Chapter 1

Synopsys Design Compiler/FPGA Compiler/ ModelSim Tutorial for CPLDs

This tutorial shows you how to use Synopsys' Design Compiler / FPGA Compiler (VHDL / Verilog) for compiling XC9500 / XL / XV and Xilinx CoolRunner (XCR) CPLD designs, and Model Technology's ModelSim for simulation. It guides you through a typical CPLD 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"
- "Design Flow"
- "Creating the tenths LogiBLOX component for the XC9500"
- "RTL Simulation"
- "Synthesizing the Design Using Synopsys"
- "Implementing the Watch Design"
- "XC9500 / XL / XV Timing simulation"
- "XCR 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.

There is a companion Watch tutorial for Xilinx FPGAs, which have an on chip oscillator. Xilinx CPLDs do not have an on chip oscillator, and most of the differences in the tutorials are due to the use of an external system clock for the CPLD.

The Watch design inputs, outputs, and modules 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 the stopwatch has been stopped.
- **CLK** — Externally supplied system clock. A 36 KHz clock is used on XCR demo board.

Outputs

- **TENSOUT[6:0]**—7-bit bus which represents the tens-digit of the stopwatch value. This is viewable on the 7-segment LCD display of the XCR series 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. The output is not displayed.

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

- **DIVIDER**—A clock divider which divides the 36 KHz clock input to 17.5 Hz for internal use.
- **STWATCH**—A state machine that controls starting, stopping, and clearing the counters.
- **TENTHS**—A 10-bit counter which outputs the Tenths digit as 10-bit value. Optionally implemented using either the tenths.vhd (tenths.v) file or a LogiBLOX macro.

- **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.

Before Beginning the Tutorial

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

1. Install the following software.
   - Xilinx Development System 2.1i or WebPACK v2.1
   - Synopsys Design Compiler / FPGA Compiler v1998.08 or later
   - Model Technology ModelSim EE / PE 5.2 or later
   - Target XC9500 / XL / XV and / or XCR libraries (available from ftp: / / www.xilinx.com)
2. Verify that your system is properly configured. Consult the release notes and installation notes that came with your software package for more information.

Design Flow

The general flow is to do a functional simulation using ModelSim, and then use Synopsys Design Compiler / FPGA Compiler to compile the Verilog or VHDL files to an edif file. The <design>.edf file is input into a Xilinx tool, which produces a jedec file for programming the device and various results files, including timing simulation models. A ModelSim timing simulation is then run using the timing simulation model.

Designs targeting the XC9500 / XL / XV CPLDs, Xilinx Design Manager or WebPACK can be used for implementation. Designs targeting the XCR CPLDs can use WebPACK but not Design Manager. WebPACK does not support LogiBLOX, so if the design targets an XCR device, the tenths.vhd (tenths.v) module is used.

Functional simulation is the same for both XC9500 / XL / XV and XCR devices. Simulating the timing of the XC9500 / XL / XV is slightly different from that of the XCR CPLDs. The XV9500 / XL / XV uses the simprims library and generates a verilog or vhdl and sdf file. Designs targeting XCR devices use a delay-annotated verilog (.vo) or vhdl (.vho) file for timing simulation.

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. If LogiBLOX is used, the tenth files may be created in later steps. The following table lists the contents of the tutorial directories.

Directory	Description
cpld_tut/vhdl/watch	VHDL source, simulation, and script files
cpld_tut/verilog/watch	Verilog source, simulation, and script files

VHDL Design Files

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

- watch.vhd
- divider.vhd
- stmchine.vhd
- smallcntr.vhd
- cnt60.vhd
- hex2led.vhd
- tenths.vhd
- testbench.vhd (VHDL testbench for simulation)

Note: For designs targeting the XC9500/XL/XV CPLDs, the tenths counter can be created as a LogiBLOX macro, or the tenths.vhd source can be used. For designs targeting XCR CPLDs, the tenths.vhd source is used. LogiBLOX is not currently supported in the XCR flow.

Verilog Design Files

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

- watch.v
- divider.v
- stmchine.v
- smallcntr.v
- cnt60.v
- hex2led.v
- tenths.v
- testfixture.v (Verilog test fixture for simulation)

Note: For designs targeting the XC9500/XL/XV CPLDs, the tenths counter can be created as a LogiBLOX macro, or the tenths.v source can be used. For designs targeting XCR CPLDs, the tenths.v source is used. LogiBLOX is not currently supported in the XCR flow.

