Chapter 1

Synplify/ModelSim Tutorial

This tutorial shows you how to use Synplicity's Synplify (VHDL / Verilog) for XC4000E / EX / XL / XV designs using MTI's ModelSim for simulation. It guides you through a typical FPGA HDL-based design procedure using a design of a runner's stopwatch called Watch. This tutorial contains the following sections.

- "Design Description"
- "Before Beginning the Tutorial"
- "Tutorial Installation"
- "Creating the Tenths LogiBLOX Component"
- "RTL Simulation"
- "Synthesizing the Design Using Synplicity"
- "Implementing the Watch Design"
- "Timing Simulation"

Design Description

Throughout this tutorial, the design is referred to as Watch which is a design for a runner's stop watch. The tutorial assumes that you have a working knowledge of VHDL and / or Verilog.

The Watch design is a counter that counts up from 0 to 59, then resets to zero, and starts over. There are two external inputs and three external outputs in the completed design. The system clock is an internally generated signal produced by OSC4, the internal oscillator in XC4000 devices.

The Watch design inputs, outputs, and macros are summarized below.

Inputs

- **STRTSTOP**—The start / stop button of the stopwatch. This is an active-low signal that must be depressed then released to start or stop the counting.
- **RESET**—Forces the signals TENSOUT and ONESOUT to be “00” after it has been stopped.

Outputs

- **TENSOUT[6:0]**—7-bit bus which represents the tens-digit of the stopwatch value. This is viewable on the 7-segment LED display of the Xilinx demo board.
- **ONESOUT[6:0]**—Similar to TENSOUT bus above, but represents the one-digit of the stopwatch value.
- **TENTHSOUT[9:0]**—10-bit bus which represents the tenths-digit of the stopwatch value. This bus is one-hot encoded. The output is displayed to the LED bar.

Macros

The top level of the Watch design consists of the following functional blocks.

- **OSC4**—Internal oscillator macro which is used to generate the clock signal.
- **STWATCH**—A statemachine that controls starting, stopping, and clearing the counters. (One-hot encoded)
- **TENTHS**—A LogiBLOX 10-bit one-hot counter macro which outputs the Tenths digit as 10-bit one-hot value.
- **CNT60**—A Counter that outputs Ones and Tens digits as 4-bit binary values. Counts 0 to 59 (decimal).
- **HEX2LED**—Converts 4-bit values of Ones and Tens to 7-segment LED format.
- **DEBUG_CKT**—A Circuit that supports synchronous debugging and readback using the Hardware Debugger.

Before Beginning the Tutorial

Before you begin this tutorial, set up your system to use the Synplicity, Model Technology, and Xilinx software as follows.

1. Install the following software.
   - Xilinx Development System 2.1i
   - Synplicity Synplify 5.1.4 or later
   - Model Technology ModelSim EE/PE 5.2 or later
2. Verify that your system is properly configured. Consult the release notes and installation notes that came with your software package for more information.

Tutorial Installation

The Watch tutorial file is available for download from the Xilinx Web site at http://www.xilinx.com/support/techsup/tutorials.

Tutorial Directory and Files

The tutorial directory and tutorial files needed to complete the design are provided for you. Some files are not present since you will create them in later steps. The following table lists the contents of the tutorial directories.

Directory	Description
synplify_tut/vhdl/src	VHDL source and script files
synplify_tut/vhdl/watch_4ke	VHDL solutions directory for XC4003E-PC84
synplify_tut/vhdl/watch	VHDL Tutorial Directory
synplify_tut/verilog/src	Verilog source and script files
synplify_tut/verilog/watch_4ke	Verilog solutions directory for XC4003E-PC84
synplify_tut/verilog/watch	Verilog Tutorial Directory

VHDL Design Files

Watch is the top level design. The tutorial uses the following VHDL files.

- watch.vhd
- stmchine.vhd
- smallcntr.vhd
- cnt60.vhd
- hex2led.vhd
- debug_ckt.vhd
- tenths.vhd (functional RTL simulation only)
- testbench.vhd (VHDL testbench for simulation)

Note: The Tenths one-hot counter is a LogiBLOX macro.

Verilog Design Files

Watch is the top level design. The tutorial uses the following Verilog files.

- watch.v
- stmchine.v
- smallcntr.v
- cnt60.v
- hex2led.v
- debug_ckt.v
- glbl.v
- tenths.v (functional RTL simulation only)
- testfixture.v (Verilog test fixture for simulation)

Note: The Tenths one-hot counter is a LogiBLOX macro.

Script Files

The following script files are provided to automate the steps in this tutorial.

- rtl_sim.do
- stim.do
- synth.tcl
- time_sim.do

Simulation Models for MTI

To simulate Xilinx designs with ModelSim, you need the following simulation libraries which you must compile.

- **UNISIMS Library**—The Unisim library is used for behavioral (RTL) simulation with instantiated components in the netlist, and for post-synthesis simulation. The VHDL library is VITAL compliant, and it also adds the device start-up components ROC, ROCBUF, TOC, TOCBUF, and STARTBUF, for simulation. The Verilog library has separate libraries by device family: UNI3000, UNISIMS (for 4000E/L/X, SPARTAN/XL, VIRTEX/E), UNI5200, UNI9000.
- **LogiBLOX Library**—The LogiBLOX library is used for designs containing LogiBLOX components, during pre-synthesis (RTL), and post-synthesis simulation. VITAL VHDL simulation only. Verilog uses the SIMPRIM libraries.
- **SIMPRIM Library**—The SIMPRIM library is used for post Ngdbuild (gate level functional), post-Map (partial timing), and post-place-and-route (full timing) simulations. This library is architecture independent, and supports VHDL and Verilog.

