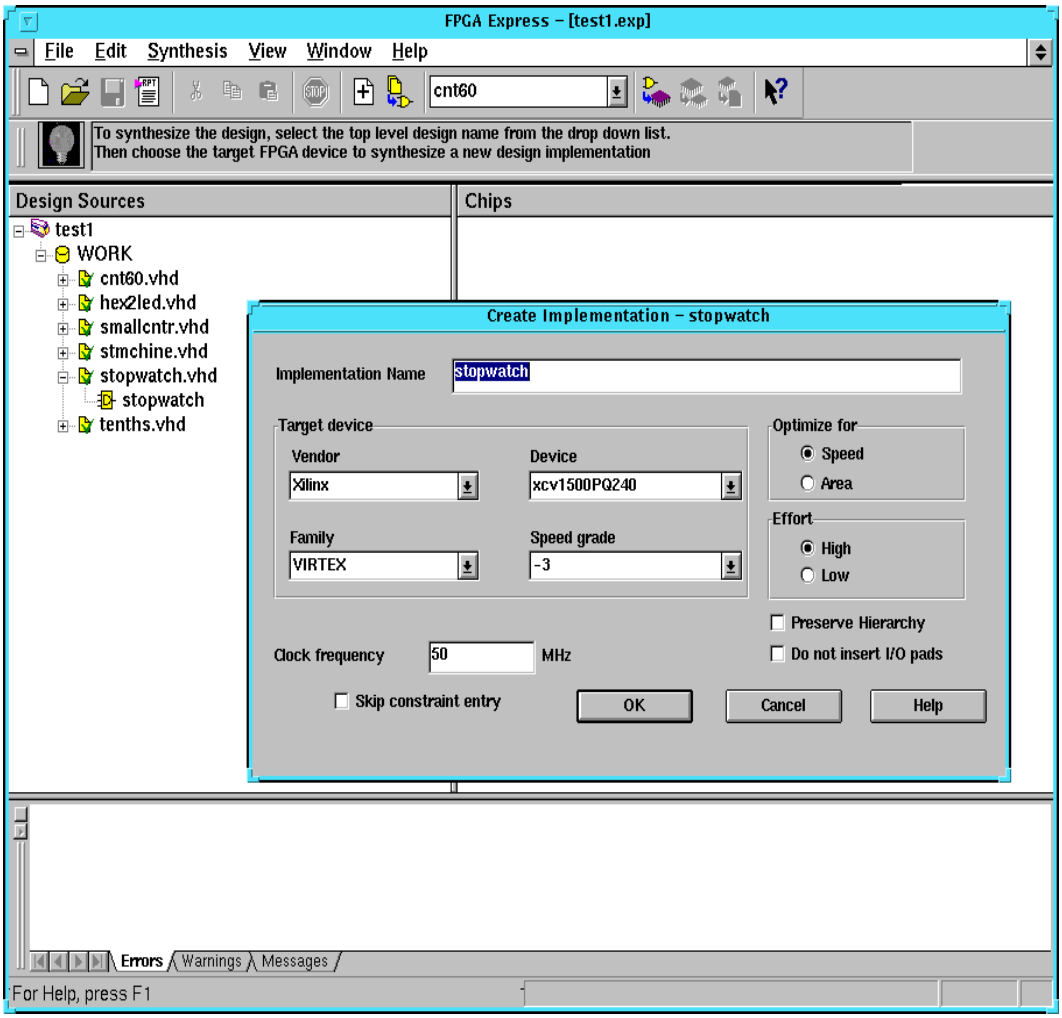

Figure 1-18 Top Level Entity and Target Device Selection (Virtex)

3. An implementation appears in the right-hand window. Select the implementation and then press the Optimize button on the tool bar, as in the following figure.