Script Files

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

- rtl_sim.do
- stim.do
- time_sim.do

Simulation Models for MTI

To simulate Xilinx designs with ModelSim, you can use the following simulation libraries which you must compile as described below. This isn’t necessary for designs targeting XCR devices, or for functional simulation or synthesis of designs targeting XC9500/XL/XV devices. The simprims library is used for timing simulation of the XC9500/XL/XV design.

- **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. The Verilog library has separate libraries by device family: UNI3000, UNISIMS (for 4000E/L/X, SPARTAN/XL, VIRTEX/E), UNI5200, UNI9000. This tutorial does not instantiate any Unisim primitives.
- **LogiBLOX Library**—The LogiBLOX library is used for designs containing LogiBLOX components, during pre-synthesis (RTL), and post-synthesis simulation. This is used in VITAL VHDL simulation only. Verilog uses the SIMPRIMS libraries.
- **SIMPRIMS Library**—The SIMPRIMS 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.

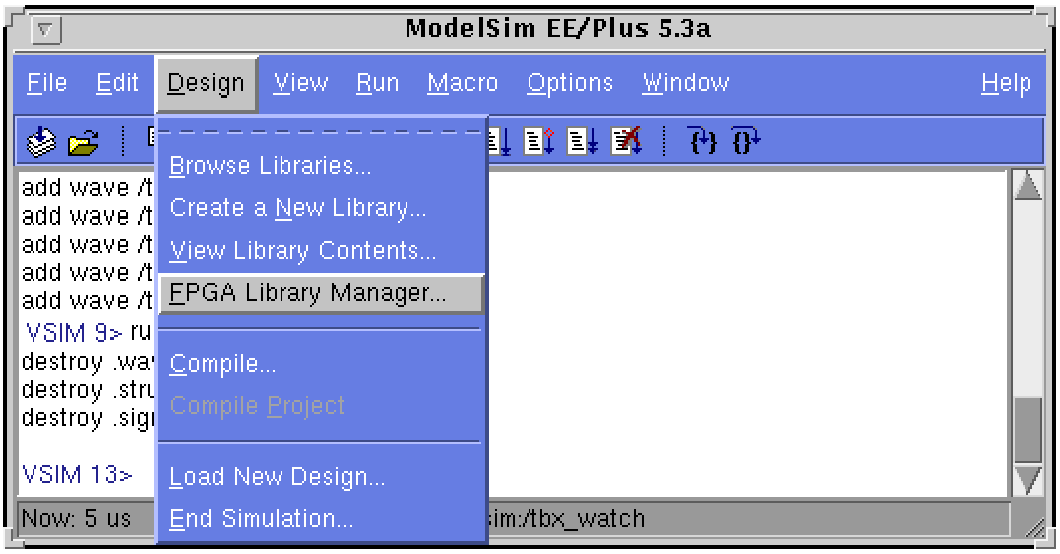
This tutorial uses the simprims library for XC9500/XL/XV designs. To compile the simprims library, invoke ModelSim by entering

vsim &

Design -> FPGA Library Manager -> Vendor File selection

Open **fpgavendor_xilinx.tcl** in the dialog box.

Compile the simprims library.



```
add wave /t
add wave /t
add wave /t
add wave /t
add wave /t
VSIM 9> ru
destroy .wa
destroy .stru
destroy .sig

VSIM 13>
Now: 5 us
sim:/tbx_watch
```