For detailed instructions on compiling these simulation libraries, see the instructions in Xilinx Solution # 2561 which is available at http://www.xilinx.com/techdocs/2561.htm.

After compiling the libraries, notice that ModelSim creates a file called modelsim.ini. View this file and notice that the upper portion defines the locations of the compiled libraries. When doing a simulation, the modelsim.ini file must be provided either by copying the file directly to the directory where the HDL files are to be compiled and the simulation is to be run, or by setting the MODELSIM environment variable to the location of your master .ini file. You must set this variable since the ModelSim installation does not initially declare the path for you. For UNIX, type the following.

```
setenv MODELSIM /path/to/the/modelsim.ini
```

Copying the Tutorial Files

In this tutorial, “watch” is the name of the directory where the tutorial will be performed. For the VHDL tutorial, copy all the files from the **`/synplify_tut/vhdl/src`** directory to the **`/synplify_tut/vhdl/watch`** directory. For the Verilog tutorial, copy all the files from the **`/synplify_tut/verilog/src`** directory to the **`/synplify_tut/verilog/watch`** directory.

Creating the Tenths LogiBLOX Component

Since the Watch design contains a LogiBLOX macro, you must create it before performing RTL simulation or implementation. While creating the LogiBLOX component, you will create a behavioral simulation netlist for RTL simulation, as well as the implementation netlist and an instantiation netlist. To create the LogiBLOX component, follow these steps.

1. To invoke the LogiBLOX GUI, type **`lbgui`** at the UNIX prompt, or if you are using a PC, click on the LogiBLOX icon in the Xilinx Program group.

   The LogiBLOX GUI and Setup dialog box open.

2. In the Vendor tab of the Setup dialog box, select B(I) for bus type and “Other” for vendor.

   Normally at this point you would either select the bus notation for parenthesis B(I) if your format is EDIF or select the bus notation for angle-brackets B<I> if your format is XNF. But for this tutorial, select B(I) since Synplify 5.0 and above supports EDIF.

3. In the Project Directory tab, use the Browse button or type the path to specify the project directory where you wish to write files.

4. In the Device Family tab, choose the xc4000e family since the design is targeted for the Xilinx demoboard.

5. In the Options tab, set the following options.

   VHDL tutorial settings.

   - Simulation Netlist: Behavioral VHDL netlist
   - Component Declaration: VHDL Template

- Implementation Netlist: NGC File

Verilog tutorial settings.

- Simulation Netlist: Structural Verilog netlist
- Component Declaration: Verilog Template
- Implementation Netlist: NGC File

6. Click OK to close the Setup dialog box.

Note: If you are familiar with LogiBLOX, notice that the implementation netlist extension is now .ngc. This was introduced in the Xilinx Alliance 1.5 software. For more details, read *Xilinx Solution # 3904* which is available at http://www.xilinx.com/techdocs/3904.htm.

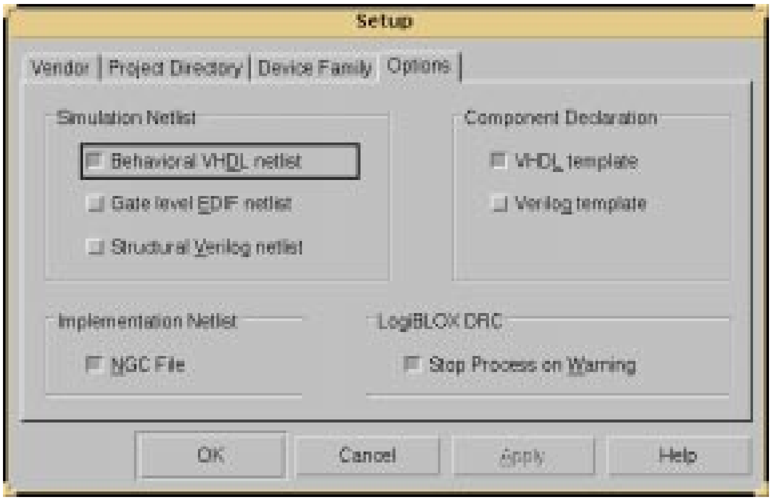
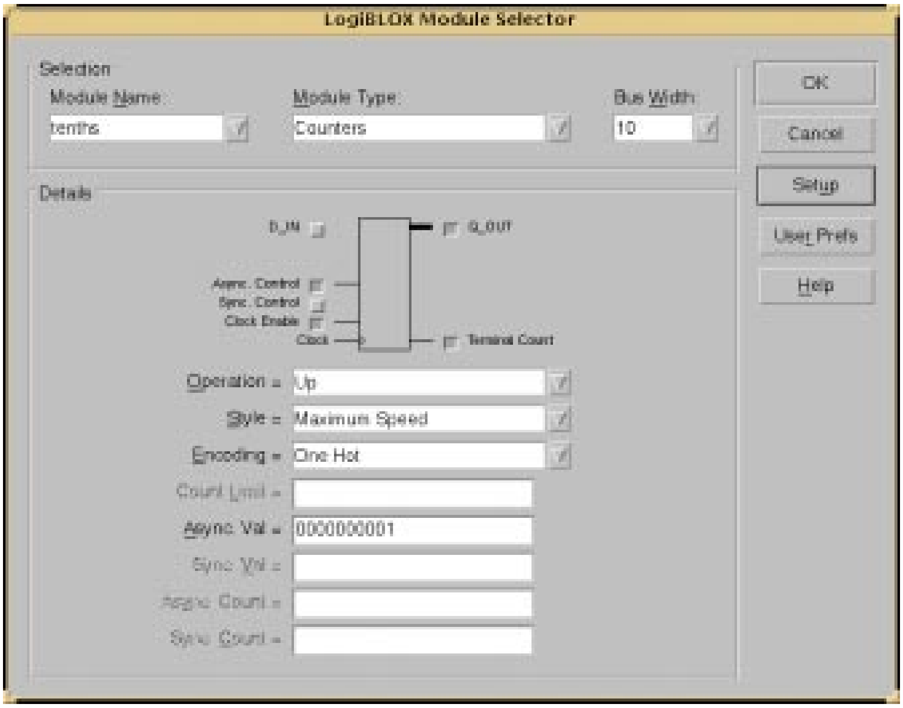