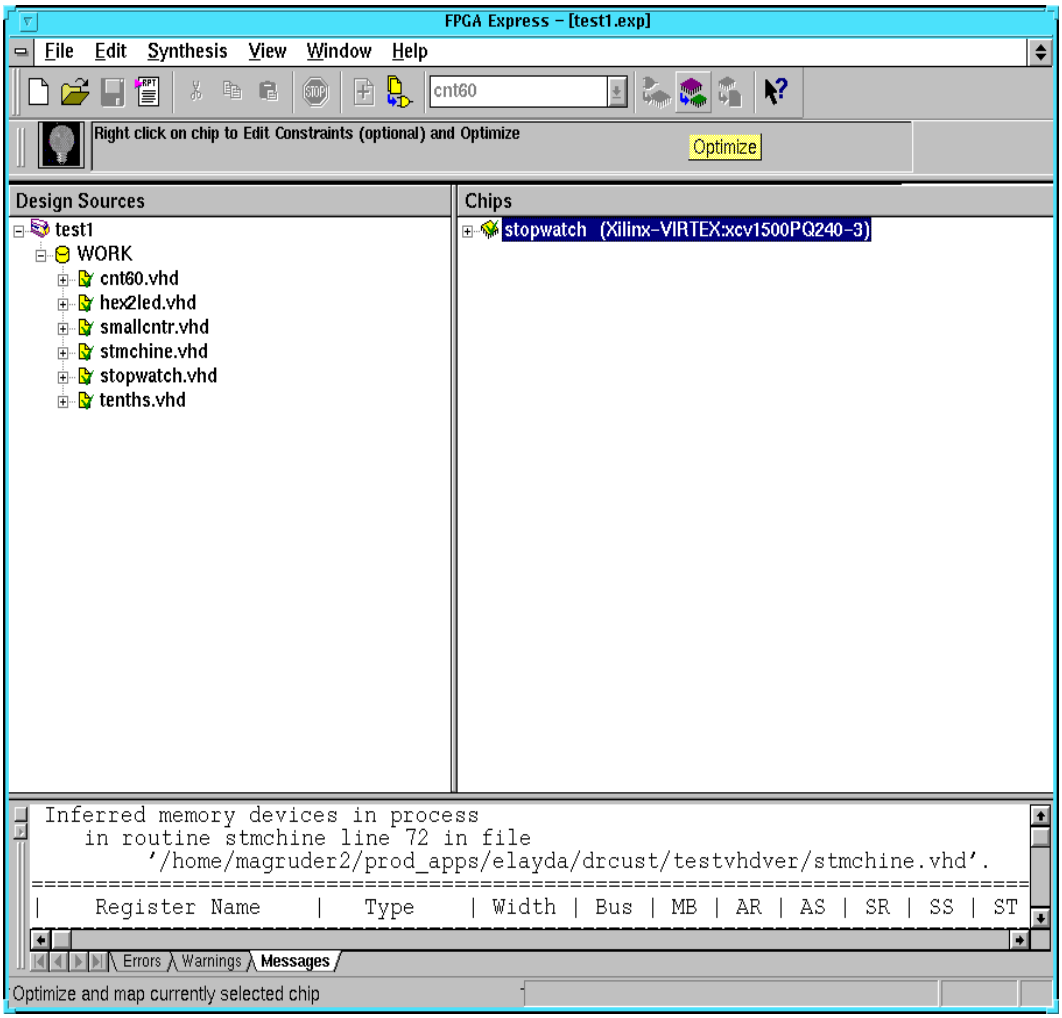

Figure 1-19 Optimized Implementation in FPGA Express (Virtex)

4. Select the optimized design and write out the netlist by pressing the Export Netlist button on the toolbar, as shown in the following illustration.