Figure 1-1 Compiling the simprims library

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. 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>/modelsim.ini
```

Copying the Tutorial Files

In this tutorial, “watch” is the name of the directory where the tutorial will be performed.

Creating the Tenths LogiBLOX Component

Designs targeting the XC9500/XL/XV may use LogiBLOX to generate the tenths macro. If used, it must be created 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. 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.
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 xc9500 family.
5. In the Options tab, set the following options.

   VHDL tutorial settings.

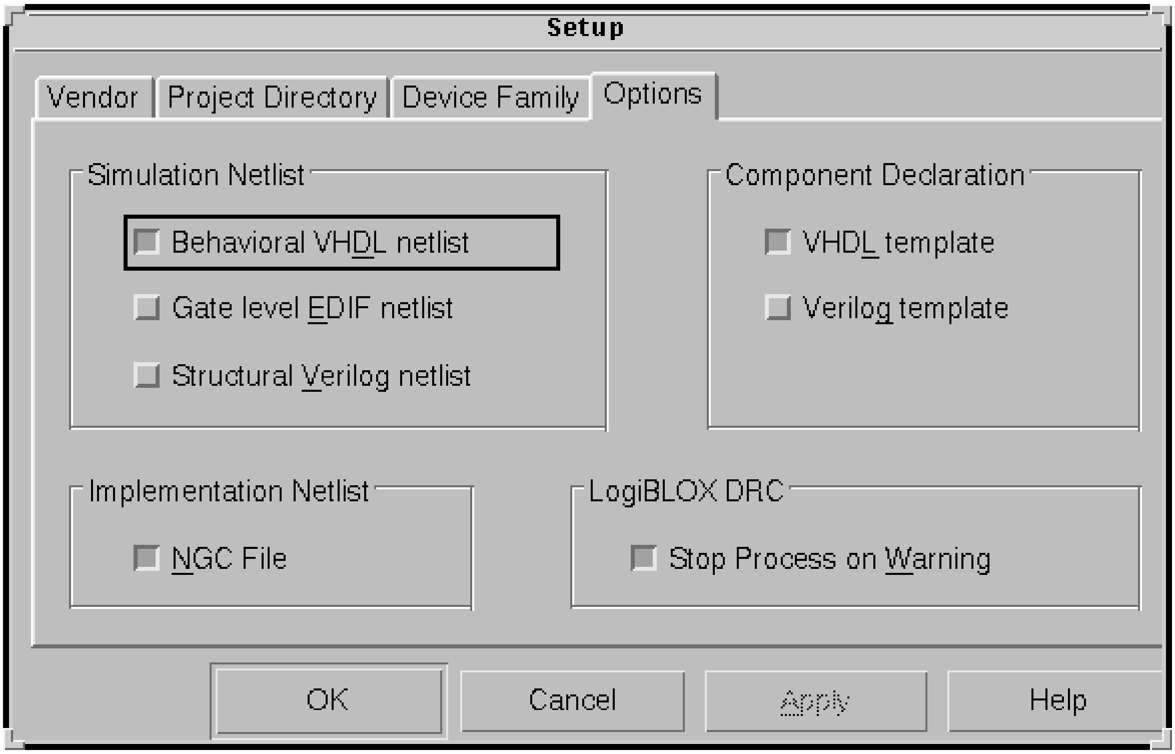
   - Simulation Netlist: Behavioral VHDL netlist
   - Component Declaration: VHDL Template
   - Implementation Netlist: NGC File

   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.



Simulation Netlist

- [✓] Behavioral VHDL netlist
- [ ] Gate level EDIF netlist
- [ ] Structural Verilog netlist

Component Declaration

- [✓] VHDL template
- [ ] Verilog template

Implementation Netlist

- [✓] NGC File

LogiBLOX DRC

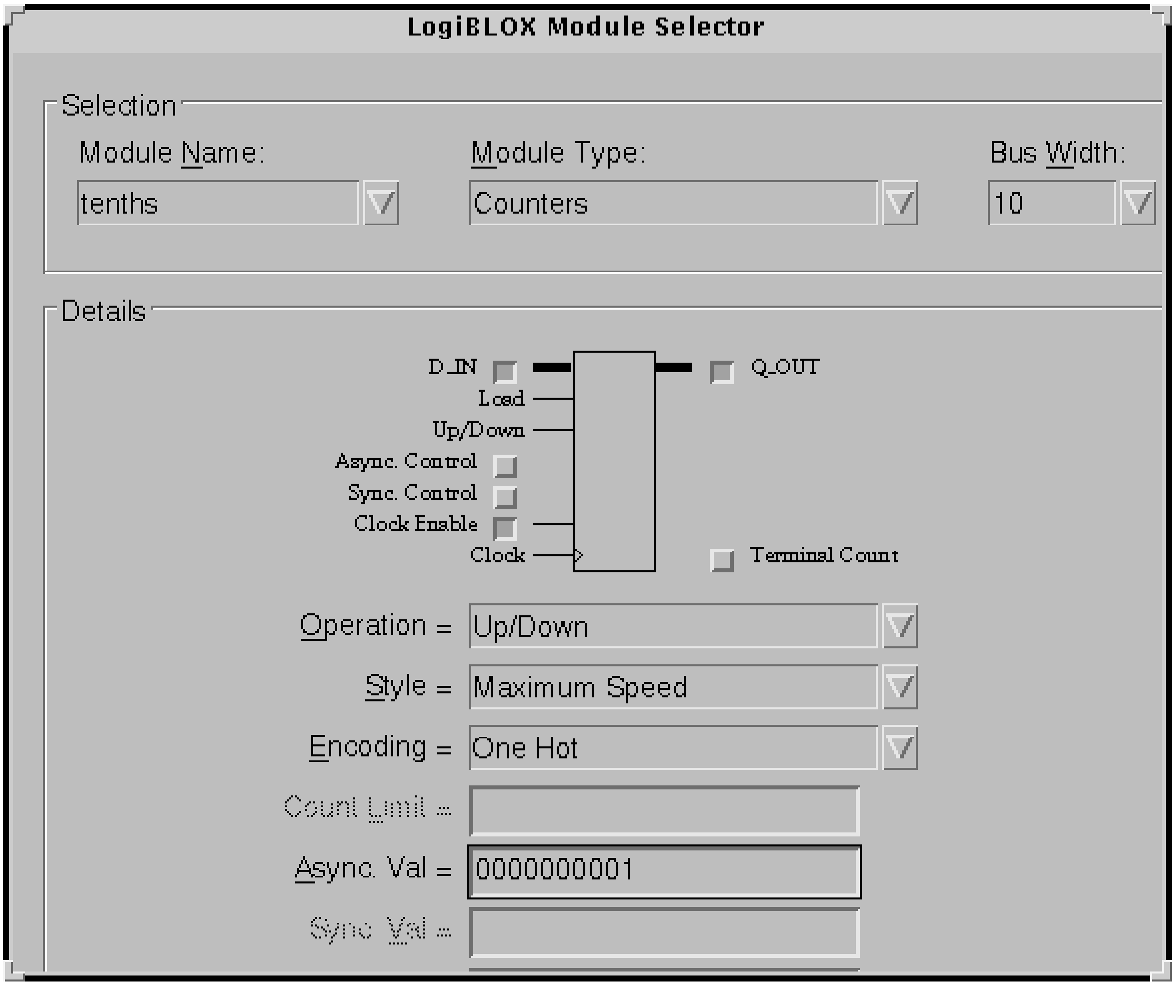
- [✓] Stop Process on Warning

Figure 1-2 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/Down

Style = Maximum Speed

Encoding = One Hot

Async. Val = 0000000001

Figure 1-3 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

Functional simulation is the same for the XC9500/XL/XV and XCR devices. In this tutorial, no simulation library is used for functional simulation.

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 CPLD. If this is not done, the flip-flops and latches in your simulation enter an unknown state. The general procedure for specifying 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. 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 **`/cpld_tut/vhdl/watch/func`** directory.

- divider.vhd
- smallcntr.vhd
- cnt60.vhd
- hex2led.vhd
- tenths.vhd

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

Verilog

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

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

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 &
```