Figure 1-1 LogiBLOX Setup Dialog Box

7. In the LogiBLOX Module Selector dialog box, set the following options.
   - Module Type: Counters
   - Module Name: tenths (Typed by the user)
   - Bus Width: 10 (Optionally typed by the user)
   - Operation = Up
   - Deselect D_IN

- Select Async. Control and Terminal Count
- By default, the following is selected: Clock Enable, Q_OUT
- Style = Maximum Speed
- Encoding = One Hot
- Async. Val = 2#0000000001#



Module Name: tenths
Module Type: Counters
Bus Width: 10
Operation = Up
Style = Maximum Speed
Encoding = One Hot
Count Limit =
Async. Val = 0000000001
Sync. Val =
Async. Count =
Sync. Count =

Figure 1-2 LogiBLOX Module Selector

8. Click OK.

   LogiBLOX generates the following output Files.

   - logblox.ini - shows the LogiBLOX options used
   - logiblox.log - log file of the LogiBLOX GUI messages window
   - tenths.mod - LogiBLOX Modules options file
   - tenths.ngc - implementation netlist

- tenths.vhi - VHDL declaration / instantiation template
- tenths.vhd - VHDL behavioral simulation netlist
- tenths.vei - Verilog declaration / instantiation template
- tenths.v - Verilog structural simulation netlist

RTL Simulation

The Watch design contains an XC4000E library part, OSC4. This component represents the on-chip oscillator that generates nominal clock frequencies of 8 MHz, 500 KHz, 16 KHz, 490 Hz, and 15 Hz. The Watch design uses the 15-Hz output from this component when targeted for XC4000E family designs. The clock output from OSC4 is buffered through a BUFG global clock buffer to minimize clock skew.

XC4000E family devices have eight on-chip clock buffers, one BUFGP (primary global buffer), and one BUFGS (secondary global buffer) in each corner of the device. Although it is possible to use them for other purposes, BUFGPs are best used to route externally-generated clock signals. BUFGSs have more flexibility, and can be used to route any large fan-out net, even if it is internally sourced. A BUFG symbol can represent either type of buffer, and allows the implementation software to choose the type of global buffer that is best in each situation. BUFG also facilitates design retargeting to other Xilinx device families, since it can represent any type of global buffer in any family. The BUFG in the Watch design is substituted for a BUFGS during design implementation, because the clock is generated internally by the on-chip oscillator. See the *Xilinx Libraries Guide* and the *Xilinx Programmable Logic Data Book* for more information on global clock buffers for Xilinx devices.

For simulation purposes, it is not necessary to create a clock for the Watch testbench, since the design already has the OSC4 component generating a 15Hz signal. However, one problem with this is that the OSC4 component also has a 8MHz pin, and therefore the OSC4 simulation model has to simulate the toggling of the 8MHz pin. This means that it takes an extraordinary amount of time for ModelSim to simulate a 15Hz clock signal.

Note: For Verilog simulation, the OSC4 model has a timescale precision of 100ps. The timescale value is set to 1 ps because that is the basic unit used in the NCD and speed files. To make a transition to the first edge of the 15Hz clock, which is at 3.33E10 ps (.0333 seconds), requires 3.33E10 / 100 = 333 million simulation events. The OSC4.v UNISIM model is located at **$XILINX/verilog/src/unisims**.

Therefore, a clock is defined in the testbench/testfixture that clocks much faster, and this clock is selected through a multiplexer to force its values onto the CLK signal, bypassing the OSC4 F15 clock.

Note: *Xilinx Solution # 3767* contains further information on the use of the OSC4 with VHDL simulation for ModelSim. This is available at http://www.xilinx.com/techdocs/3767.htm for review.

Note: For Verilog simulation, all behaviorally described (inferred) and instantiated registers should have a common signal which asynchronously sets or resets the registers. Toggling the global set/reset emulates the Power-On-Reset of the FPGA. If you do not do this, the flip-flops and latches in your simulation enter an unknown state. The general procedure for specifiying global set/reset or global reset during a pre-Ngdbuild Verilog UNISIMS simulation involves defining the global reset signals with the $XILINX/verilog/src/glbl.v module. The VHDL UniSims library contains the ROC, ROCBUF, TOC, TOCBUF, and STARTUP cells to assist in VITAL VHDL simulation of the global set/reset and tri-state signals. However, Verilog allows a global signal to be modified as a wire in a global module, and, thus does not contain these cells.

Copying Source Files to the Functional Simulation Directory

VHDL

For the VHDL tutorial, copy the following files into the **/synplify_tut/vhdl/watch/func** directory.

- smallcntr.vhd
- cnt60.vhd
- hex2led.vhd
- debug_ckt.vhd
- tenths.vhd
- watch.vhd

- stmchine.vhd
- testbench.vhd
- rtl_sim.do

Verilog

For the Verilog tutorial, copy the following files into the **`/synplify_tut/verilog/watch/func`** directory.

