ModelSim®

SE

Tutorial

Version 5.6d

Published: 6/Aug/02

The world's most popular HDL simulator

Model Technology Incorporated
10450 SW Nimbus Avenue / Bldg. R-B
Portland OR 97223-4347 USA

phone: 503-641-1340
fax: 503-526-5410
e-mail: support@model.com, sales@model.com
home page: http://www.model.com
support page: http://www.model.com/support

Table of Contents

Introduction

Chapter contents

Software versions

This documentation was written to support ModelSim SE 5.6d for UNIX and Microsoft Windows 98/Me/NT/2000/XP. If the ModelSim software you are using is a later release, check the README file that accompanied the software. Any supplemental information will be there.

Although this document covers both VHDL and Verilog simulation, you will find it a useful reference even if your design work is limited to a single HDL.

ModelSim's graphic interface

While your operating system interface provides the window-management frame, ModelSim controls all internal-window features including menus, buttons, and scroll bars. The resulting simulator interface remains consistent within these operating systems:

- SPARCstation with OpenWindows, OSF/Motif, or CDE
- IBM RISC System/6000 with OSF/Motif
- Hewlett-Packard HP 9000 Series 700 with HP VUE, OSF/Motif, or CDE
- Linux (Red Hat v. 6, 7 or later) with KDE or GNOME
- Microsoft Windows 98/Me/NT/2000/XP

Because ModelSim's graphic interface is based on Tcl/Tk, you also have the tools to build your own simulation environment. Easily accessible preference variables and configuration commands, simulator preference variables, and graphic interface commands give you control over the use and placement of windows, menus, menu options and buttons.

Standards supported

ModelSim VHDL supports both the IEEE 1076-1987 and 1076-1993 VHDL, the 1164-1993 *Standard Multivalue Logic System for VHDL Interoperability*, and the 1076.2-1996 *Standard VHDL Mathematical Packages* standards. Any design developed with ModelSim will be compatible with any other VHDL system that is compliant with either IEEE Standard 1076-1987 or 1076-1993.

ModelSim Verilog is based on IEEE Std 1364-1995 and a partial implementation of 1364-2001 (see */<install_dir>/modeltech/docs/technotes/vlog_2000.note* for implementation details) *Standard Hardware Description Language*. The Open Verilog International *Verilog LRM version 2.0* is also applicable to a large extent. Both PLI (Programming Language Interface) and VCD (Value Change Dump) are supported for ModelSim PE and SE users.

In addition, all products support SDF 1.0 through 3.0, VITAL 2.2b, VITAL'95 – IEEE 1076.4-1995, and VITAL 2000 – IEEE 1076.4-2000.

Assumptions

We assume that you are familiar with the use of your operating system. You should be familiar with the window management functions of your graphic interface: either OpenWindows, OSF/Motif, CDE, KDE, GNOME, or Microsoft Windows 98/Me/NT/2000/XP.

We also assume that you have a working knowledge of VHDL and Verilog. Although ModelSim is an excellent tool to use while learning HDL concepts and practices, this document is not written to support that goal.

Where to find our documentation

ModelSim documentation is available from our website at www.model.com/support/documentation.asp or in the following formats and locations:

Document	Format	How to get it
Start Here for ModelSim SE (installation & support reference)	paper	shipped with ModelSim
	PDF	select **Main window > Help > SE Documentation**; also available from the Support page of our web site: www.model.com
ModelSim SE Quick Guide (command and feature quick-reference)	paper	shipped with ModelSim
	PDF	select **Main window > Help > SE Documentation**, also available from the Support page of our web site: www.model.com
ModelSim SE Tutorial	PDF, HTML	select **Main window > Help > SE Documentation**; also available from the Support page of our web site: www.model.com
ModelSim SE User's Manual	PDF, HTML	select **Main window > Help > SE Documentation**
ModelSim SE Command Reference	PDF, HTML	select **Main window > Help > SE Documentation**
ModelSim Foreign Language Interface Reference	PDF, HTML	select **Main window > Help > SE Documentation**
Std_DevelopersKit User's Manual	PDF	www.model.com/support/pdf/sdk_um.pdf The Standard Developer's Kit is for use with Mentor Graphics QuickHDL.
ModelSim Command Help	ASCII	type `help [command name]` at the prompt in the Main window
Error message help	ASCII	type `verror <msgNum>` at the prompt in the Main window or at a shell prompt
Tcl Man Pages (Tcl manual)	HTML	select **Main window > Help > Tcl Man Pages**, or find *contents.htm* in *\modeltech\docs\tcl_help_html*

Document	Format	How to get it
application notes	HTML	www.model.com/resources/techdocs.asp
frequently asked questions	HTML	www.model.com/resources/faqs.asp
tech notes	ASCII	select **Main window > Help > Technotes**, or located in the *\modeltech\docs\technotes* directory

Technical support and updates

The Model Technology web site includes links to support, software updates, and many other information sources.

Support

www.model.com/support/default.asp

Customers in Europe should contact their distributor for support. See www.model.com/contact_us.asp for distributor contact information.

Updates

www.model.com/products/release.asp

Latest version email

Place your name on our list for email notification of news and updates.
www.model.com/support/register_news_list.asp

Before you begin

Preparation for some of the lessons leaves certain details up to you. You will decide the best way to create directories, copy files and execute programs within your operating system. (When you are operating the simulator within ModelSim's GUI, the interface is consistent for all platforms.)

Additional details for VHDL, Verilog, and mixed VHDL/Verilog simulation can be found in the *ModelSim User's Manual* and *Command Reference*. (See "Where to find our documentation" (T-7).)

Examples show Windows path separators - use separators appropriate for your operating system when trying the examples.

Command, button, and menu equivalents

Many of the lesson steps are accomplished by a button or menu selection. When appropriate, VSIM command line (PROMPT:) or menu (MENU:) equivalents for these selections are shown in parentheses within the step. This example shows three options to the **run -all** command, a button, prompt command, and a menu selection.

(PROMPT: run -all) (MENU: Simulate > Run > Run -All) 

Drag and drop

Drag and drop allows you to copy and move signals among windows. If drag and drop applies to a lesson step, it is noted in a fashion similar to MENUS and PROMPTS with: DRAG&DROP.

Command history

As you work on the lessons, keep an eye on the Main transcript window. The commands invoked by buttons and menu selections are echoed there. You can scroll through the command history with the up and down arrow keys, or the command history may be reviewed with several shortcuts at the ModelSim/VSIM prompt.

Shortcut	Description
click on prompt	left-click once on a previous ModelSim or VSIM prompt in the transcript to copy the command typed at that prompt to the active cursor
`his or history`	shows the last few commands (up to 50 are kept)

Reusing commands from the Main transcript

ModelSim's Main transcript can be saved, and the resulting file used as a DO (macro) file to replay the transcribed commands. You can save the transcript at any time before or during simulation. You have the option of clearing the transcript (File > Transcript > Clear Transcript) if you don't want to save the entire command history.

To save the contents of the transcript select **File > Transcript > Save Transcript As** from the Main menu.

Replay the saved transcript with the **do** command:

```
do <do file name>
```

For example, if you saved a series of compiler commands as *mycompile.do* (the .do extension is optional), you could recompile with one command:

```
do mycompile.do
```

Note: Neither the prompt nor the Return that ends a command line are shown in the examples.

Lesson 1 - Creating a Project

The goals for this lesson are:

- Create a project

A project is a collection entity for an HDL design under specification or test. Projects ease interaction with the tool and are useful for organizing files and simulation settings. At a minimum, projects have a work library and a session state that is stored in a .mpf file. A project may also consist of:

- HDL source files or references to source files
- other files such as READMEs or other project documentation
- local libraries
- references to global libraries

For more information about using project files, see the *ModelSim User's Manual*.

Creating a project

1 Start ModelSim with one of the following:

for UNIX at the shell prompt:

```
vsim
```

for Windows - your option - from a Windows shortcut icon, from the Start menu, or from a DOS prompt:

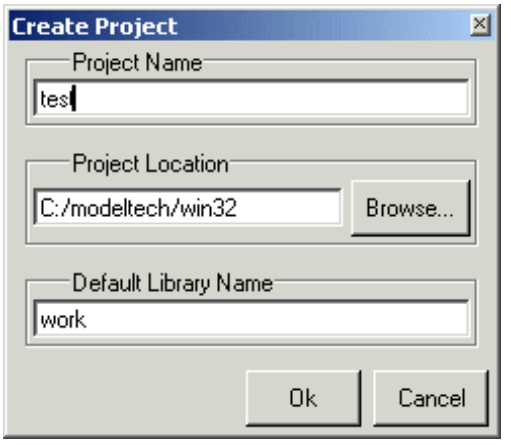
```
modelsim.exe
```

Note: if you didn't add ModelSim to your search path during installation, you will have to include the full path when you type this command at a DOS prompt.

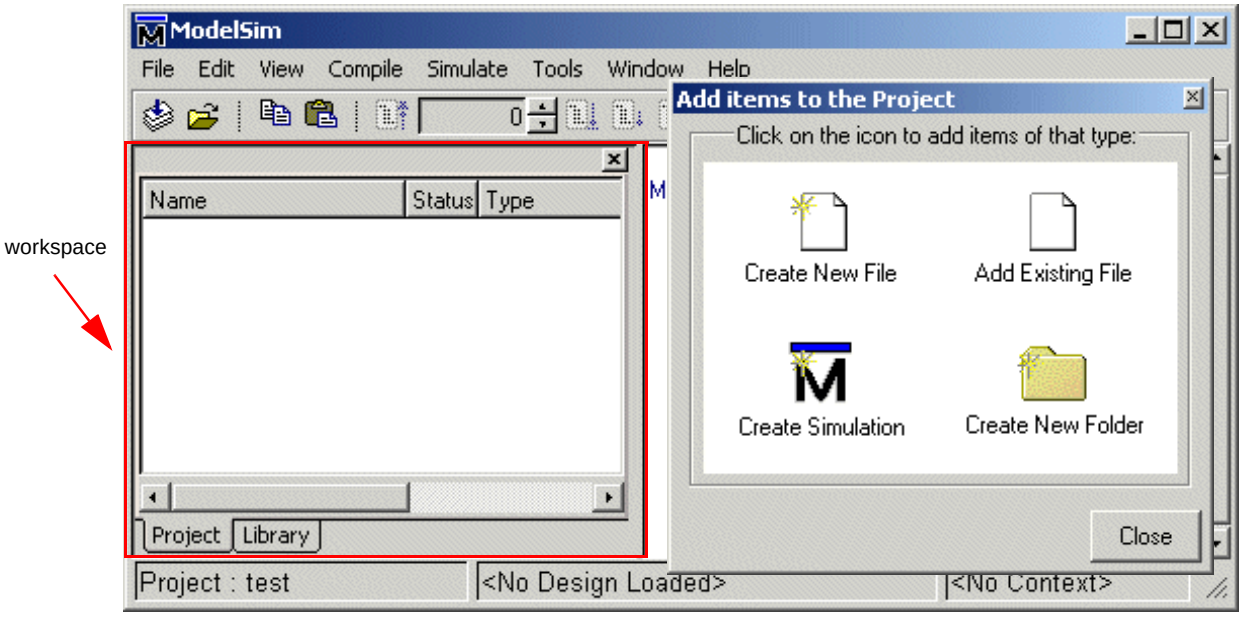
Upon opening ModelSim for the first time, you will see the **Welcome to ModelSim** dialog. (If this screen is not available, you can display it by selecting **Help** > **Welcome Menu** from the Main window.)

2 Select **Create a Project** from the Welcome dialog, or **File** > **New** > **Project** (Main window). In the **Create Project** dialog box, enter "test" as the Project Name and select

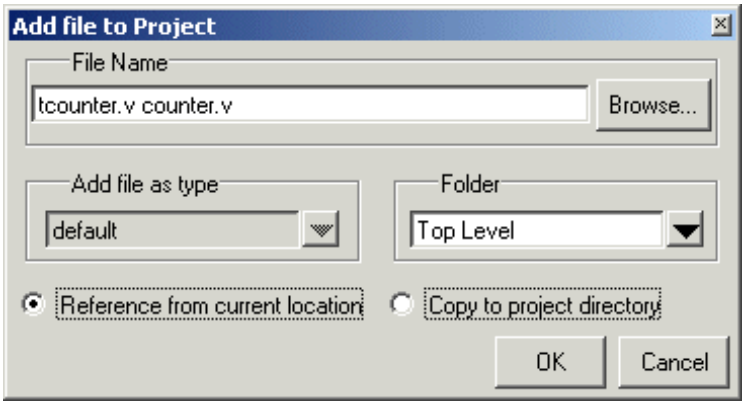
a directory where the project file will be stored. Leave the Default Library Name set to "work."