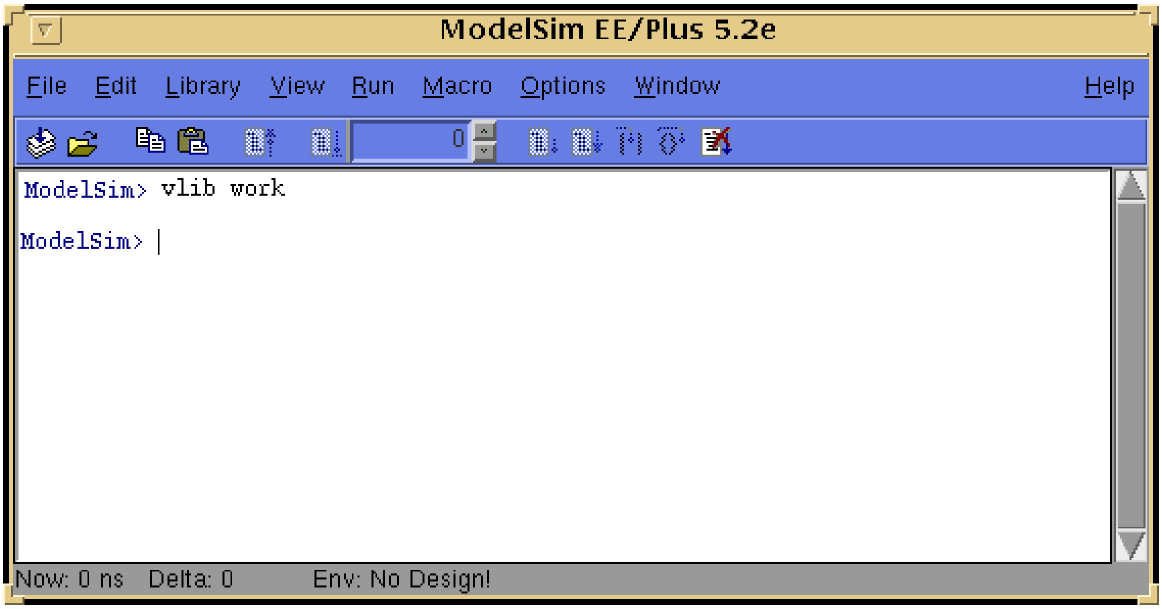
Set the project directory using the **`File`** → **`Change Directory`** menu command and select **`watch/func`**.