- smallcntr.v
- cnt60.v
- hex2led.v
- debug_ckt.v
- tenths.v
- watch.v
- stmchine.v
- testfixture.v
- rtl_sim.do
- glbl.v

Starting ModelSim

If you are using the PC, invoke the simulator by selecting **`Programs`** → **`Model Tech`** → **`ModelSim`** from the Start menu. For UNIX workstations, type the following at the prompt.

```
vsim -i &
```

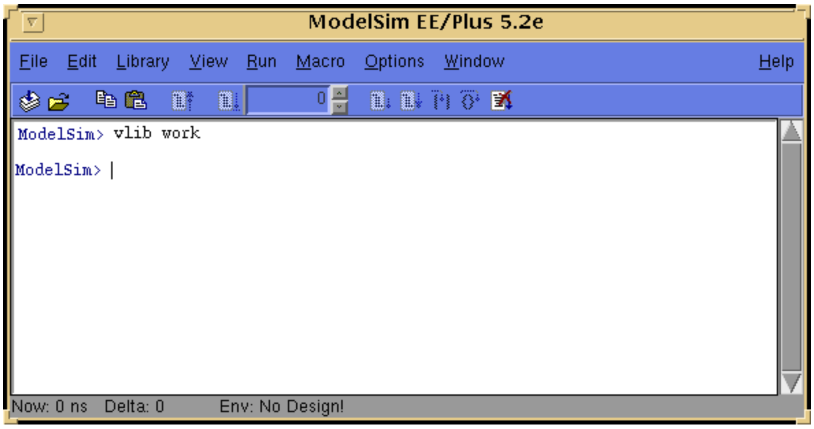
Set the project directory using the **`File`** → **`Change Directory`** menu command and select **`watch/func`**.

Creating the Work Directory

Before compiling the VHDL / Verilog source files, you must create a directory for use as a library. Type the following at the ModelSim prompt.

```
vlib work
```

This action is echoed in the Transcript window as shown in the following figure.



```
ModelSim> vlib work

ModelSim> |
```

```
Now: 0 ns   Delta: 0        Env: No Design!
```

Figure 1-3 MTI Transcript Window

Compiling the Source Files

VHDL

Since Xilinx Unified library components are instantiated within the VHDL source code, the UNISIM simulation models must be provided for the OSC4, BUFG, MD0, MD1, IBUF, OBUF, RDBK, and STARTUP components. The following lines must be added in the files watch.vhd and debug_ckt.vhd.

```
library unisim;
use unisim.vcomponents.all;
```

As a key point, Synplify supports translate_off / translate_on directives. Translate_off instructs Synplify not to read in and synthesize anything after the translate_off directive, until a translate_on directive is found. In this tutorial, these directives are used to declare the simulation library without removing the declaration for synthesis.

You also need to comment out the following lines in watch.vhd and debug_ckt.vhd (using “--” at the beginning of each line).

```
library xc4000;
use xc4000.components.all;
```

The xc4000 library is used solely by Synplify. The library contains pre-defined black boxes for Xilinx primitives so you can manually instantiate them into your design.

The Vcom command compiles VHDL code for use with Vsim RTL simulation. Also, to enhance simulation, both Synplify and ModelSim support VHDL ‘93. The -93 switch is used to enable support for 1076-93. Type the following at the ModelSim prompt.

vcom -93 -explicit smallcntr.vhd

vcom -93 cnt60.vhd tenths.vhd

vcom -93 debug_ckt.vhd hex2led.vhd stmchine.vhd

vcom -93 watch.vhd testbench.vhd

The -explicit is used to compile smallcntr.vhd since there is a definition of “=” in the std_logic_1164 and std_logic_unsigned libraries that are declared for the entity. The option resolves resolution conflicts in favor of explicit function.

Verilog

You need to comment out the following line in the watch.v file (using “//” at the beginning of the line).

```
‘include “/path/to/synplify/lib/xilinx/xc4000.v”
```

The xc4000 library is used solely by Synplify. The library contains pre-defined black boxes for Xilinx primitives so you can manually instantiate them into your design.

Also, comment out the Tenths module declaration within watch.v since the simulation model for this component was generated with the creation of LogiBLOX component. The following declaration is used as a place-holder for synthesis since the NGC was created earliear, so it is unnecessary to synthesize the tenths module.

```
module tenths (CLK_EN, CLOCK, ASYNC_CTRL, Q_OUT,
TERM_CNT)

/* synthesis black_box */;

input CLK_EN, CLOCK, ASYNC_CTRL;

output [9:0] Q_OUT;

output TERM_CNT;
```

```
endmodule
```

The Vlog command compiles Verilog code for use with Vsim RTL simulation. Type the following at the ModelSim prompt.

```
vlog testfixture.v watch.v stmchine.v hex2led.v
debug_ckt.v cnt60.v smallcntr.v tenths.v glbl.v
```

Invoke the Simulator

For the VHDL tutorial, type the following at the ModelSim prompt to invoke the ModelSim simulator.

vsim overall

For the Verilog tutorial, type the following at the ModelSim prompt to invoke the ModelSim simulator.

vsim -L simprims_ver -L unisims_ver test glbl

Since Xilinx Unified library components are instantiated within the Verilog source code, the UNISIM simulation models must be provided for the OSC4, BUFG, MD0, MD1, IBUF, OBUF, RDBK, and STARTUP components. Also, notice that the library, simprim_ver, is listed as well. For LogiBLOX generated components, Ngd2ver is used to generate a structural Verilog netlists to facilitate functional simulation. The structural netlist contains SIMPRIM library components which are mapped to the simprims_ver library.