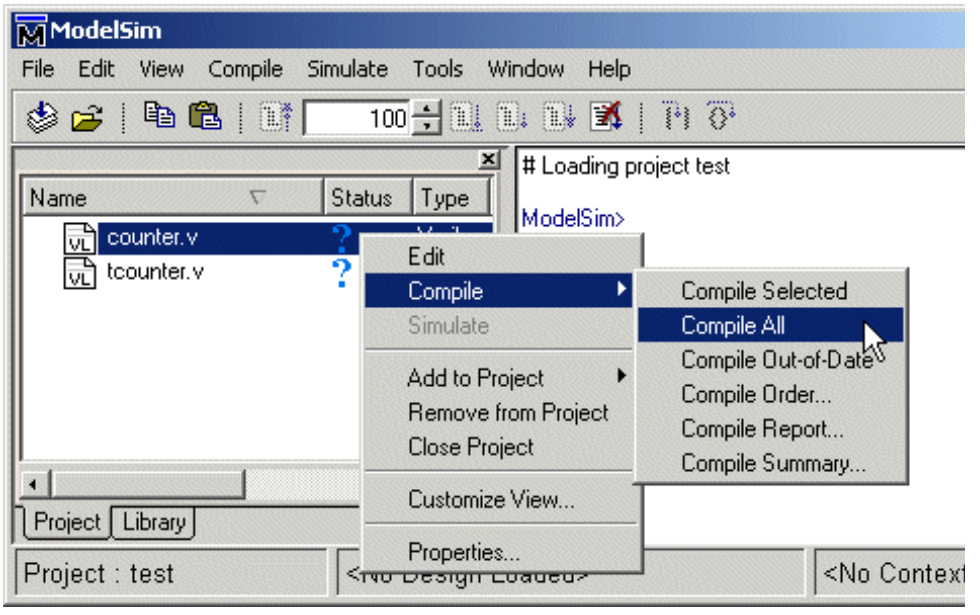
Upon selecting OK, you will see a blank Project tab in the workspace area of the Main window and the **Add Items to the Project** dialog.



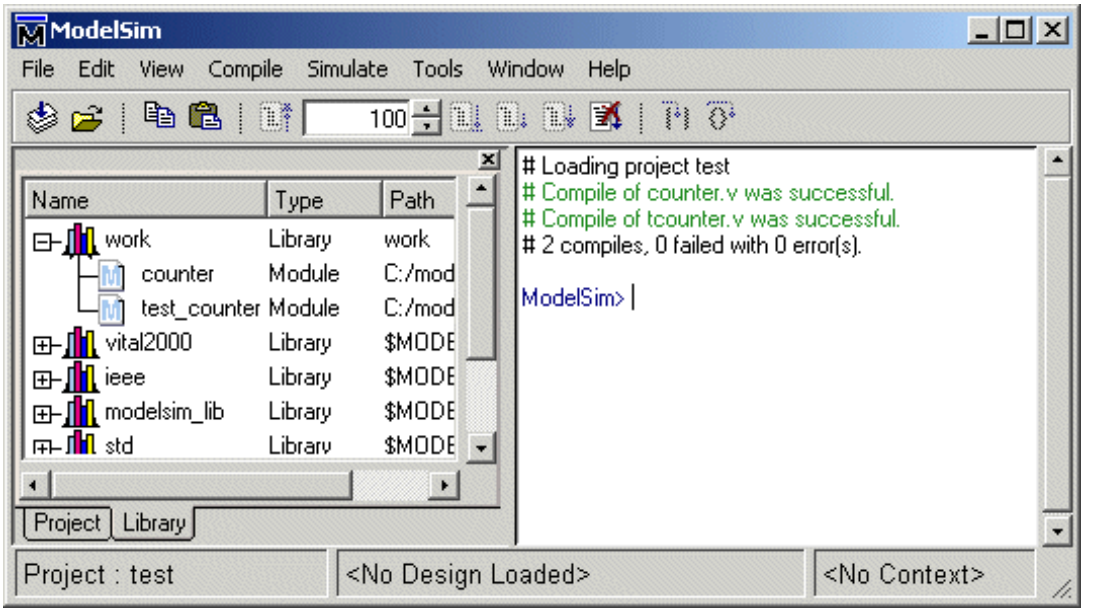
3 The next step is to add the files that contain your design units. Click **Add Existing File** in the **Add Items to Project** dialog. For this exercise, we'll add two Verilog files. Click the **Browse** button in the Add file to Project dialog box and open the examples directory in your ModelSim installation. Select *tcounter.v* and *counter.v*. Select **Reference from current location** and then click OK.



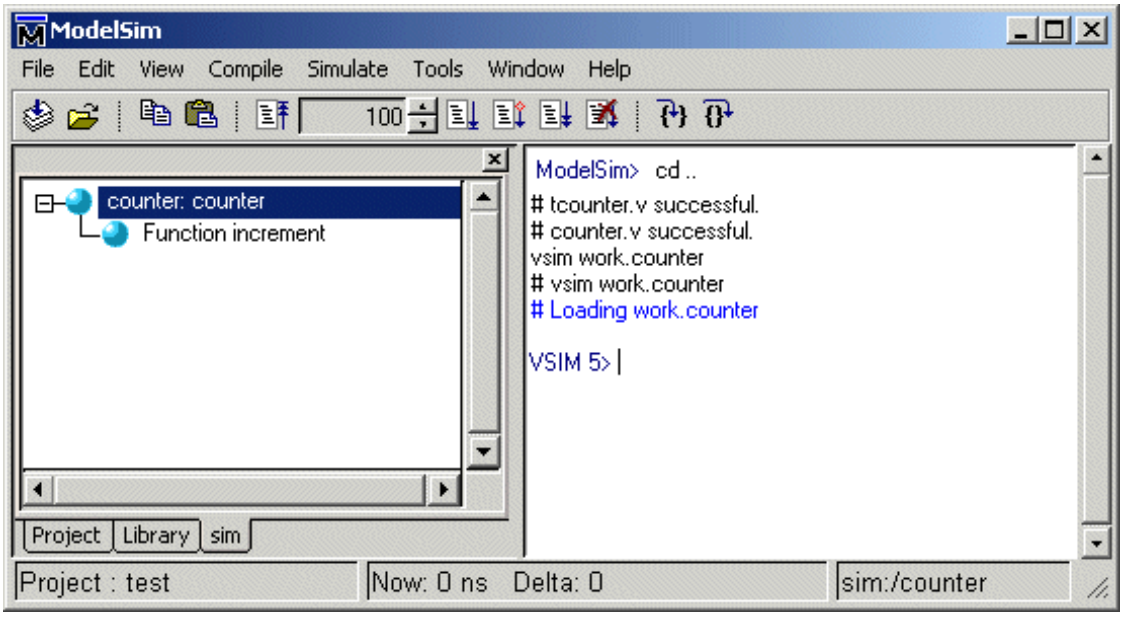
4 Click your right mouse button (2nd button in Windows; 3rd button in UNIX) in the Project page and select **Compile > Compile All**.



5 The two files are compiled. Click on the Library tab and expand the *work* library by clicking the "+" icon. You'll see the compiled design units listed.



6 The last step in this exercise is to load one of the design units. Double-click *counter* on the Library page. You'll see a new page appear in the Workspace that displays the structure of the *counter* design unit.



At this point, you would generally run the simulation and analyze or debug your design. We'll do just that in the upcoming lessons. For now, let's wrap up by ending the simulation and closing the project. Select **Simulate > End Simulation** and confirm that you want to

quit simulating. Next, select **File > Close > Project**, confirm that you want to close the project, and select **Yes** to update your project file with the changes you made during this session.

Note that a *test.mpf* file has been created in your working directory. This file contains information about the project *test* that you just created. ModelSim will open this project automatically the next time you invoke the tool.

Lesson 2 - Basic VHDL simulation

The goals for this lesson are:

- Create a library and compile a VHDL file
- Load a design
- Learn about the basic ModelSim windows, mouse, and menu conventions
- Force the value of a signal
- Run ModelSim using the **run** command
- Set a breakpoint
- Single-step through a simulation run

The project feature covered in Lesson 1 executes several actions automatically such as creating and mapping work libraries. In this lesson we will go through the entire process so you get a feel for how ModelSim really works.

Compiling the design

1 Start by creating a new directory for this exercise (in case other users will be working with these lessons). Create the directory, then copy all of the VHDL (.vhd) files from *\<install_dir\>\modeltech\examples* to the new directory.

Make sure the new directory is the current directory. Do this by invoking ModelSim from the new directory or by selecting **File > Change Directory** (Main window).

2 Start ModelSim with one of the following:

for UNIX at the shell prompt:

```
vsim
```

for Windows - your option - from a Windows shortcut icon, from the Start menu, or from a DOS prompt:

```
modelsim.exe
```

Note: If you didn't add ModelSim to your search path during installation, you will have to include the full path when you type this command at a DOS prompt.

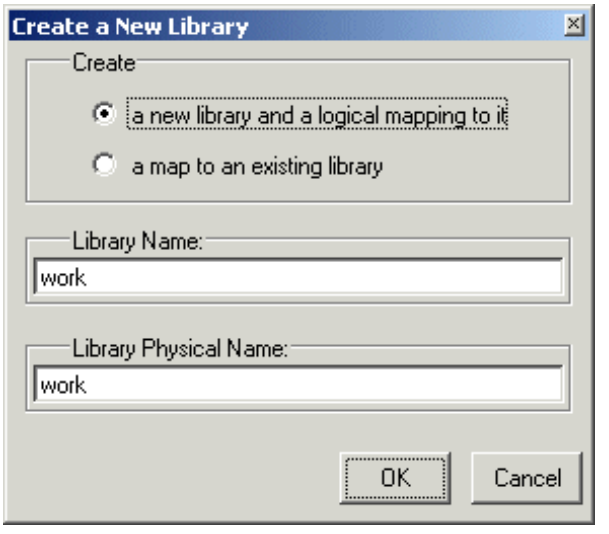
Select "Proceed to ModelSim" if the Welcome dialog appears.

3 Before you compile any HDL code, you'll need a design library to hold the compilation results. To create a new design library, make this menu selection in the Main window: **File > New > Library**.

Make sure **Create: a new library and a logical mapping to it** is selected. Type "work" in the Library Name field and then select **OK**.

This creates a subdirectory named *work* - your design library - within the current directory. ModelSim saves a special file named *_info* in the subdirectory.

(PROMPT: vlib work
vmap work work)

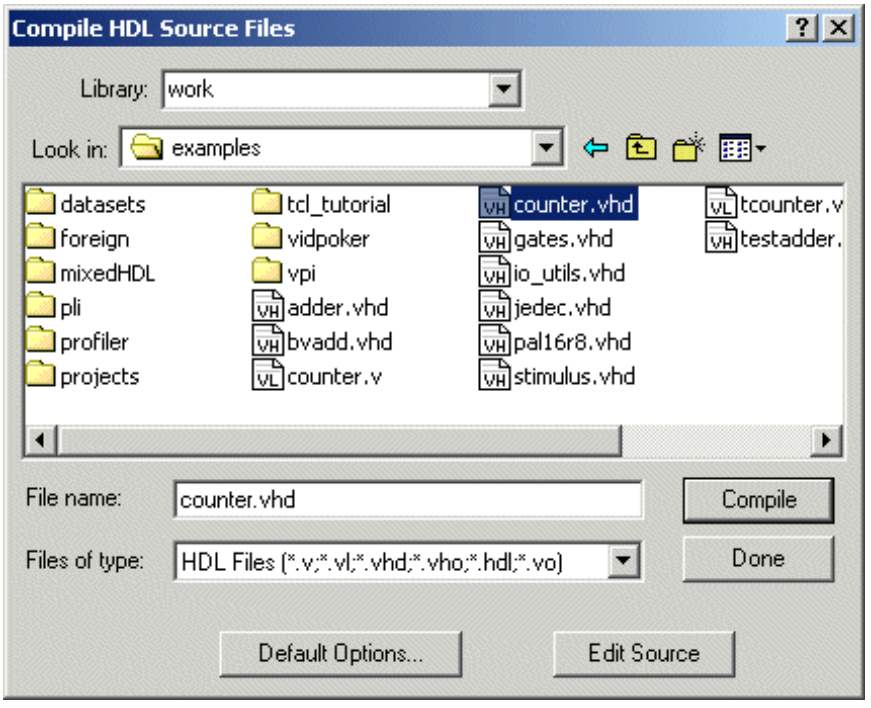


Note: Do not create a Library directory using UNIX or Windows commands, because the *_info* file will not be created. Always use the File menu or the **vlib** command from either the ModelSim or UNIX/DOS prompt.)

4 Compile the file *counter.vhd* into the new library by selecting **Compile > Compile**.

(PROMPT: vcom counter.vhd) 

This opens the Compile HDL Source Files dialog box. (You won't see this dialog box if you invoke vcom from the command line.)



Complete the compilation by selecting *counter.vhd* from the file list and clicking **Compile**. Select **Done** when you are finished.

You can compile multiple files in one session from the file list. Individually select and compile the files in the order required by your design.