Creating the Work Directory

Before compiling the VHDL/Verilog source files, 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 1-4   MTI Transcript Window

Compiling the Source Files

VHDL

Design Compiler / FPGA Compiler supports translate_off / translate_on directives. Translate_off instructs Design Compiler / FPGA Compiler 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.

The Vcom command compiles VHDL code for use with Vsim RTL simulation. Also, to enhance simulation, 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 divider.vhd cnt60.vhd tenths.vhd
```

vcom -93 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

If LogiBLOX is used, 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 earlier, 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 divider.v stmchine.v
hex2led.v cnt60.v smallcntr.v tenths.v
```

Invoke the Simulator

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

vsim tbx_watch tbx_arch

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

vsim watch_tf.v watch.v

For LogiBLOX generated components, Ngd2ver is used to generate a structural Verilog netlist to facilitate functional simulation. The structural netlist contains SIMPRIMS library components which are mapped to the simprims library.

Note: The file, rtl_sim.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, execute 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.

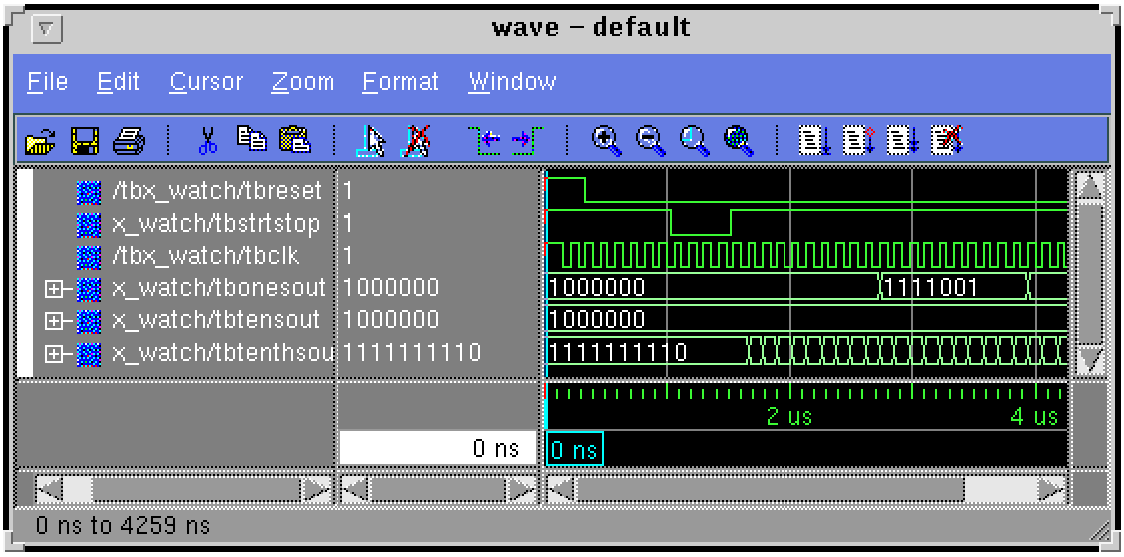


Figure 1-5 Simulation Output in Wave Window

Synthesizing the Design Using Synopsys

In this section, you synthesize the design using a script file. Synthesis is the similar whether targeting a XC9500/XL/XV or XCR device, with the target, link, and symbol library in the .synopsys_dc.setup file the only difference. An example .synopsys_dc.setup file for targeting the XC9500/XL/XV device is given below.

```
designer = “Xilinx CPLD Applications”
company = “Xilinx, Inc”
search_path = {., ~/synlib, /cadappl/packages/synopsys/1998.08/
libraries/syn, ~} ;
link_library = {xc9000.db} ;
target_library = {xc9000.db};
symbol_library = {xc9000.sdb} ;
synthetic_library = {standard.sldb}
```

```
bussing_no_ladder = “true”;
edifout_netlist_only = “true”;
edifout_power_and_ground_representation = “cell”;
edifout_power_name = “VCC”;
edifout_ground_name = “GND”;
edifout_power_pin_name = P ;
edifout_ground_pin_name = G ;
compile_fix_multiple_port_nets = “true” ;
bus_naming_style = “%s<%d>”
bus_inference_style = “%s<%d>”
bus_dimension_separator_style  = “><“
edif_write_properties_list = “instance_number pad_location part”
compile_fix_multiple_port_nets = true
hdlin_translate_off_skip_text = true
edifout_write_properties_list = {INIT IO LOC PWR_MODE
PAD_LOCATION PART }
```