Note: The file, rtl_sim_xilinx.do, runs the above commands; you can run it instead of executing each command. The file is located in the src directory and you can copy it into the watch / func directory. To execute the file, type the following at the ModelSim prompt.

```
do rtl_sim.do
```

Optionally, you may launch the macro via the **Macro** → **Execute Macro** menu command.

Running the Simulation

To perform simulation using ModelSim, follow these steps.

1. To view all the ModelSim debug windows, type the following.

   view *

2. Add the signals from the selected region in the Structure window to the Wave and List windows by issuing the following commands at the ModelSim prompt.

   add wave *

   add list *

3. In the Structure window, notice that VHDL design units are indicated by squares and Verilog modules are indicated by circles. You can expand and collapse regions of hierarchy by clicking on the (+) and (-) notations.

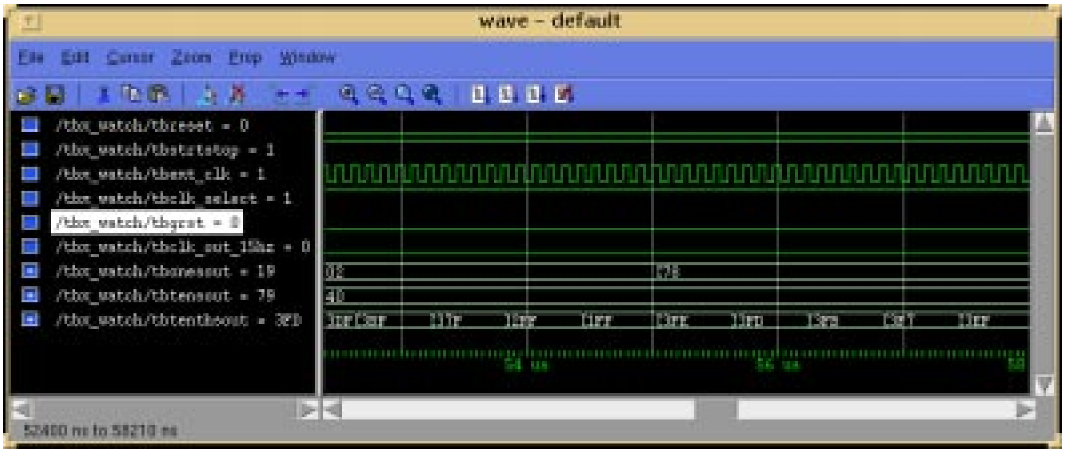
4. To run the simulation for a specified amount of time at the ModelSim prompt, type the following.

   ```
   run 100000 ns
   ```

   The simulation output is displayed in the Wave window. You may have to zoom in/out to view the waveforms.

5. In the Wave window, try adding or removing cursors with the **`Cursor`** → **`Add | Remove`** menu command. When multiple cursors are drawn, ModelSim adds a delta measurement showing the time difference between the cursors. The selected cursor is drawn as a solid line and the values at the cursor location are shown to the right of the signal name. All other cursors are drawn as dotted lines. If you cannot see the signal value next to the signal name, select the bar separating the signal names from the waveforms and drag it to the right.

Note: The above commands have been combined into a macro file called stim.do. You can execute them at the ModelSim prompt.



```
/tbx_watch/tbreset = 0
/tbx_watch/tbstrtstop = 1
/tbx_watch/tbext_clk = 1
/tbx_watch/tbclk_select = 1
/tbx_watch/tbgrst = 0
/tbx_watch/tbclk_out_15hz = 0
/tbx_watch/tbonesout = 19
/tbx_watch/tbtensout = 79
/tbx_watch/tbtenthsout = 3FD
52400 ns to 58210 ns
```

Figure 1-4 Simulation Output in Wave Window

Synthesizing the Design Using Synplicity

In this section, you synthesize the design using two methods, Synplify GUI and Synplify batch mode.

Synthesizing the Design Using the Synplify GUI

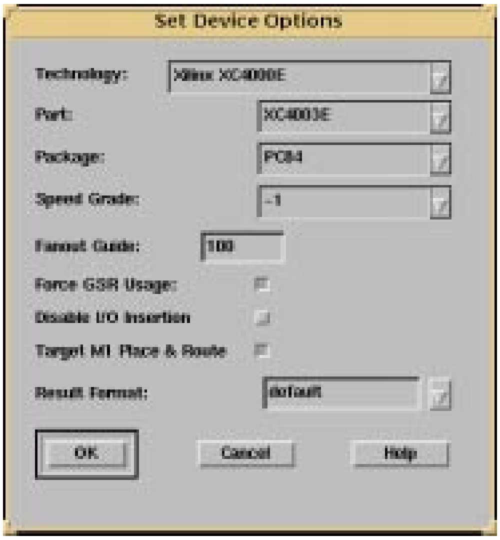
1. Invoke the Synplify Graphical User Interface as follows.
   - UNIX users, type **`synplify &`** at the prompt.
   - Windows NT users, double-click on the Synplify icon in the Synplicity program group.
   - Windows 95 users, **`Choose Programs`** → **`Synplicity`** → **`Synplify`** from the Start button.

   This launches the Synplicity Synplify main window. Projects are typically set up interactively from the Project Window, which is the main window in Synplify. The Project window lists your source files, result file, and target information. You can open a new project with the **`File`** → **`New`** menu, or by clicking the P button on the Synplify button bar.
2. To specify the target technology from the menu, select **`Target`** → **`Set Device Options`**.