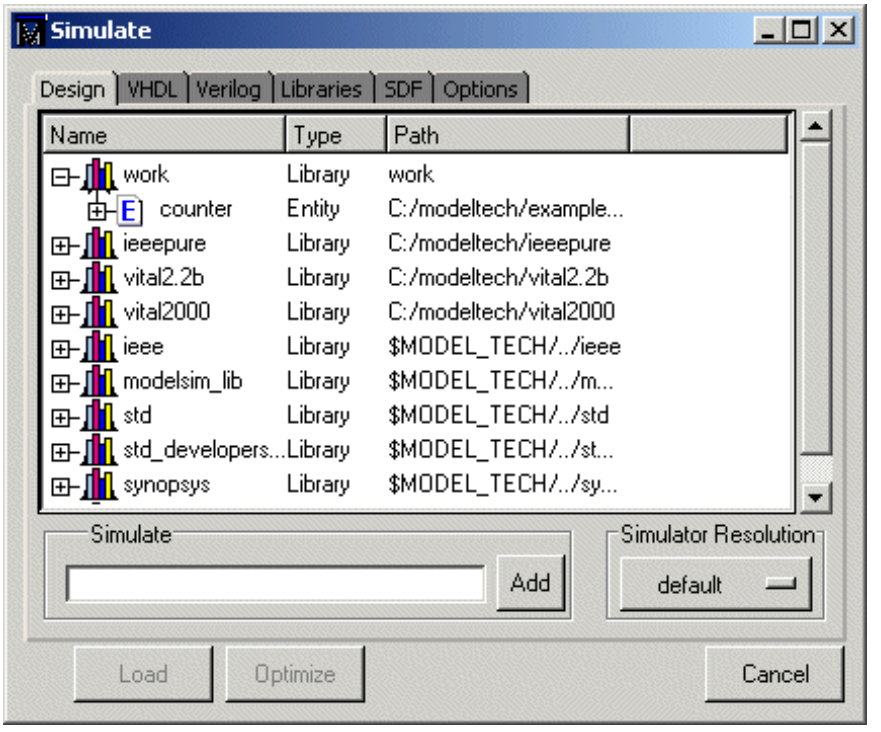
Note that you can have ModelSim determine the compile order. See "Auto-generating compile order" in the Project chapter of the *ModelSim User's Manual* for details.

Loading the design

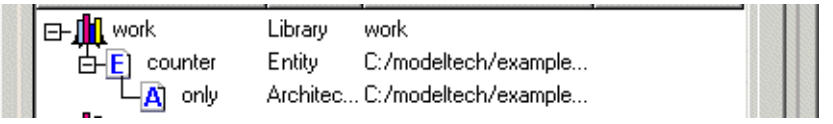
1 Load the design unit by selecting **Simulate > Simulate**.

(PROMPT: vsim counter) 

The Simulate dialog box appears. Click the "+" sign next to 'work' to see the **counter** design unit. (You won't see this dialog box if you invoke **vsim** with *counter* from the command line.)



If the Design Unit is an entity (like **counter** in this design), you can expand it to view any associated architectures.



Select the entity **counter** and choose **Load**.

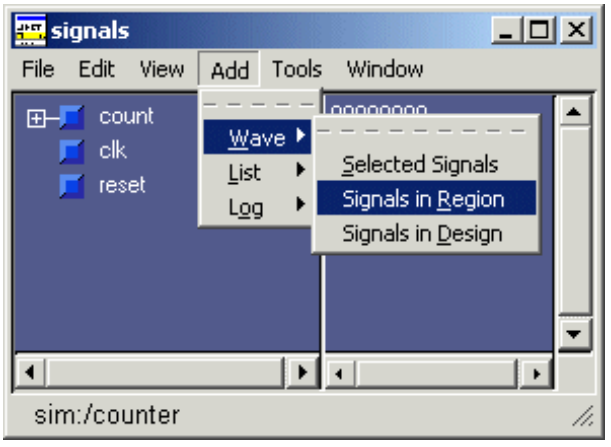
2 Next, select **View > All Windows** from the Main window menu to open all ModelSim windows.

(PROMPT: view *)

For descriptions of the windows, consult the *ModelSim User's Manual*.

3 Next let's add top-level signals to the Wave window by selecting **Add** > **Wave** > **Signals in Region** from the Signals window menu.

(PROMPT: add wave /counter/*)



Running the simulation

We will start the simulation by applying stimulus to the clock input.

1. Click in the Main window and enter the following command at the VSIM prompt:

   ```
   force clk 1 50, 0 100 -repeat 100
   ```

   (Signals MENU: Edit > Clock)

   ModelSim interprets this **force** command as follows:

   - force clk to the value 1 at 50 ns after the current time
   - then to 0 at 100 ns after the current time
   - repeat this cycle every 100 ns

2. Now you will exercise two different **Run** functions from the toolbar buttons on either the Main or Wave window. (The **Run** functions are identical in the Main and Wave windows.) Select the **Run** button first. When the run is complete, select **Run -All**.

   Run. This causes the simulation to run and then stop after 100 ns.

   (PROMPT: run 100) (Main MENU: Simulate > Run > Run 100ns) 

   Run -All. This causes the simulator to run forever. To stop the run, go on to the next step.

   (PROMPT: run -all) (Main MENU: Simulate > Run > Run -All) 

3. Select the **Break** button on either the Main or Wave toolbar to interrupt the run. The simulator will stop running as soon as it gets to an acceptable stopping point.

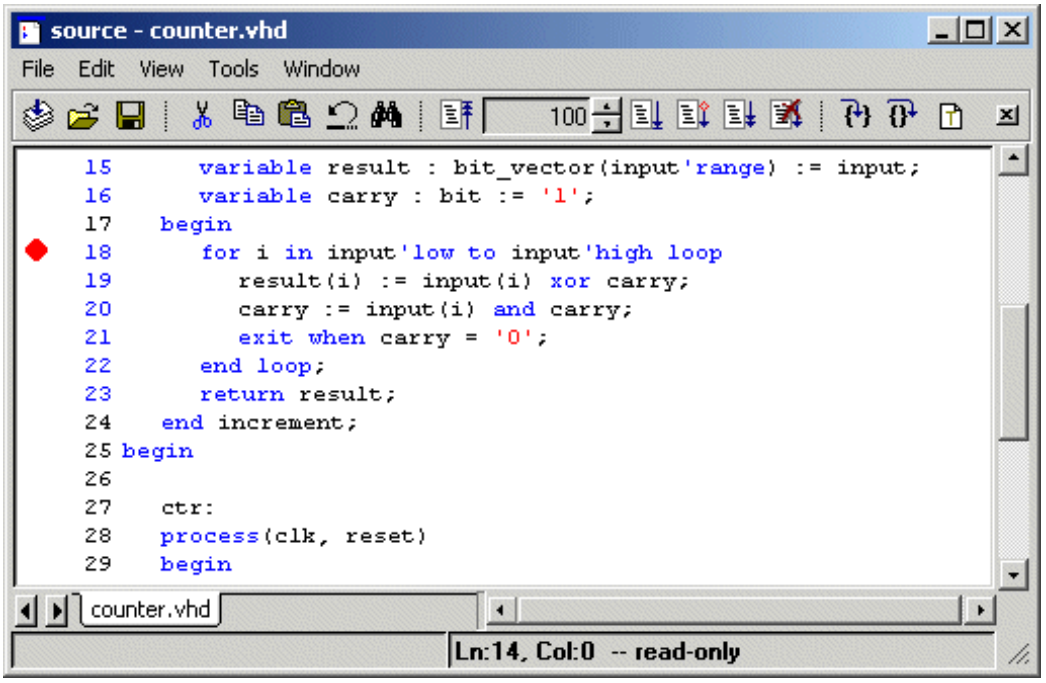
   (Main MENU: Simulate > Break) 

   The arrow in the Source window points to the next HDL statement to be executed. (If the simulator is not evaluating a process at the time the Break occurs, no arrow will be displayed in the Source window.)

4 Next, you will set a breakpoint in the function on line 18. Scroll the Source window until line 18 is visible. Click on or near line number 18 to set the breakpoint.

You should see a red dot next to the line number where the breakpoint is set. The breakpoint can be toggled between enabled and disabled by clicking it. When a breakpoint is disabled, the dot appears open. To delete the breakpoint, click the line number with your right mouse button and select Remove Breakpoint 18.

(PROMPT: bp counter.vhd 18)



Note: Breakpoints can be set only on executable lines, denoted by blue line numbers.

5 Select the **Continue Run** button to resume the run that you interrupted. ModelSim will hit the breakpoint, as shown by an arrow in the Source window and by a Break message in the Main window.

(PROMPT: run -continue) (MENU: Simulate > Run > Continue) 

6 Click the **Step** button in the Main or Source window several times to single-step through the simulation. Notice that the values change in the Variables window (you may need to expand the Variables window).

(PROMPT: step) (MENU: Simulate > Run > Step) 

7 This concludes the basic VHDL simulation tutorial. When you're done, quit the simulator by entering the command:

```
quit -force
```

This command exits ModelSim without asking for confirmation.

Lesson 3 - Basic Verilog simulation

The goals for this lesson are:

- Compile a Verilog design
- List signals in the design
- Examine the hierarchy of the design
- Simulate the design
- Change the default run length
- Set a breakpoint

The project feature covered in Lesson 1 executes several actions automatically such as creating and mapping work libraries. In this lesson we will go through the entire process so you get a feel for how ModelSim really works.

Compiling the design

If you've completed any previous VHDL lesson, you'll notice that Verilog and VHDL simulation processes are almost identical.

1. Create and change to a new directory to make it the current directory.

   You can make the directory current by invoking ModelSim from the new directory or by using the **File > Change Directory** command from the ModelSim Main window.

2. Copy the Verilog files (files with ".v" extension) from the *\<install_dir>\modeltech\examples* directory into the current directory.

   Before you can compile a Verilog design, you need to create a design library in the new directory. If you are familiar only with interpreted Verilog simulators such as Cadence Verilog-XL, this will be a new idea for you. Since ModelSim is a compiled Verilog simulator, it requires a target design library for the compilation. ModelSim can compile both VHDL and Verilog code into the same library if desired.

3. Invoke ModelSim:

   for UNIX at the shell prompt:

   ```
   vsim
   ```

   for Windows - your option - from a Windows shortcut icon, from the Start menu, or from a DOS prompt:

   ```
   modelsim.exe
   ```

   Note: If you didn't add Model*Sim* to your search path during installation, you will have to include the full path when you type this command at a DOS prompt.

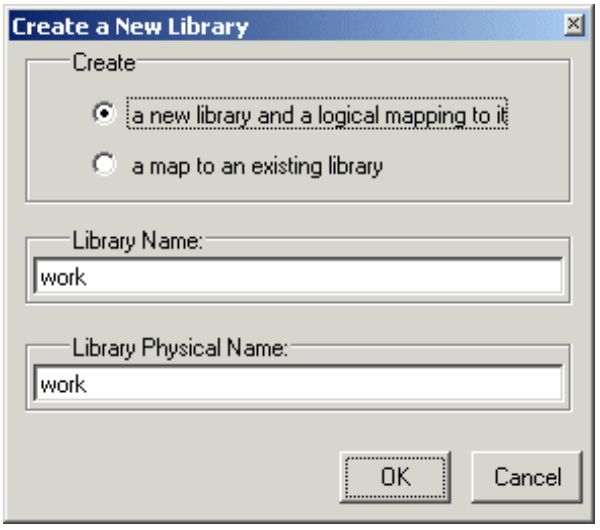
   Click **Close** if the Welcome dialog appears.

4 Before you compile any HDL code, you'll need a design library to hold the compilation results. To create a new design library, make this menu selection in the Main window: **File > New > Library**.

Make sure **Create: a new library and a logical mapping to it** is selected. Type "work" in the Library Name field and then select **OK**.

This creates a subdirectory named *work* - your design library - within the current directory. ModelSim saves a special file named *_info* in the subdirectory.

(PROMPT: vlib work
vmap work work)



Note: Do not create a Library directory using UNIX or Windows commands, because the *_info* file will not be created. Always use the File menu or the **vlib** command from either the ModelSim or UNIX/DOS prompt.)

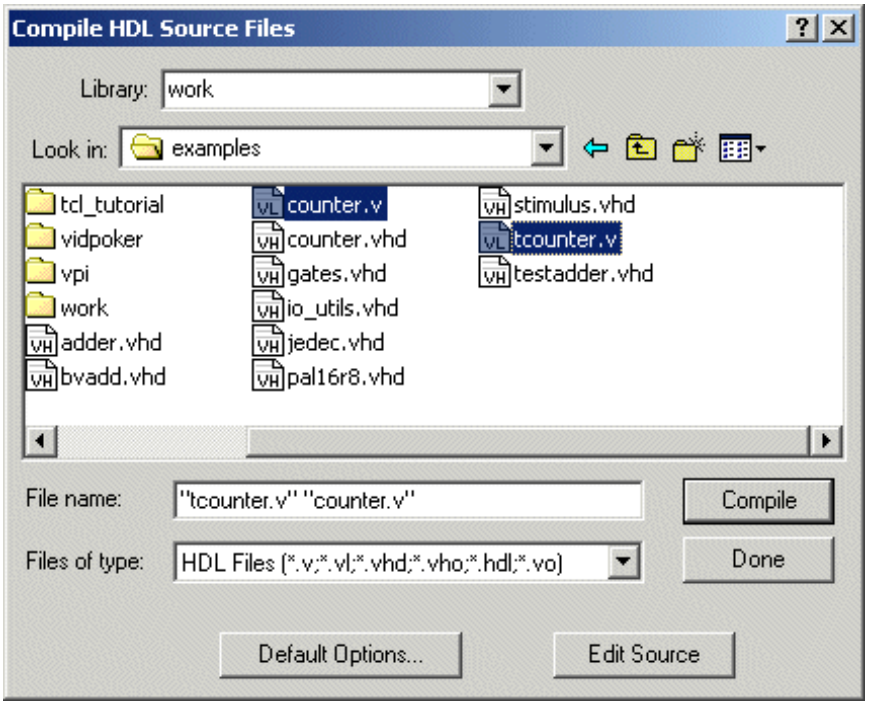
In the next step you'll compile the Verilog design. The example design consists of two Verilog source files, each containing a unique module. The file *counter.v* contains a module called **counter**, which implements a simple 8-bit binary up-counter. The other file, *tcounter.v*, is a testbench module (**test_counter**) used to verify **counter**.