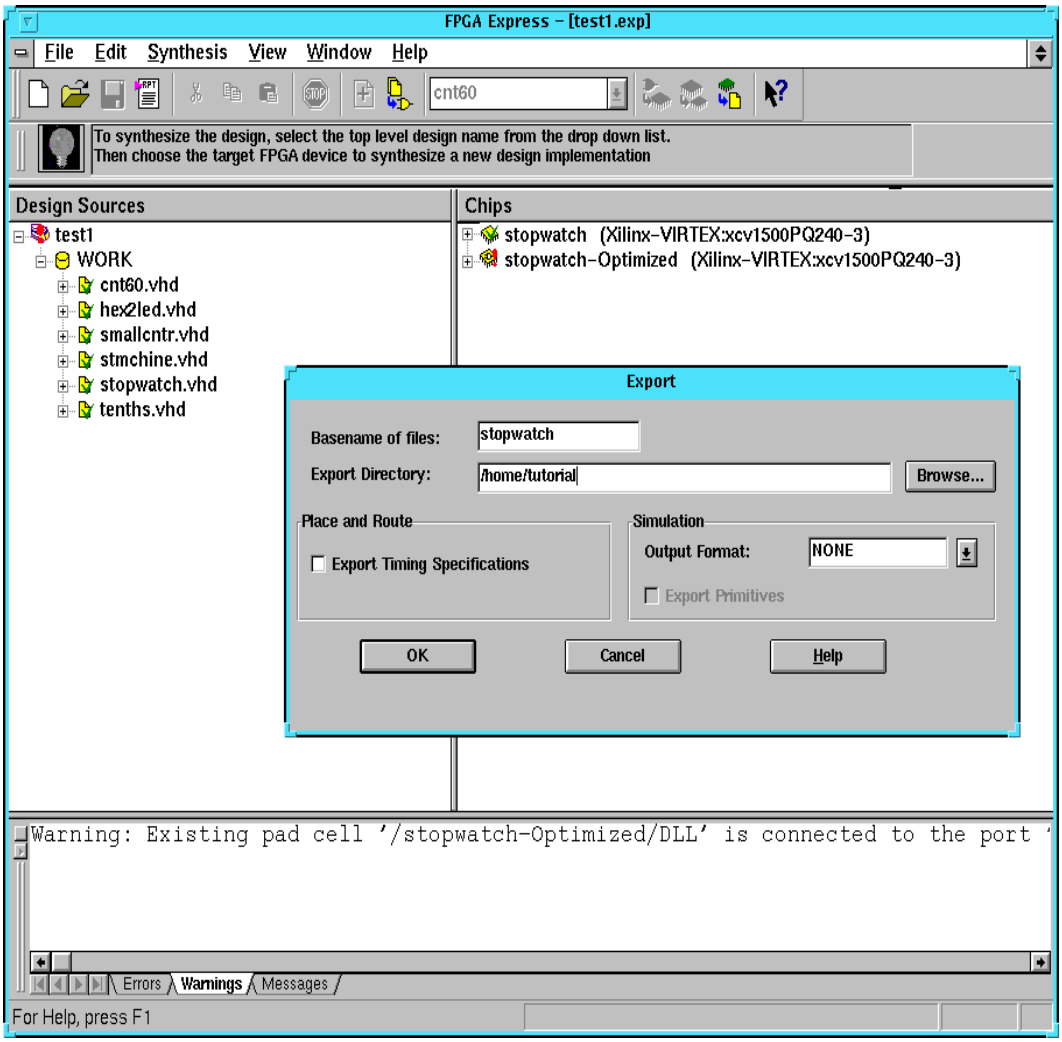

Figure 1-20 FPGA Express Optimized Netlist (VHDL/Virtex))

5. Take the EDIF file produced by FPGA Express and place and route the design for timing simulation using the following script.

```
#!/bin/csh –f

ngdbuild –p v50pq240-4 stopwatch.edf

map stopwatch.ngd
```

```
par stopwatch.ncd stopwatch_r.ncd

ngdanno stopwatch_r.ncd

ngd2vhdl stopwatch_r.nga
```

Optionally, you can place and route by using the A1.5i GUI. Refer to the *Quick Start Guide Tutorial* for more information about using the GUI for place and route.

Conducting Timing Simulation

To perform timing simulation, use NGD2VHDL to create an SDF file and structural VHDL file, along with a testbench. The script file timing.sim performs the timing simulation.

Run timing.sim in the directory that contains the .synopsys_vss.setup file you created earlier.

Open the script file timing.sim in a text editor and notice the following items.

- vhdlan uses w the –i option. Always use vhdlan with the –i option. By default, vhdlan uses the –c option, which works only if your system uses a certain type of C compiler. If you want to use the -c option with vhdlan, refer to the Synopsys web site for the proper setup.
- Specify the –sdf_top option first when invoking vhdldbx or vhdlsim during a timing simulation.
- The contents of the file timing.sim.

```
#!/bin/csh –f

rm –r WORK

mkdir WORK

vhdlan –i stopwatch_r.vhd

vhdlan –i testbencht.vhd

vhdlsim –sdf_top /testbenchf/uut \
–sdf stopwatch_r.sdf –e commandt.txt overall
```

Close the timing.sim file in the text editor. Run the timing simulation, which produces a waveform view similar to the functional simulation, by typing the following command at the UNIX prompt.

```
./timing.sim
```

When the script runs, if you get error or warning messages about "Bad Regions" or undefined libraries, make sure you set up the simulation libraries for A1.5i XSI VSS correctly. Make sure that the paths in the .synopsys_vss.setup file point to paths which exist in your setup. For further information, refer to the "Using Common Setup Procedures" section.