3. If you are going to download this design to the demoboard, choose the following settings.

   - Technology: Xilinx 4000E
   - Part: XC4003E
   - Package: PC84

   Leave all other synthesis options at their default settings.



Technology: Xilinx XC4000E
Part: XC4003E
Package: PC84
Speed Grade: -1
Fanout Guide: 100
Force GSR Usage:
Disable I/O Insertion
Target M1 Place & Route
Result Format: default

Figure 1-5 Set Device Options Dialog Box

4. The Source Files portion of the Synplify main window is where you specify input design files. To specify your input files, press the right mouse button in the Source Files list box, and select **`Add Source Files`**. You can also add files to the Project Window by dragging and dropping files from File Manager or Explorer.

5. You need to uncomment the library declarations in the HDL files to read the synthesis library cell definitions. Also, you need to modify the following line in watch.v to the correct location of xc4000 library from Synplify.

   ```
   ‘include “/path/to/synplify/lib/xilinx/xc4000.v”
   ```

6. For the VHDL tutorial, change the order of the VHDL input source files to the following order.

smallcntr.vhd cnt60.vhd hex2led.vhd stmchine.vhd
debug_ckt.vhd watch.vhd

For the Verilog tutorial, change the order of the Verilog input source files to the following order.

smallcntr.v cnt60.v hex2led.v stmchine.v debug_ckt.v watch.v

By default, Synplify scans the input source files from top to bottom of the Source File list box. For Verilog, the top level module is the last module it finds that is not instantiated somewhere in the design. For VHDL, it is the last architecture of the last entity that is compiled. Therefore, the recommendation for both languages is to put your top level as the last object in the last design file in your list of input source files. To specify a different top-level design block, simply move a different file to the bottom of the Source Files list box (select and drag with the left mouse button). In a Tcl script, you can also choose a different top level design by using the **`set_option -top_module`** command.

7. Define the pinout using a SDC file.

   It is highly recommended that you let the automatic placement and routing program, Par, define the pinout. Pre-assigning locations to the I/Os can sometimes degrade the performance of the place and route tools. However, it is usually necessary, at some point, to lock the pinout of a design so that it can be integrated into a board design. The initial pinout should be defined by running the place and route tools without pin assignments, then locking down the I/O placement so that it reflects the locations chosen by the tools. As a general rule, inputs should be placed on the left side of the die, and outputs on the right. I/O in the tutorial must be assigned pin locations so that the Watch design can function in the Xilinx demonstration boards. Since the design is fairly simple, these pin assignments do not adversely affect the ability of Par to place and route the design completely.

   You will use a constraints file to lock down selected signals to designated pins. A constraints file (.sdc file) is used for user timing constraints and vendor specific attribute constraints. For ease of use and saving time, all other pins have been locked in the SDC file.

Note: Pin assignments can also made directly into the HDL. Please read *Xilinx Solution # 2379* which is available at http://www.xilinx.com/techdocs/2379.htm for instructions.

8. To lock the RESET signal to pin 28 and STRTSTOP signal to pin 18, edit the partially completed SDC file, watch.sdc, that is provided for you. Add the following lines.

   define_attribute RESET xc_loc “P28”

   define_attribute STRTSTOP xc_loc “P18”

9. Save and add the constraints file to the Source Files list box in the Synplify Project Window.
10. Click the Add button, and follow the menu **`List Files of Type`** → **`Constraint Files (.sdc)`**.
11. Highlight **`watch.sdc`** and click the OK button.

Note: VHDL is case-insensitive but Tcl is case-sensitive. You must match signal names as they appear in the HDL source code.

12. Enable the Symbolic FSM Compiler checkbox on the Synplify Project Window.

    The Symbolic FSM Compiler can be enabled for your entire design by clicking the Symbolic FSM Compiler checkbox on the Synplify Project Window. If this check box is set, then no changes of any kind need to be made to the source code. Set this option since Synplify automatically recognizes and extracts the state machines in the design, and performs the Symbolic FSM Compiler optimizations.

13. At this point, all the options are set and you are ready to synthesize the design. Click the RUN button.

    Synplify displays *DONE!* when synthesis is complete. Synplify displays *ERRORS!* if there are user errors in your source file. If there are warnings (but no errors), Synplify will display *DONE(warnings).*

    If Synplify reports only warnings, and no errors, it does complete the mapping to your target device. Nevertheless, it is important to investigate any warnings messages from Synplify, before continuing your design process.

Notice, there is a warning in the debug_ckt file that states "*Port direction mismatch between component and entity*" that needs investigation.