Under simulation you will see that these two files are configured hierarchically with a single instance (instance name **dut**) of module **counter** instantiated by the testbench. You'll get a chance to look at the structure of this code later. For now, you need to compile both files into the **work** design library.

5 Compile the *counter.v*, and *tcounter.v* files into the **work** library by selecting **Compile > Compile** from the menu.

(PROMPT: vlog counter.v tcounter.v) 

This opens the Compile HDL Source Files dialog box.



Select *counter.v* and *tcounter.v* (use Ctrl + click) and then choose **Compile** and then **Done**.

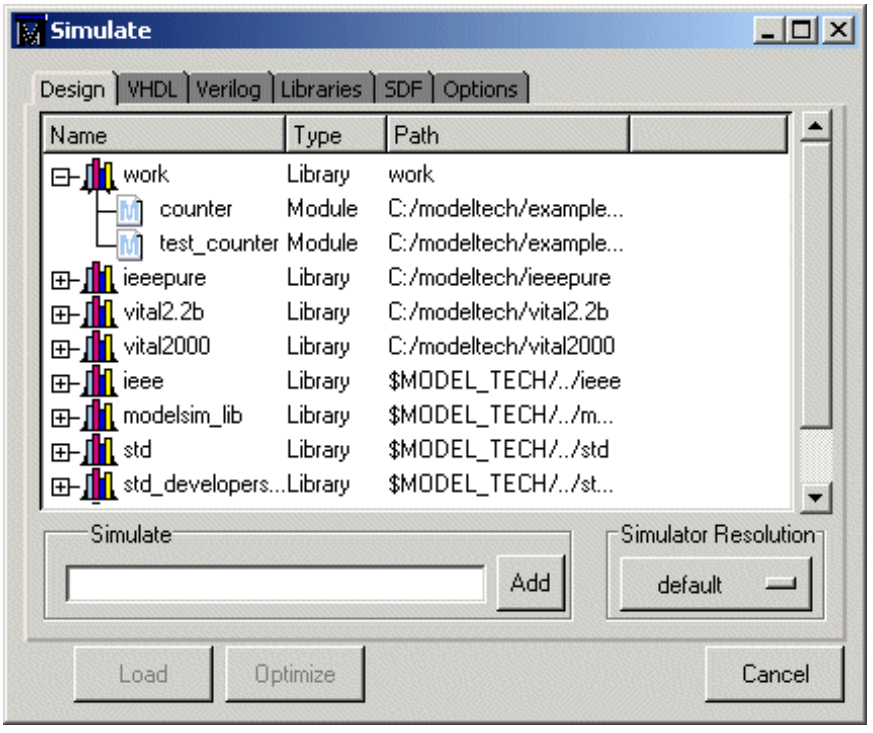
Note: The order in which you compile the two Verilog modules is not important (other than the source-code dependencies created by compiler directives). This may again seem strange to Verilog-XL users who understand the possible problems of interface checking between design units, or compiler directive inheritance. ModelSim defers such checks until the design is loaded. So it doesn't matter here if you choose to compile *counter.v* before or after *tcounter.v*.

Loading the design

1 Load the design by selecting **Simulate > Simulate** from the menu:

(PROMPT: vsim test_counter) 

The Simulate dialog appears. Click the "+" sign next to 'work' to see the **counter** and **test_counter** design units. (You won't see this dialog box if you invoke **vsim** with *test_counter* from the command line.)



The Simulate dialog allows you to select a design unit to load from the specified library. You can also select the resolution limit for the simulation. The default resolution is 1 ns.

Select **test_counter** and click **Load** to accept these settings.

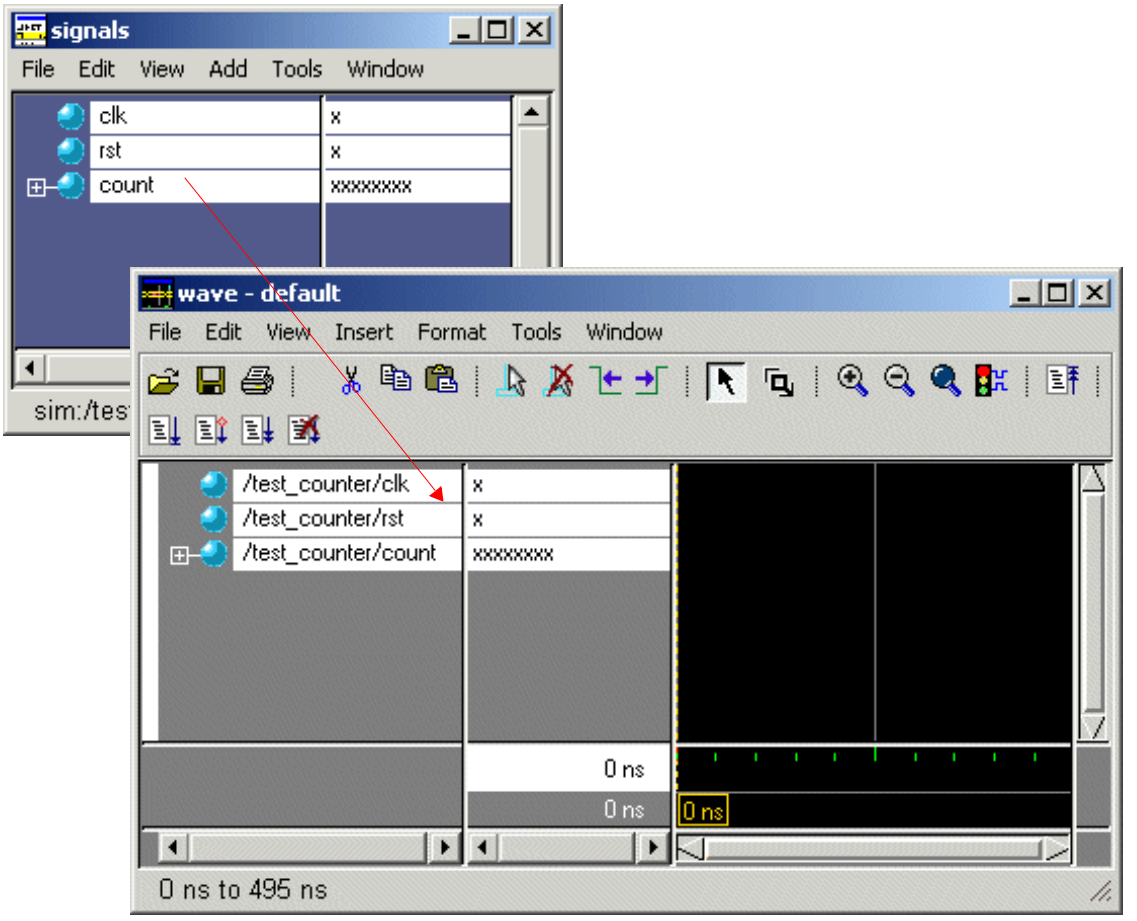
2 Bring up the Signals, Source, and Wave windows by entering the following command at the VSIM prompt within the Main window:

```
view signals source wave
```

(Main MENU: View > <window name>)

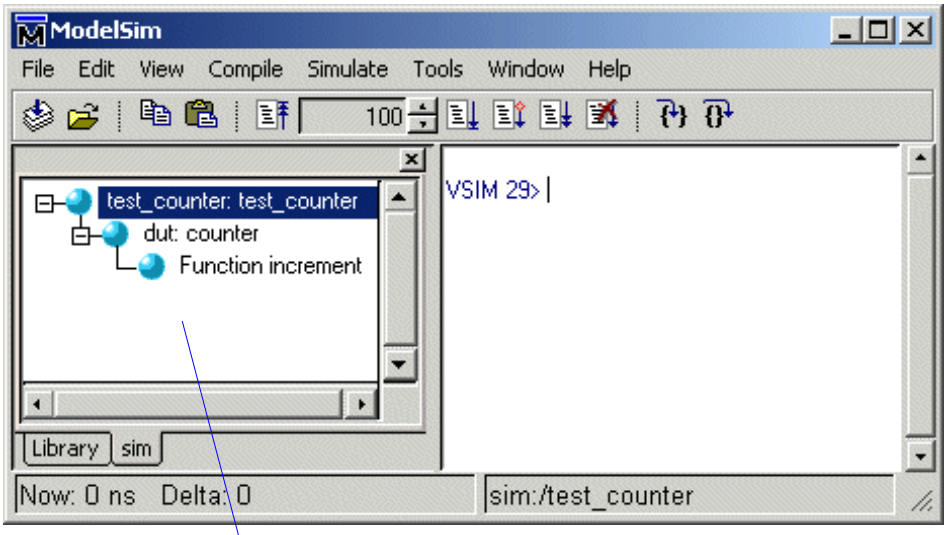
3 Now let's add signals to the Wave window with ModelSim's drag and drop feature.

In the Signals window, select **Edit > Select All** to select the three signals. Drag the signals to either the pathname or the values pane of the Wave window.



HDL items can also be copied from one window to another (or within the Wave and List windows) with the **Edit > Copy** and **Edit > Paste** menu selections.

4 You may have noticed when you loaded the design in Step 1 that a new tab appeared in the workspace area of the Main window.



The Structure tab shows the hierarchical structure of the design. By default, only the top level of the hierarchy is expanded. You can navigate within the hierarchy by clicking on any line with a "+" (expand) or "-" (contract) symbol. The same navigation technique works anywhere you find these symbols within ModelSim.

By clicking the "+" next to **dut: counter** you can see all three hierarchical levels: **test_counter**, **counter** and a function called **increment**. (If **test_counter** is not displayed you simulated **counter** instead of **test_counter**.)

5 Click on **Function increment** and notice how other ModelSim windows are automatically updated as appropriate. Specifically, the Source window displays the Verilog code at the hierarchical level you selected in the Structure window, and the Signals window displays the appropriate signals. Using the Structure tab in this way is analogous to scoping commands in interpreted Verilogs.

For now, make sure the **test_counter** module is showing in the Source window by clicking on the top line in the Structure pane.

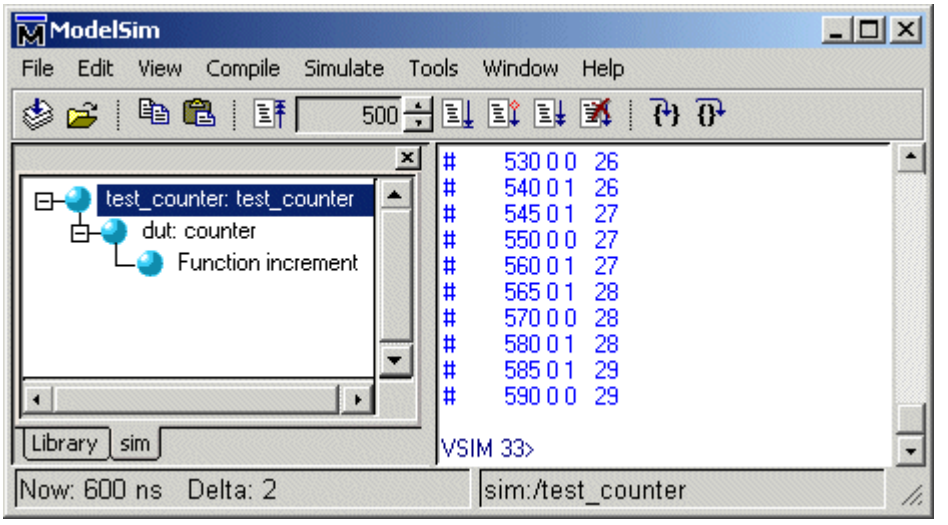
Running the simulation

Now you will exercise different Run functions from the toolbar.

1 Select the **Run** button on the Main window toolbar. This causes the simulation to run and then stop after 100 ns (the default simulation length).

(PROMPT: run) (MENU: Simulate > Run > Run 100 ns) 

2 Next change the run length to 500 on the **Run Length** selector and select the **Run** button again.



Now the simulation has run for a total of 600ns (the default 100ns plus the 500 you just asked for). The status bar at the bottom of the Main window displays this information.