If you are targeting a XCR device, change the link_library and target library to xcr.db. Remove the symbol library and edifout_write_properties_list.

For the VHDL tutorial, create watch.script as shown below.

```
read -format vhdl cnt60.vhd
read -format vhdl divider.vhd
read -format vhdl hex2led.vhd
read -format vhdl smallcntr.vhd
read -format vhdl stmchine.vhd
read -format vhdl tenths.vhd
read -format vhdl watch.vhd
check_design
ungroup -all -flatten
uniquify
compile -map_effort low
ungroup -all -flatten
current_design
write -format edif -output watch.edf
exit
```

For the Verilog tutorial, create watch.script as shown above for VHDL designs, substituting Verilog as the format and .v as the file extension.

At the Unix prompt run

```
dc_shell -f watch.script | tee log
```

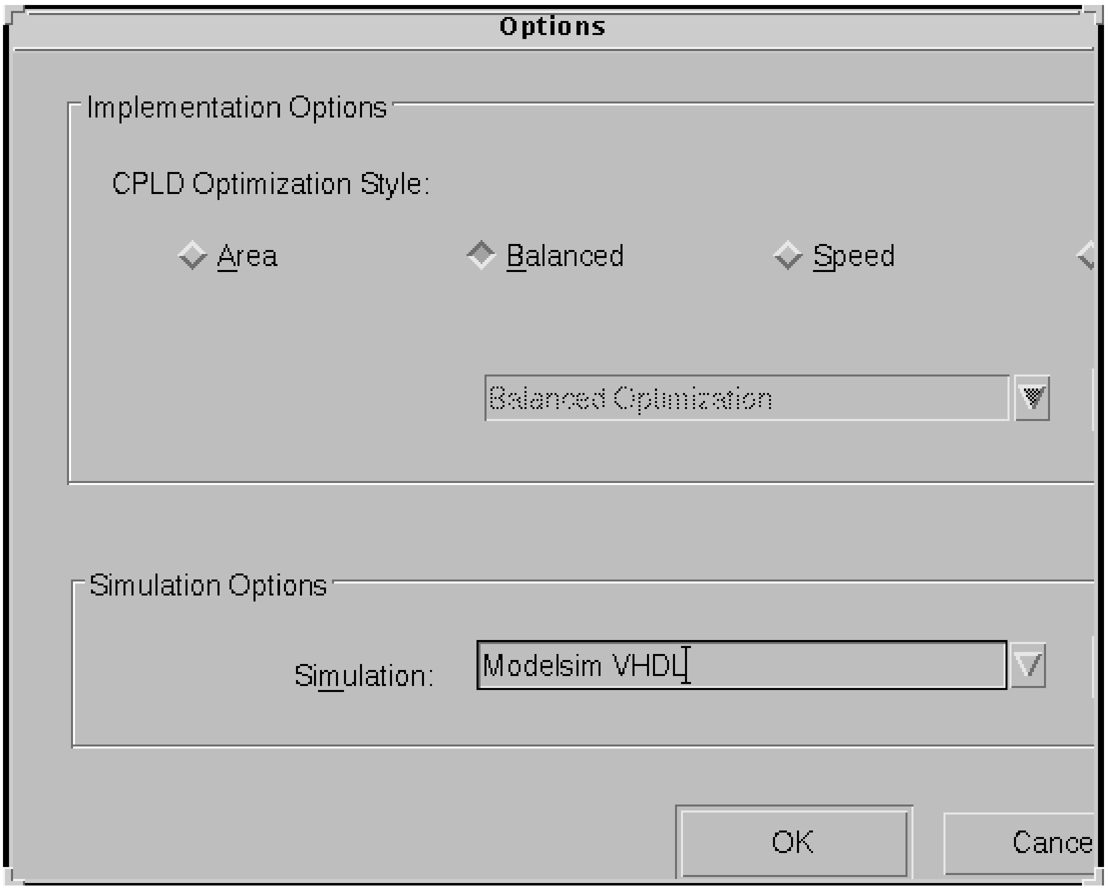
Implementing the Watch Design

The XC9500/XL/XV can be implemented in either Xilinx Design Manager or WebPACK, while the XCR can only be implemented in WebPACK. To implement the Watch design, refer to the *Xilinx Design Manager Tutorial* or to WebPACK documentation. You need the following files for implementation.