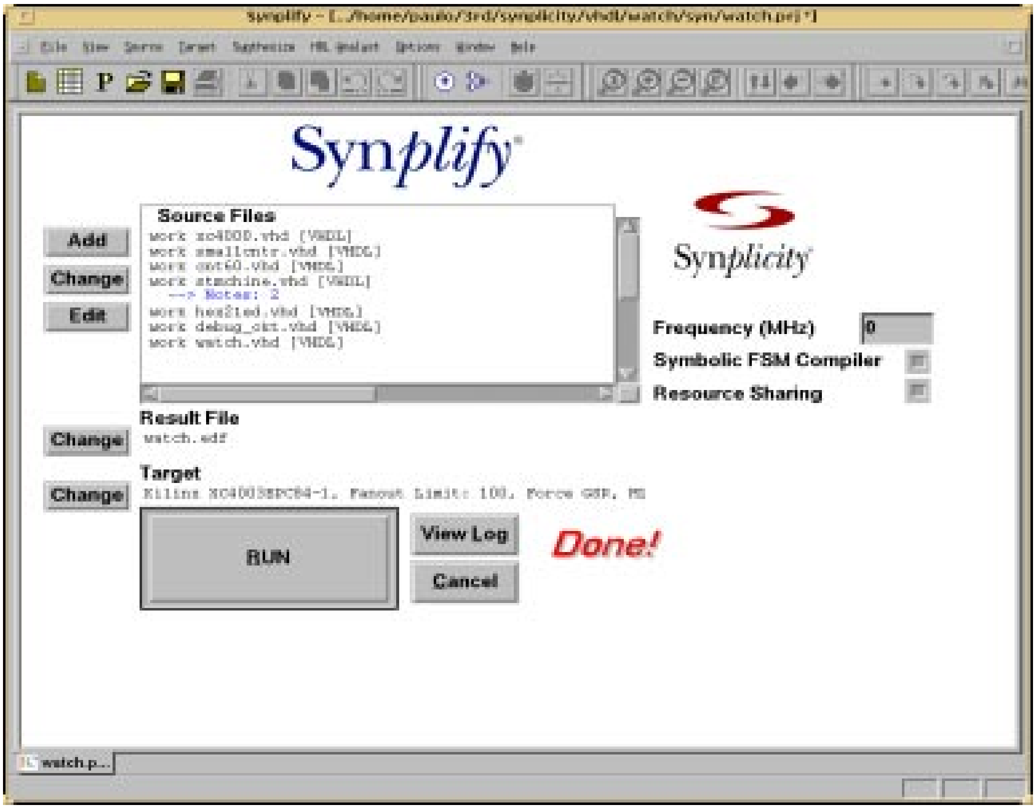
To correct the problem, the user will need to modify the Synplify Xilinx macros library file, xc4000.v or xc4000.vhd, located at $SYNPLICITY/lib/xilinx/ on UNIX or C:\synplcty\lib\xilinx on the PC. Copy this file to your working directory and add it to to the Source Files list box in the Synplify Project Window. Edit the port declaration of MD0 to be an output as opposed to an input.

Re-run synthesis.

14. When synthesis is done, Synplify creates the result file with the filename specified in the user interface.

    Double-click the left mouse button on the result filename to see it displayed in the Synplify Editing Window.

15. Click the View Log button to see the log file, including Resource Usage Reports.


synplify - [../home/paulo/3rd/synplicity/vhdl/watch/syn/watch.prj *]
File View Source Target Synthesize HDL Analyst Options Window Help
Synplify
Synplicity
Add
Change
Edit
Change
Change
RUN
View Log
Cancel
Done!
watch.p...

```
Source Files
work xc4000.vhd [VHDL]
work smallcntr.vhd [VHDL]
work cnt60.vhd [VHDL]
work stmchine.vhd [VHDL]
  --> Notes: 2
work hex2led.vhd [VHDL]
work debug_ckt.vhd [VHDL]
work watch.vhd [VHDL]

Result File
watch.edf

Target
Xilinx XC4003EPC84-1, Fanout Limit: 100, Force GSR, MS

Frequency (MHz)          0
Symbolic FSM Compiler
Resource Sharing
```

Figure 1-6 Synplify Window

16. If you wish to view how the design was synthesized, select **`HDL Analyst`** → **`RTL View`**.

Synthesizing the Design Using the Synplify Batch Mode

Synplicity has extended the Tcl language with some synthesis commands so Tcl can be used as a scripting language to run Synplify. Tcl scripts have a .tcl extension and are executed in Synplify from the **`File`** → **`Run Tcl Script`** menu command. As part of the 5.0 release, batch mode operation is standard with a floating license. Please contact Synplicity to request this feature. The script file, synthesis.tcl, has been created for you to illustrate this feature. To run Synplify in batch mode, type the following.

```
synplify -batch synth.tcl
```

This executes the Tcl script file and exits when finished. The files watch.edf and watch.srr file are created. The flow through Synplify is fully defined by the commands in the script. The script can use any Synplify command including all Tcl and shell commands that can be found in the path.

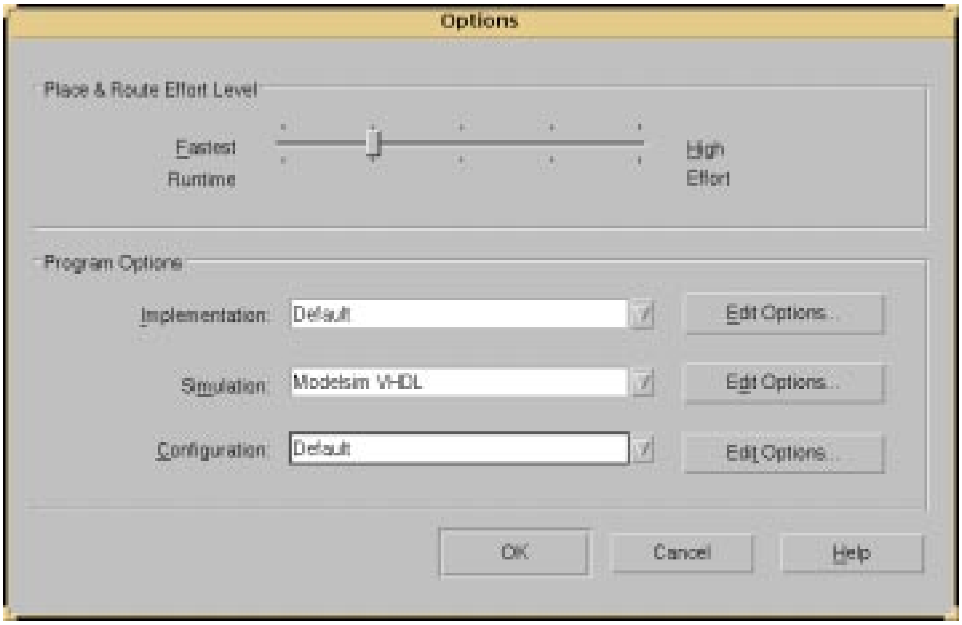
Implementing the Watch Design

To implement the Watch design, refer to the *Xilinx Design Manager Tutorial*. You need the following files for implementation.