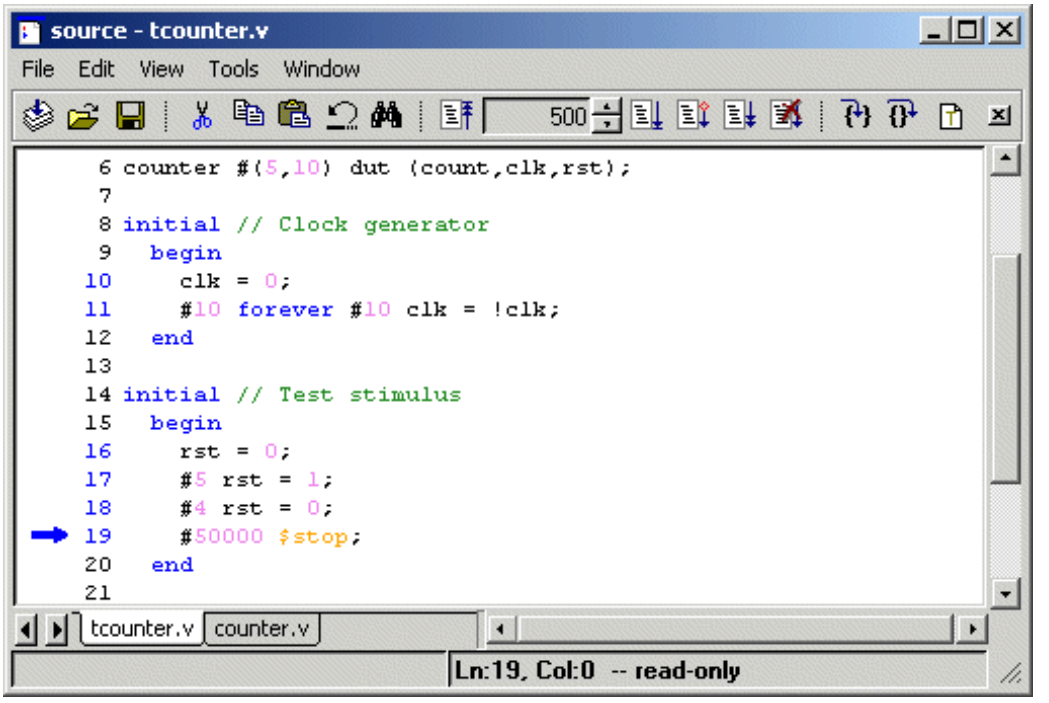
3 The last command you executed (**run 500**) caused the simulation to advance for 500ns. You can also advance simulation to a specific time. Type:

```
run @ 3000
```

This advances the simulation to time 3000ns. Note that the simulation actually ran for an additional 2400ns (3000 - 600).

4 Now select the **Run -All** button from the Main window toolbar. This causes the simulator to run until the stop statement in *tcounter.v*.

(PROMPT: run -all) (MENU: Simulate > Run > Run -All) 



You can also use the **Break** button to interrupt a run.

(MENU: Simulate > Break) 

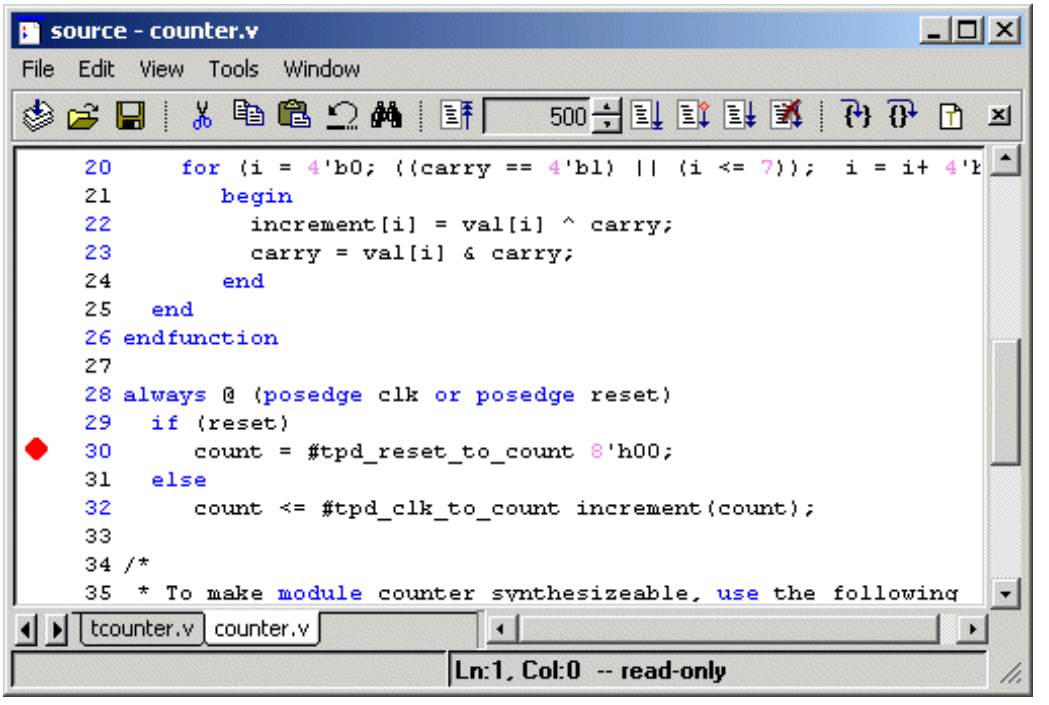
Debugging

Next we'll take a brief look at an interactive debugging feature of the ModelSim environment.

1 Let's set a breakpoint at line 30 in the *counter.v* file (which contains a call to the Verilog function increment). To do this, select **dut: counter** in the Structure pane of the Workspace. Move the cursor to the Source window and scroll the window to display line 30. Click on or near line number 30 to set a breakpoint. You should see a red dot next to the line number where the breakpoint is set.

The breakpoint can be toggled between enabled and disabled by clicking it. When a breakpoint is disabled, the dot appears open. To delete the breakpoint, click the line number with your right mouse button and select **Remove Breakpoint**.

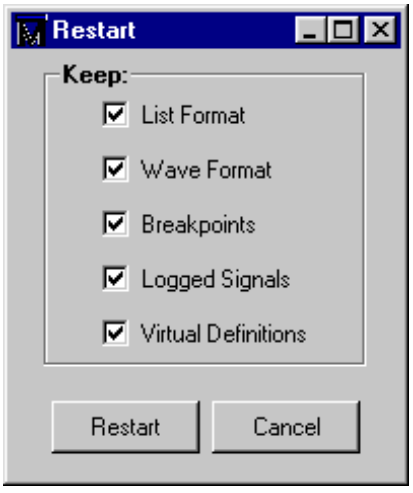
Note: Breakpoints can be set only on executable lines, denoted by blue line numbers.



2 Select the **Restart** button to reload the design elements and reset the simulation time to zero.

(Main MENU: Simulate > Run > Restart) (PROMPT: restart) 

Make sure all items in the Restart dialog box are selected, then click **Restart**.



3 Select the **Run -All** button to re-start the simulation run.

(PROMPT: run -all) (Main MENU: Simulate > Run > Run -All) 

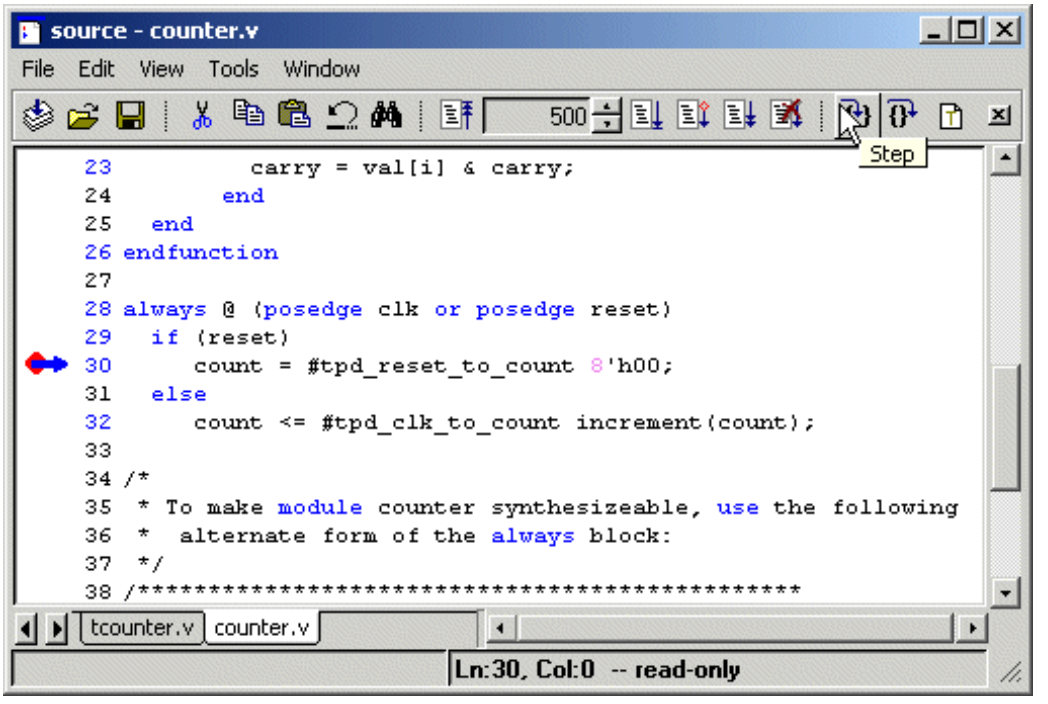
When the simulation hits the breakpoint, it stops running, highlights the line with an arrow in the Source window, and issues a Break message in the Main window.

4 When a breakpoint is reached, typically you will want to know one or more signal values. You have several options for checking values:

- look at the values shown in the Signals window
- hover your mouse pointer over the *count* variable in the Source window and a "balloon" will pop up with the value
- select the *count* variable in the Source window, right-click it, and select Examine from the context menu;
- use the **examine** command to output the value to the Main window transcript:

```
examine count
```

5 Let's move through the Verilog source functions with ModelSim's Step command. Click **Step** on the toolbar.



This command single-steps the debugger.

6 Experiment by yourself for awhile. Set and clear breakpoints and use the Step and Step Over commands until you feel comfortable with their operation. When you're done, quit the simulator by entering the command:

```
quit -force
```

Lesson 4 - Mixed VHDL/Verilog simulation

The goals for this lesson are:

- Compile multiple VHDL and Verilog files
- Simulate a mixed VHDL and Verilog design
- View the design in the Structure window
- View the HDL source code in the Source window

Note: You must be using ModelSim SE/PLUS or ModelSim SE/MIXED to do this lesson.

Compile the design

1 Start by creating a new directory for this exercise. Create the directory, then copy the VHDL and Verilog example files to the directory:

```
<install_dir>\modeltech\examples\mixedHDL\*.vhd
<install_dir>\modeltech\examples\mixedHDL\*.v
```

Make sure the new directory is the current directory. Do this by invoking ModelSim from the new directory or by using the **File > Change Directory** command from the ModelSim Main window.

2 Start ModelSim with one of the following:

for UNIX at the shell prompt:

```
vsim
```

for Windows - your option - from a Windows shortcut icon, from the Start menu, or from a DOS prompt:

```
modelsim.exe
```

Note: If you didn't add ModelSim to your search path during installation, you will have to include the full path when you type this command at a DOS prompt.

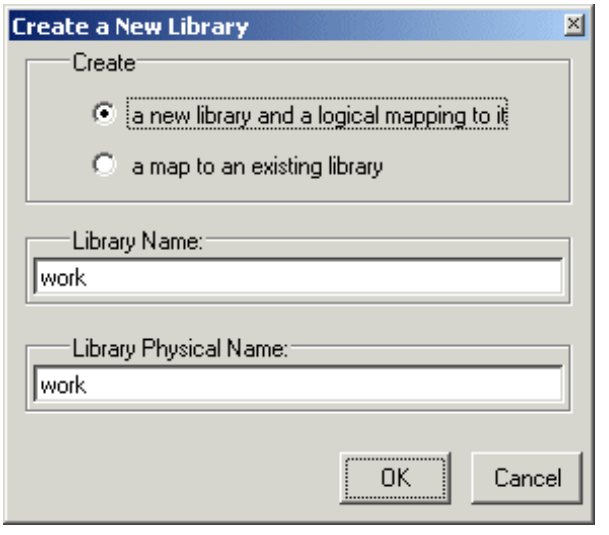
Select "Proceed to ModelSim" if the Welcome dialog appears.

3 Before you compile any HDL code, you'll need a design library to hold the compilation results. To create a new design library, make this menu selection in the Main window: **File > New > Library**.

Make sure **Create: a new library and a logical mapping to it** is selected. Type "work" in the Library Name field and then select **OK**.

This creates a subdirectory named *work* - your design library - within the current directory. ModelSim saves a special file named *_info* in the subdirectory.

(PROMPT: vlib work
vmap work work)



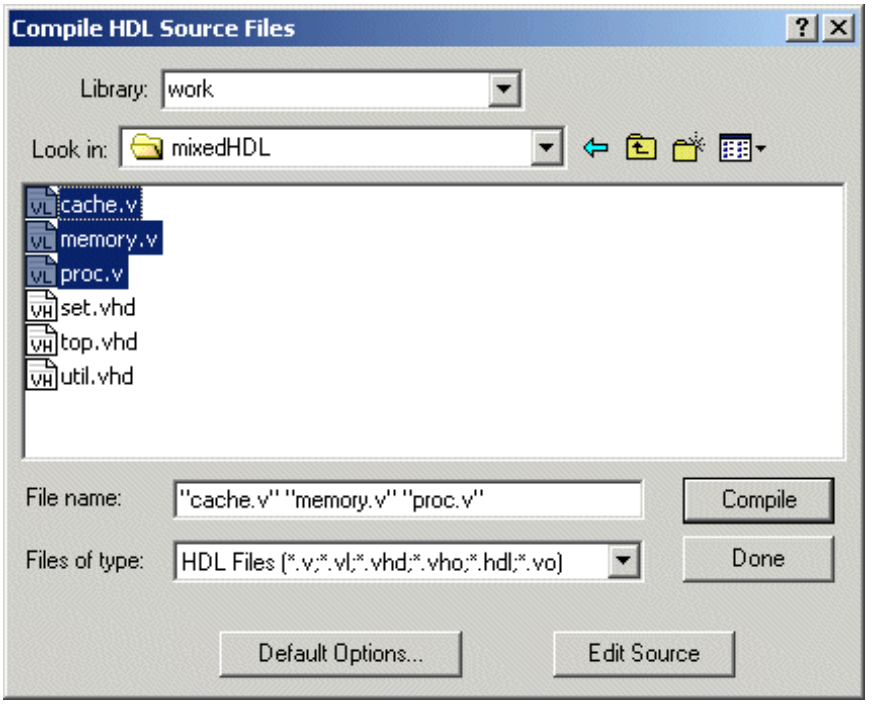
Note: Do not create a Library directory using UNIX or Windows commands, because the *_info* file will not be created. Always use the File menu or the **vlib** command from either the ModelSim or UNIX/DOS prompt.)