- watch.edf
- tenths.ngc (if LogiBLOX is used)

When you implement the Watch design with the Xilinx Design Manager, 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.



CPLD Optimization Style: Balanced

Balanced Optimization

Simulation: Modelsim VHDL

Figure 1-6 Design Manager Implement Dialog Box

3. Proceed with the Design Manager or WebPACK 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.

XC9500/XL/XV Timing Simulation

VHDL

For VHDL simulation, you need two files.

- time_sim.vhd

- time_sim.sdf

To perform timing simulation, follow these steps.

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

   `/cpld_tut/vhdl/watch/time`

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

   `/cpld_tut/vhdl/watch/time`

3. Create the work directory.

   `vlib work`

4. Compile the VHDL source files and the testbench.

   vcom time_sim.vhd testbench.vhd

5. Read in the SDF file for timing simulation.

   `vsim -sdftyp uut=time_sim.sdf tbx_watch tbx_arch`

   Alternatively, select **`File`** → **`Load New Design`**. Highlight the design 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 the time_sim.sdf file. Type **`uut`** in the Apply to Region field and click the Load button.

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

   view wave signals source

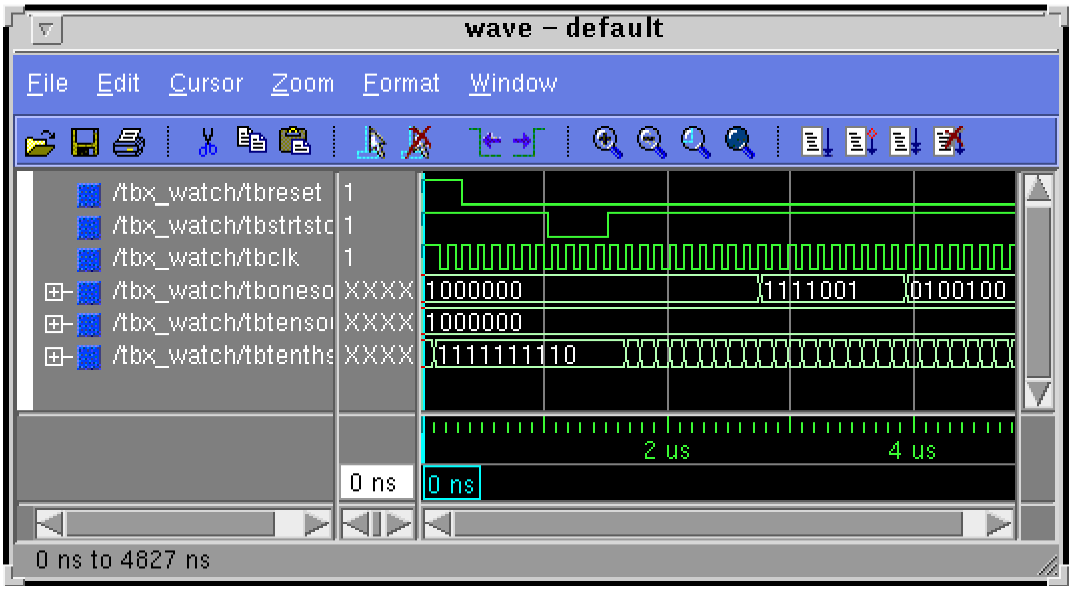
7. View and add the signals of the design to the waveform window.

8. At the ModelSim prompt type.

   `run 100000 ns`

9. Right click in the waveform window and zoom in. Another way to zoom in is to 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 simulation you need two files.

- 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.

   /cpld_tut/verilog/watch/time

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

   /cpld_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 test
   ```

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

6. View the necessary debugging windows by typing the following command at the ModelSim prompt. Use the ModelSim **Combine** command to group the tensout and onesout signals into buses.

   ```
   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 is to 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.

XCR Timing Simulation

VHDL

For timing simulation of a VHDL design using a XCR CPLD, you need two files.