- watch.edf
- tenths.ngc

When you implement the Watch design with the Xilinx Design Manager, you need to set the Implementation Options Timing Template to ModelSim VHDL for the VHDL tutorial to produce the time_sim.vhd file, or ModelSim Verilog for the Verilog tutorial to produce the time_sim.v file, and time_sim.sdf for timing simulation. To set these options, follow these steps.

1. In the Design Manger's Implement window, select Options under the Design pull-down menu, to open the Options dialog box.
2. In the Program Option Template, set Simulation to ModelSim VHDL for the VHDL tutorial or ModelSim Verilog for the Verilog tutorial.



Implementation: Default
Simulation: Modelsim VHDL
Configuration: Default

Figure 1-7 Design Manager Implement Dialog Box

3. Proceed with the Design Manager Tutorial.

Note: Although not included in this tutorial, it is possible to run a post-Ngdbuild and post-Map simulation, which may be helpful for debugging the design.

Timing Simulation

VHDL

For VHDL tutorial, you need two files from the Xilinx core tools.

- time_sim.vhd
- time_sim.sdf

Now that the HDL netlist has been resolved into primitives, you need to modify the testbench configuration. The Unisim library was referenced since the pre-synthesis netlist contained instantiated Xilinx macros.

To perform timing simulation, follow these steps.

1. Copy time_sim.vhd, time_sim.sdf, and testbench.vhd to the following directory.

   `/synplify_tut/vhdl/watch/time`

2. Launch ModelSim, and navigate to the following directory.

   `/synplify_tut/vhdl/watch/time`

3. Create the work directory.

   `vlib work`

4. View the testbench.vhd file and notice that there are two sections at the bottom.

   The first section is for RTL functional simulation and is already being used. Comment this out by using the "--" at the beginning of each line starting with the line "configuration overall of TBX_WATCH is" and ending with the line "end overall," in the RTL simulation section.

5. In the Post P&R simulation section, uncomment the lines by removing the "--" symbols. Again for the line beginning with "configuration overall of TBX_WATCH is" and ending with "end overall."

6. After editing the testbench.vhd, save the changes and exit.

7. Compile the VHDL source files and the testbench.

   vcom time_sim.vhd testbench.vhd

8. Read in the SDF file for timing simulation.

   `vsim -sdftyp uut=time_sim.sdf overall`

   Alternatively, select **`File`** → **`Load New Design`**. Highlight "overall" in the Design Unit window. Click the Add button. To apply the timing data, click on the SDF tab on the Load Design window. Click the Add button. Browse and select for the time_sim.sdf file. Type **`uut`** in the Apply to Region field and click the Load button.

9. View the necessary debugging windows by typing the following command at the ModelSim prompt.

   view wave signals source

10. View and add the signals of the design to the waveform window.

11. At the ModelSim prompt type.

    ```
    run 100000 ns
    ```

12. Right click in the waveform window and zoom in. Another way to zoom in, press and hold the middle mouse button and draw a square around the area to zoom in on. After simulating, you can then zoom in and view the delay from the clock edge to the TENSOUT, ONESOUT, and TENTHSOUT output change.

Note: The above commands have been combined into a macro file, time_sim.do, and can be executed at the ModelSim prompt.

Verilog

For Verilog tutorial, you need two files from the Xilinx core tools.

- time_sim.v
- time_sim.sdf

To perform timing simulation, follow these steps.

1. Copy time_sim.v, time_sim.sdf, and testfixture.v to the following directory.

   ```
   /synplify_tut/verilog/watch/time
   ```

2. Launch ModelSim, and navigate to the following directory.

   ```
   /synplify_tut/verilog/watch/time
   ```

3. Create the work directory.

   ```
   vlib work
   ```

4. Compile the Verilog file and the testfixture.

   ```
   vlog testfixture.v time_sim.v
   ```

5. Read in the SDF file for timing simulation. Ngd2ver automatically writes out a directive, $sdf_annotate, within the time_sim.v file. This directive specifies the appropriate SDF file to use in conjunction with the produced netlist. So, it unnecessary for the user to specify an option for ModelSim to read the SDF.

   ```
   vsim -L simprims_ver test
   ```

   Now that the HDL netlist has been resolved into primitives, we must provide the simulation models to the SIMPRIM library.

6. View the necessary debugging windows by typing the following command at the ModelSim prompt.

   ```
   view wave signals source
   ```

7. View and add the signals of the design to the waveform window.
8. At the ModelSim prompt type.

   ```
   run 100000 ns
   ```

9. Right click in the waveform window and zoom in. Another way to zoom in, press and hold the middle mouse button and draw a square around the area to zoom in on. After simulating, you can then zoom in and view the delay from the clock edge to the TENSOUT, ONESOUT, and TENTHSOUT output change.

Note: The above commands have been combined into a macro file, time_sim.do, and can be executed at the ModelSim prompt.

The Synplicity / MTI / Xilinx Tutorial is now completed!