4 Compile the HDL files by selecting **Compile** > **Compile** from the menu:

(PROMPT: vlog cache.v memory.v proc.v)

(PROMPT: vcom util.vhd set.vhd top.vhd)

This opens the Compile HDL Source Files dialog box.



A group of Verilog files may be compiled in any order. However, in a mixed VHDL/Verilog design the Verilog files must be compiled before the VHDL files.

Compile the Verilog source by double-clicking each of these Verilog files in the file list (this invokes the Verilog compiler, **vlog**):

- *cache.v*
- *memory.v*
- *proc.v*

5. Depending on the design, the compile order of VHDL files can be very specific. In the case of this lesson, the file *top.vhd* must be compiled last.

   Stay in the Compile HDL Source Files dialog box and compile the VHDL files in this order (this invokes the VHDL compiler, **vcom**):

   - *util.vhd*
   - *set.vhd*
   - *top.vhd*

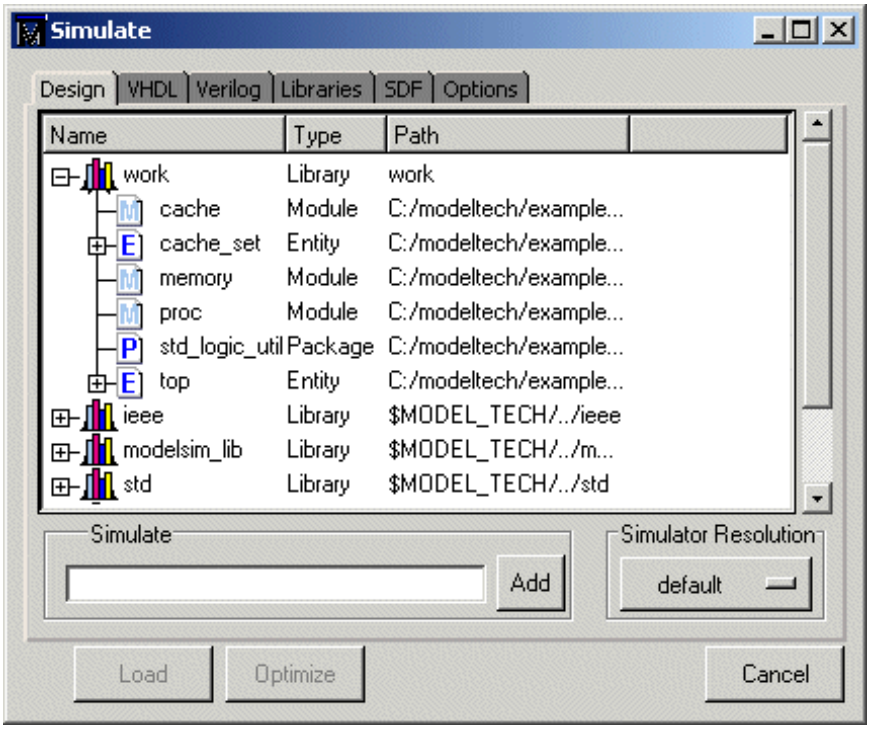
6. Click **Done** to dismiss the dialog box.

Loading the design

1 Load the design by selecting **Simulate > Simulate** from the menu.

(PROMPT: vsim top) 

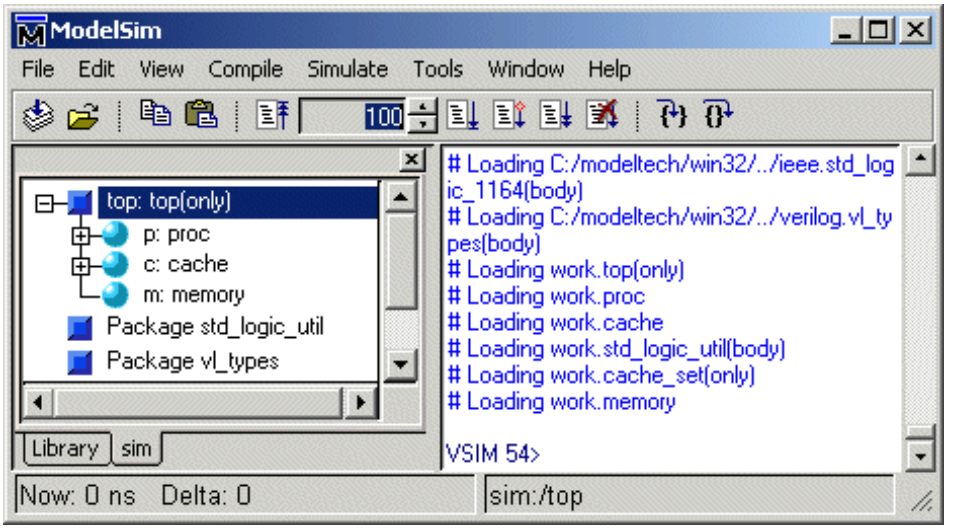
The Simulate dialog appears. Click the "+" sign next to 'work' to see the design units. (You won't see this dialog box if you invoke **vsim** with *top* from the command line.) Select **top** and then click **Load**.



2 From the Main menu select **View > All Windows** to open all ModelSim windows.

(PROMPT: view *)

3 Take a look at the Structure pane in the workspace.



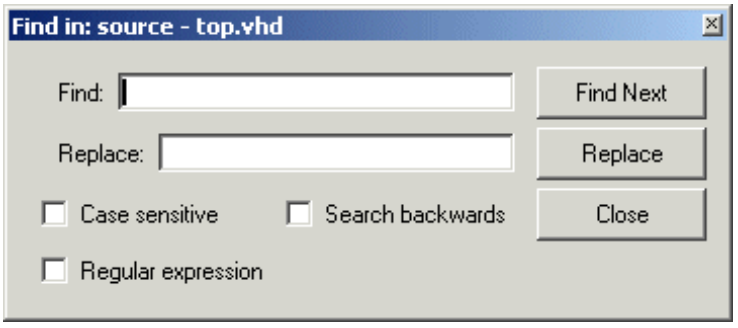
Notice the hierarchical mixture of VHDL and Verilog in the design. VHDL levels are indicated by a square “prefix”, while Verilog levels are indicated by a circle “prefix.” Try expanding (+) and contracting (-) the structure layers. You’ll find Verilog modules that have been instantiated by VHDL architectures, and similar instantiations of VHDL items by Verilog.

4 In the Structure pane, click on the Verilog module **c: cache**. The source code for the Verilog module is now shown in the Source window.

5 We'll use ModelSim's Find function to locate the declaration of cache_set within *cache.v*.

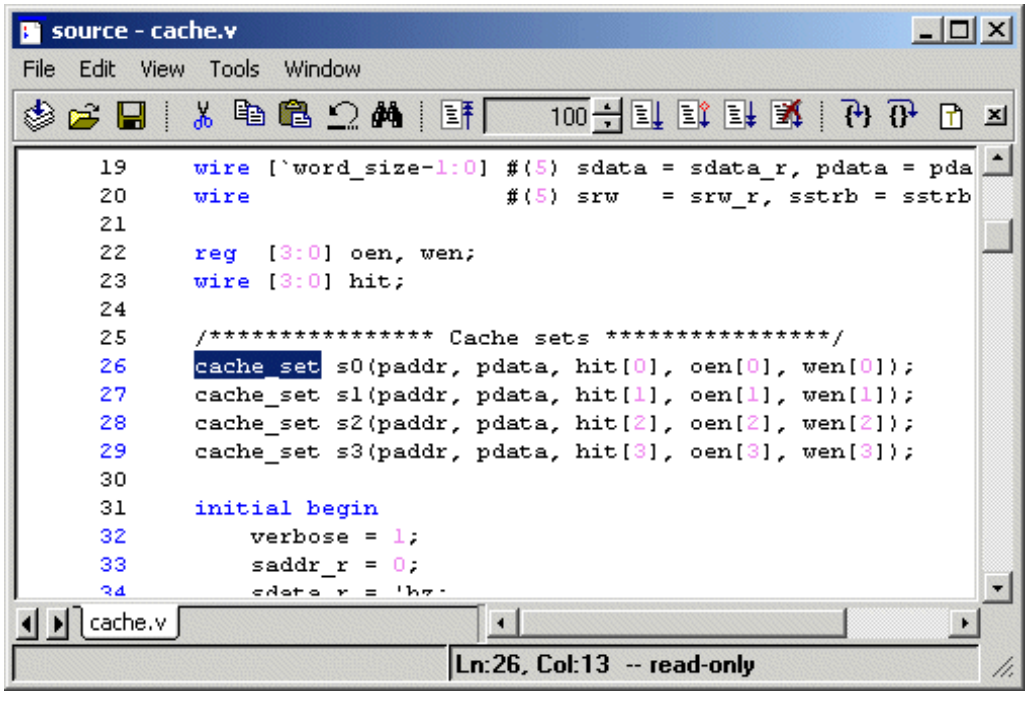
From the Source window menu select: **Edit > Find**: or 

The **Find in** dialog box is displayed.

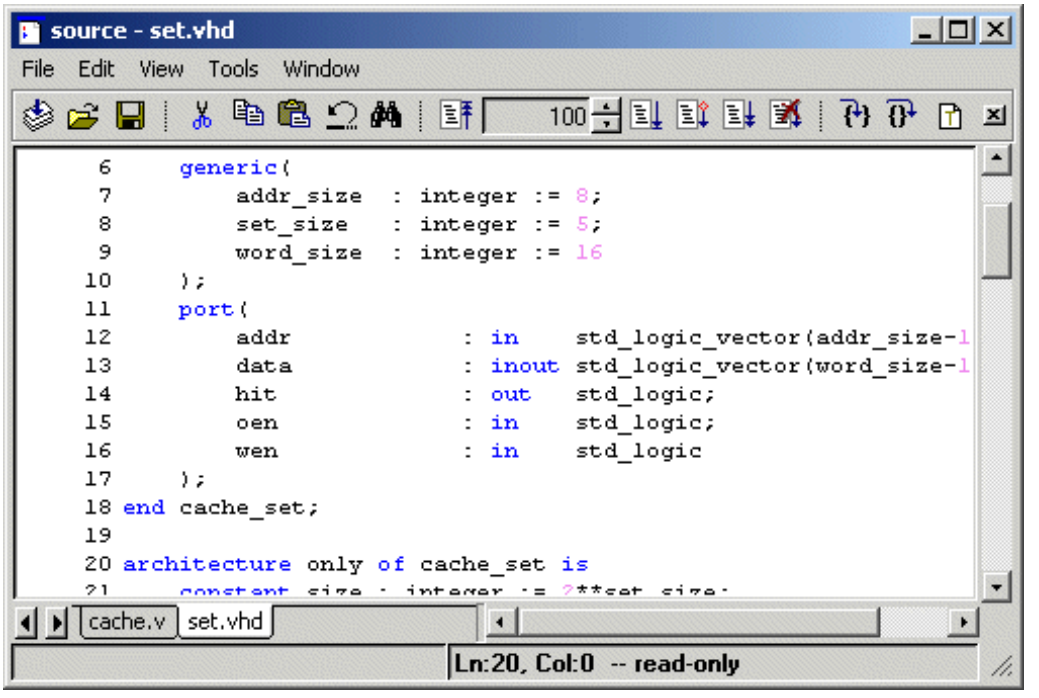


In the **Find:** field, type *cache_set* and click **Find Next**. The *cache_set* instantiations are now displayed in the Source window. (Click **Close** to dismiss the **Find in:** dialog box.)

Note that cache_set is a VHDL entity instantiated within the Verilog file *cache.v*.



6 Go back to the Main window, expand the **c:cache** entry by clicking the "+" sign, and scroll down and click on the line **"s0: cache_set(only)"**. The Source window shows the VHDL code for the cache_set entity.



Before you quit, try experimenting with some of the commands you've learned from previous lessons – add signals to the Wave window, run the simulation, etc. Note that in this design, "clk" is already driven, so you won't need to use the **force** command.

7 When you're ready to quit simulating, enter the command:

```
quit -force
```

Lesson 5 - Debugging a VHDL design

The goals for this lesson are:

- Map a logical library name to an actual library
- Change the default run length
- Recognize assertion messages in the Main window transcript
- Change the assertion break level
- Restart the simulation run using the **restart** command
- Examine composite types displayed in the Variables window
- Change the value of a variable

In this lesson we will debug an assertion message using the Source, Signals, and Variables windows. For another debugging lesson, see *Lesson 11 - Debugging with the Dataflow window*.