Note: The testbencht.vhd file is an edited version of the original testbench.vhd file. In the VHDL timing model (watch.vho), the tensout, onesout, and tenthsout bus signals are broken into discrete signals. For simulation, the component signals and uut signals in the testbench and design model must match. The component and uut instantiation statements in testbencht.vhd have been edited to match those in watch.vho.

- watch.vho
- testbencht.vhd

To perform timing simulation, follow these steps.

1. Copy watch.vho and testbencht.vhd to the following directory.

   `/cpld_tut/vhdl/watch/time`

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

   `/cpld_tut/vhdl/watch/time`

3. Create the work directory.

   `vlib work`

4. Compile the VHDL source files and the testbench.

   vcom watch.vho testbencht.vhd

5. Read in the files for timing simulation.

   `vsim tbx_watch tbx_arch`

   Alternatively, select **`File`** → **`Load New Design`**. Click the Add button. Browse and select the design file. Type **`uut`** in the Apply to Region field and click the Load button.

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

   view wave signals source

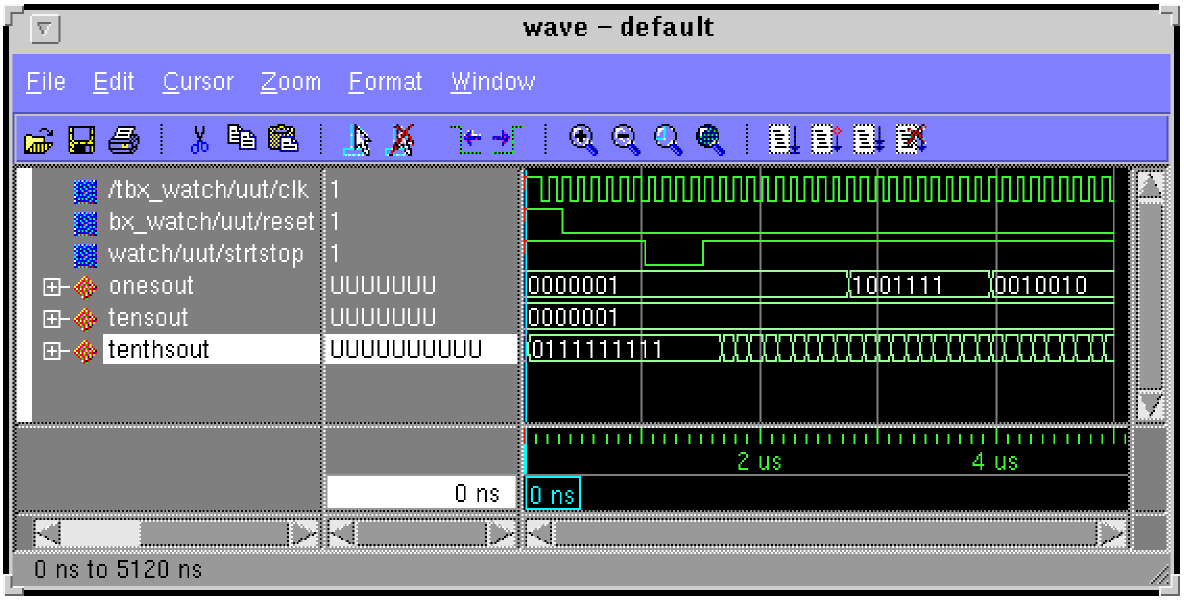
7. View and add the signals of the design to the waveform window. Use the ModelSim **Combine** command to group the tensout and onesout signals into buses.

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.



Verilog

For timing simulation of the Verilog design you need two files.

- watch.vo
- testfixturet.v

Note: The testfixturet.v file is an edited version of the original tesfixture.v file. In the Verilog timing model (watch.vo), the tensout, onesout, and tenthsout bus signals are broken into discrete signals. For simulation, the component signals and uut signals in the test-bench and design model must match.

Note:

To perform timing simulation, follow these steps.

1. Copy watch.vo and testfixturet.v to the following directory.

   `/cpld_tut/verilog/watch/time`

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

```
/cpld_tut/verilog/watch/time
```

3. Create the work directory.

   ```
   vlib work
   ```

4. Compile the Verilog file and the testfixture.

   ```
   vlog testfixturet.v watch.vo
   ```

5. Simulate the design

   ```
   vsim -L simprims_ver test
   ```

   Now that the HDL netlist has been resolved into primitives, we must provide the simulation models to the SIMPRIMS 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 Synopsys/MTI/Xilinx CPLD Tutorial is now completed!