Compiling and loading the design

1. Create a new directory for this exercise and copy the following VHDL (.vhd) files from \<*install_dir*>*modeltech**examples* to the new directory.
   - gates.vhd
   - adder.vhd
   - testadder.vhd

2. Make sure the new directory is the current directory. Do this by invoking ModelSim from the new directory or by using the **File > Change Directory** command from the ModelSim Main window.

3. Start ModelSim with one of the following:

   for UNIX at the shell prompt:

   ```
   vsim
   ```

   for Windows - your option - from a Windows shortcut icon, from the Start menu, or from a DOS prompt:

   ```
   modelsim.exe
   ```

   Note: If you didn't add ModelSim to your search path during installation, you will have to include the full path when you type this command at a DOS prompt.

4. Enter the following command at the ModelSim prompt in the Main window to create a new library:

   ```
   vlib library_2
   ```

5. Map the new library to the work library using the **vmap** command:

   ```
   vmap work library_2
   ```

   ModelSim adds this mapping to the *modelsim.ini* file.

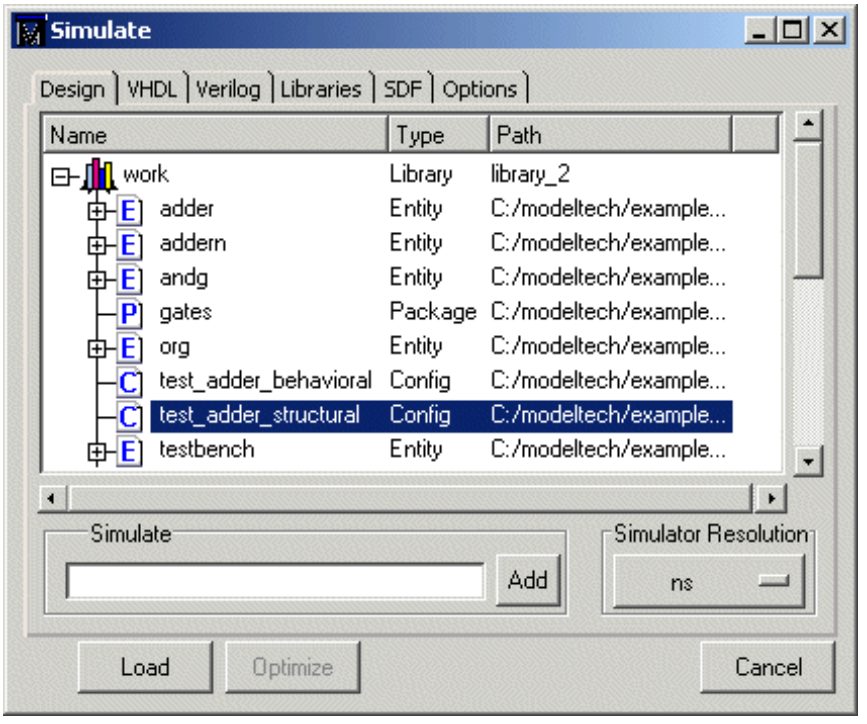
6. Compile the source files into the new library by entering this command at the ModelSim prompt:

   ```
   vcom -work library_2 gates.vhd adder.vhd testadder.vhd
   ```

7 Open the Simulate dialog by selecting **Simulate > Simulate**. Expand the work library and increase the width of the name column by clicking and dragging on the border between the Name and Type columns.

8 Make sure Simulator Resolution is set to nanoseconds, select **test_adder_structural**, and then click **Load**.

(PROMPT: vsim -t ns work.test_adder_structural)



Running the simulation

1. Start by opening the Process, Variables, and Signals windows using the command below. Note that you can abbreviate window names.

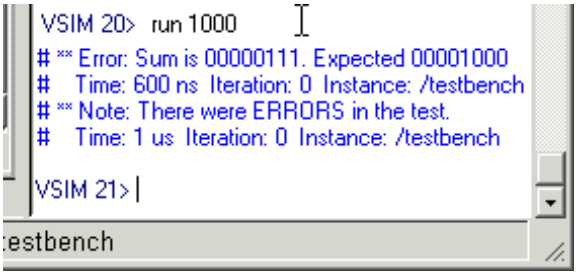
   ```
   view p si v
   ```

   (Main MENU: View > <window name>)

2. Now run the simulation for 1000 ns:

   ```
   run 1000
   ```

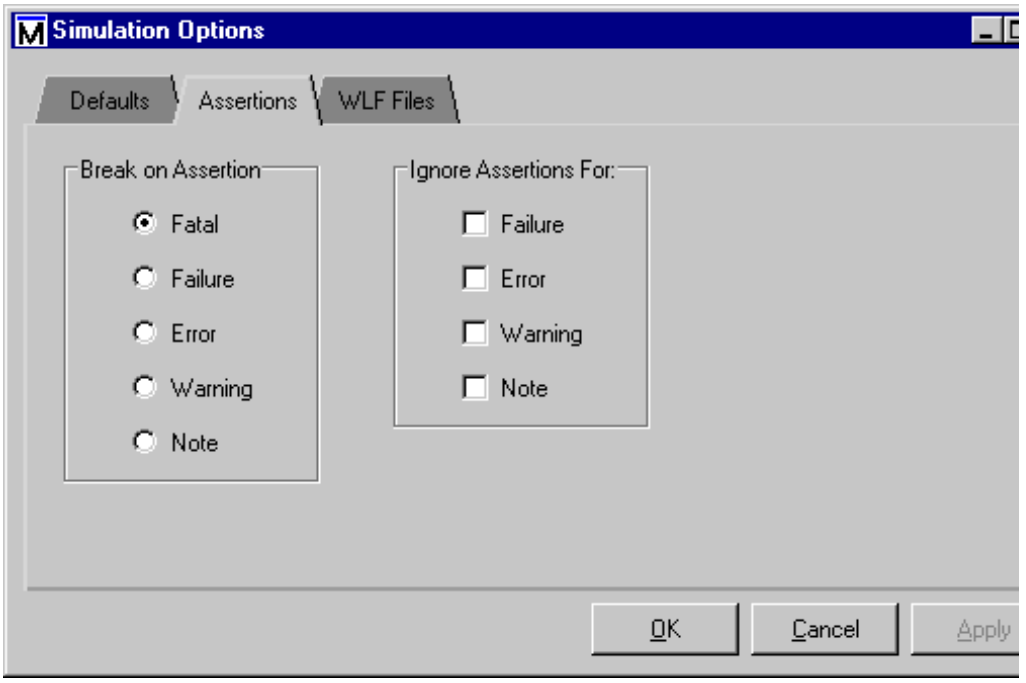
   A message in the Main window will notify you that there was an assertion error.



Debugging the simulation

Let's find out what's wrong. Perform the following steps to track down the assertion message.

1 First, change the simulation assertion options. Select **Simulate > Simulation Options** from the Main window menu.



2 Select the **Assertions** tab. Change the selection for **Break on Assertion** to **Error** and click **OK**. This will cause the simulator to stop at the HDL assertion statement.

3 Restart the simulation using the following command:

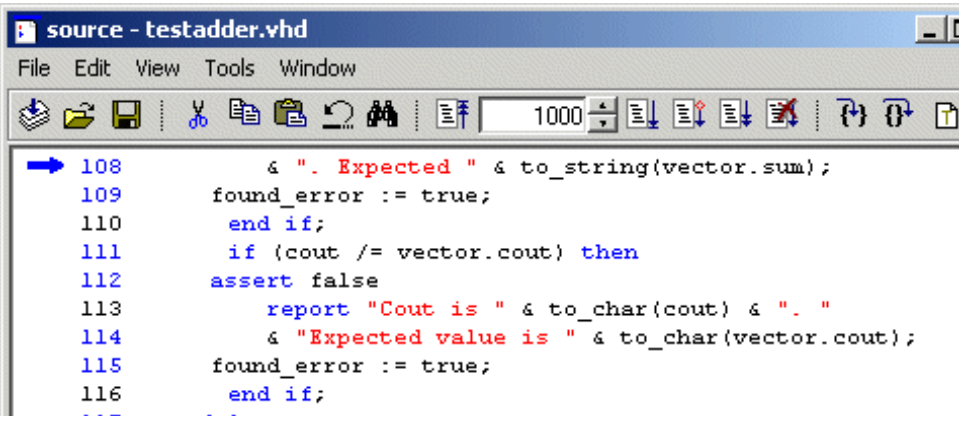
```
restart -f
```

The **-f** option causes ModelSim to restart without popping up the confirmation dialog.

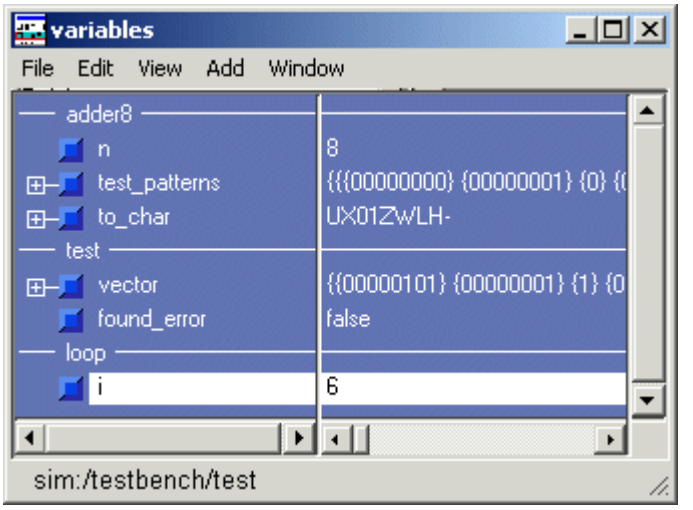
4 Run the simulation again for 1000 ns.

```
run 1000
```

Notice that the arrow in the Source window is pointing to the assertion statement.



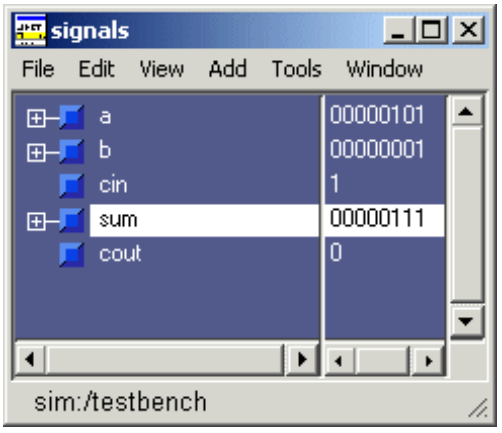
5 If you look at the Variables window now, you can see that i = 6. This indicates that the simulation stopped in the sixth iteration of the test pattern's loop.

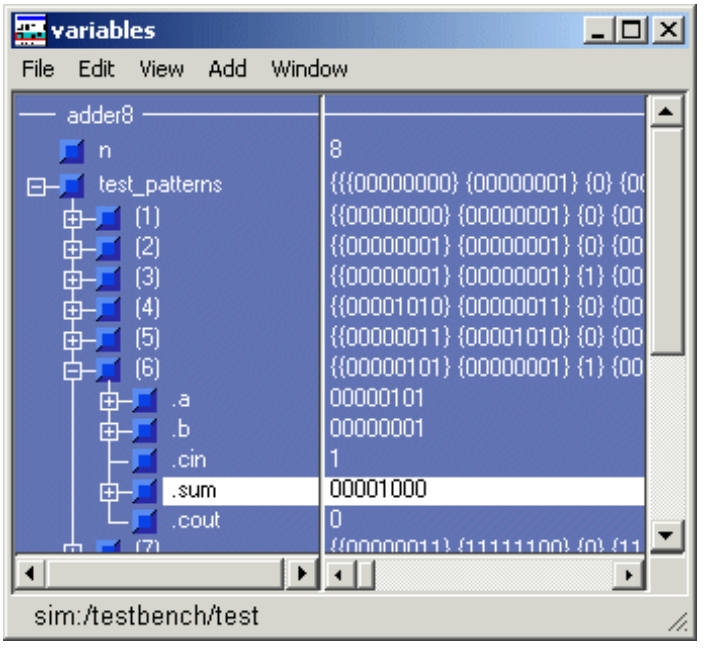


6 Expand the variable named **test_patterns** by clicking the [+]. (You may need to resize the window for a better view.)

7 Also expand the sixth record in the array **test_patterns(6)**, by clicking the [+]. The Variables window should be similar to the one below.

The assertion shows that the Signal **sum** does not equal the **sum** field in the Variables window. Note that the sum of the inputs **a**, **b**, and **cin** should be equal to the output **sum**. But there is an error in the test vectors. To correct this error, you need to restart the simulation and modify the initial value of the test vectors.



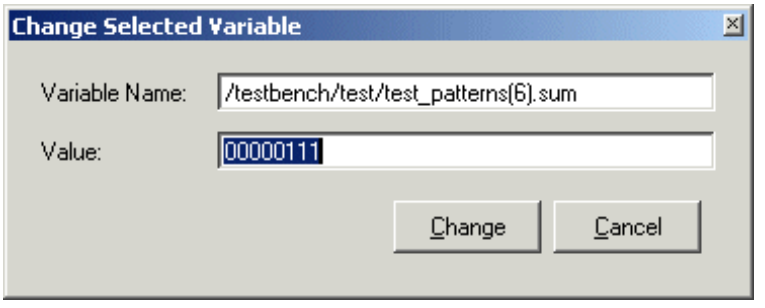


8 Restart the simulation again:

```
restart -f
```

9 Update the Variables window by selecting the **test** process in the Process window.

10 In the Variables window, expand **test_patterns** and **test_pattern(6)** again. Then highlight the **.sum** record by clicking on the variable name (not the box before the name) and select **Edit > Change** from the menu.

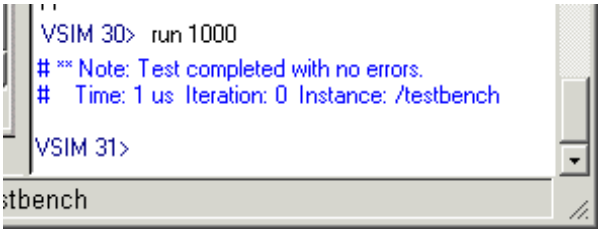


11 Change the value to **00000111** and then click **Change**. (Note that this is a temporary edit, you must use your text editor to permanently change the source code.)

12 Run the simulation again for 1000 ns.

```
run 1000
```

At this point, the simulation will run without errors.



This brings you to the end of this lesson, but feel free to explore the system further. When you are ready to end the simulation session, quit ModelSim by entering the following command at the VSIM prompt:

```
quit -f
```

Lesson 6 - Finding names and values

The goals for this lesson are:

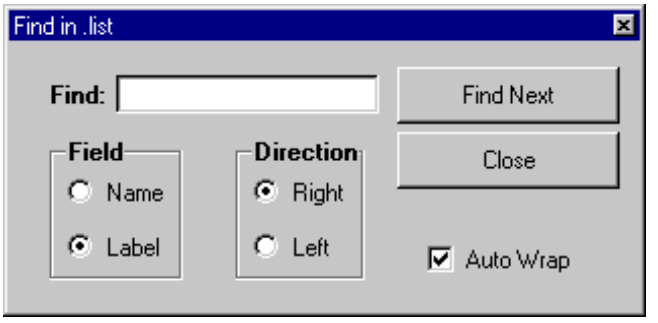
- Find items by name in tree windows
- Search for item values in the List and Wave windows

Start any of the lesson simulations to try out the Find and Search functions illustrated below.

Finding items by name in tree windows

You can find HDL item names with the **Edit > Find** menu selection in these windows: Dataflow, List, Process, Signals, Source, Structure, Variables, and Wave windows.

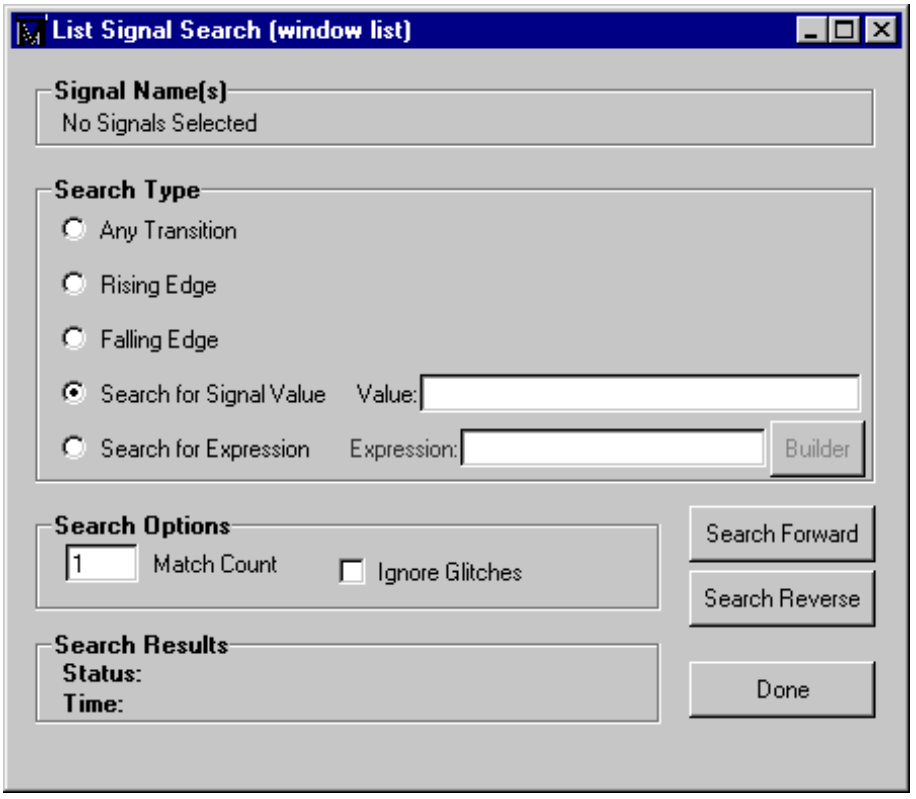
Select **Edit > Find** to bring up the Find dialog box (List window version shown).



Enter an item label and **Find** it by searching **Right** or **Left** through the window display.

Searching for item values in the List and Wave windows

You can search for HDL item values in the List and Wave windows. Select **Edit > Search** from the window's menu to bring up the Signal Search dialog box (List window version shown).



You can locate values for the **Signal Name(s)** shown at the top of the dialog box. The search is based on these options:

- **Search Type: Any Transition**
  Searches for any transition in the selected signal(s).
- **Search Type: Rising Edge**
  Searches for rising edges in the selected signal(s).
- **Search Type: Falling Edge**
  Searches for falling edges in the selected signal(s).
- **Search Type: Search for Signal Value**
  Searches for the value specified in the **Value** field; the value should be formatted using VHDL or Verilog numbering conventions.

- **Search Type: Search for Expression**
  Searches for the expression specified in the **Expression** field evaluating to a boolean true. Activates the **Builder** button so you can use the Expression Builder if desired.

  The expression may involve more than one signal but is limited to signals logged in the List or Wave window. Expressions may include constants, variables, and Tcl macros. If no expression is specified, the search will give an error. See the *ModelSim Command Reference* for more information on expression syntax.
- **Search Options: Match Count**
  You can search for the n-th transition or the n-th match on value; **Match Count** indicates the number of transitions or matches to search.
- **Search Options: Ignore Glitches**
  Ignore zero width glitches in VHDL signals and Verilog nets.

The result of your search is indicated at the bottom of the dialog box.

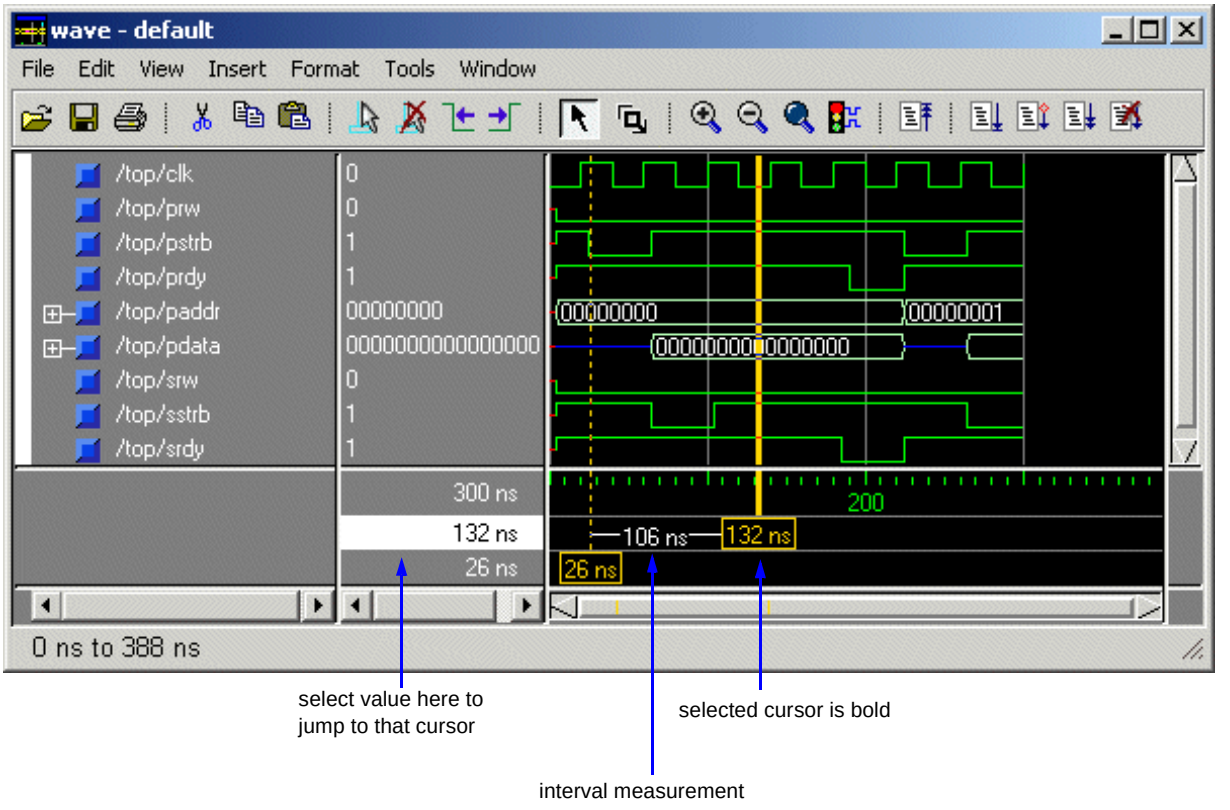
Lesson 7 - Using the Wave window

The goals for this lesson are:

- Practice using the Wave window time cursors.
- Practice zooming the waveform display.
- Practice using Wave window keyboard shortcuts.
- Practice combining items into a virtual object.
- Practice creating and viewing datasets.

Using time cursors in the Wave window

Any of the previous lesson simulations may be used with this part of the lesson, or use your own simulation if you wish.

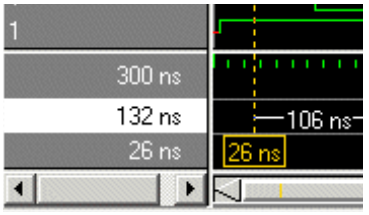


When the Wave window is first drawn, there is one cursor located at time zero. Clicking anywhere in the waveform display brings that cursor to the mouse location. You can add cursors to the waveform pane by selecting **Insert > Cursor** (or the Add Cursor button shown below). The selected cursor is drawn as a bold solid line; all other cursors are drawn with thin dashed lines. Remove cursors by selecting them and selecting **Edit > Delete Cursor** (or the Delete Cursor button shown below).

Add Cursor add a cursor to the Wave window	**Delete Cursor** delete the selected cursor from the window

Finding a cursor

The cursor value corresponds to the simulation time of that cursor. Choose a specific cursor view by selecting **View > Cursors** (Wave window). You can also select and scroll to a cursor by double-clicking its value in the cursor-value pane.



Alternatively, you can click a value with your second mouse button and type the value to which you want to scroll.

Making cursor measurements

Each cursor is displayed with a time box showing the precise simulation time at the bottom. When you have more than one cursor, each time box appears in a separate track at the bottom of the display. ModelSim also adds a delta measurement showing the time difference between two adjacent cursor positions.

If you click in the waveform display, the cursor closest to the mouse position is selected and then moved to the mouse position. Another way to position multiple cursors is to use the mouse in the time box tracks at the bottom of the display. Clicking anywhere in a track selects that cursor and brings it to the mouse position.

Cursors will "snap" to a waveform edge if you click or drag a cursor to within ten pixels of waveform edge. You can set the snap distance in the Window Preferences dialog (select **Tools > Window Preferences**). You can position a cursor without snapping by dragging in the area below the waveforms.

You can also move cursors to the next transition of a signal with these toolbar buttons:

 **Find Previous Transition** locate the previous signal value change for the selected signal	 **Find Next Transition** locate the next signal value change for the selected signal

Zooming - changing the waveform display range

Zooming lets you change the simulation range in the waveform pane. You can zoom using a context menu, toolbar buttons, mouse, keyboard, or commands.

Using the Zoom menu

You can access Zoom commands from the **View** menu on the toolbar or by clicking the right mouse button in the waveform pane.

The Zoom menu options include:

- **Zoom In**
  Zooms in by a factor of two, increasing the resolution and decreasing the visible range horizontally. (command: .wave.treee zoomin)
- **Zoom Out**
  Zooms out by a factor of two, decreasing the resolution and increasing the visible range horizontally. (command: .wave.treee zoomout)
- **Zoom Full**
  Redraws the display to show the entire simulation from time 0 to the current simulation time. (command: .wave.treee zoomfull)
- **Zoom Last**
  Restores the display to where it was before the last zoom operation.
  (command: .wave.treee zoomlast)
- **Zoom Range**
  Brings up a dialog box that allows you to enter the beginning and ending times for a range of time units to be displayed. (command: .wave.treee zoomrange)

Zooming with toolbar buttons

These zoom buttons are available on the toolbar:

 **Zoom in 2x** zoom in by a factor of two from the current view	 **Zoom out 2x** zoom out by a factor of two from current view
 **Zoom Full** zoom out to view the full range of the simulation from time 0 to the current time	 **Zoom Mode** change mouse pointer to zoom mode; see below

Zooming with the mouse

To zoom with the mouse, first enter zoom mode by selecting **View > Mouse Mode > Zoom Mode** (Wave window). The left mouse button (<Button-1>) then offers 3 zoom options by clicking and dragging in different directions:

- Down-Right: Zoom Area (In)
- Up-Right: Zoom Out
- Up-Left: Zoom Fit

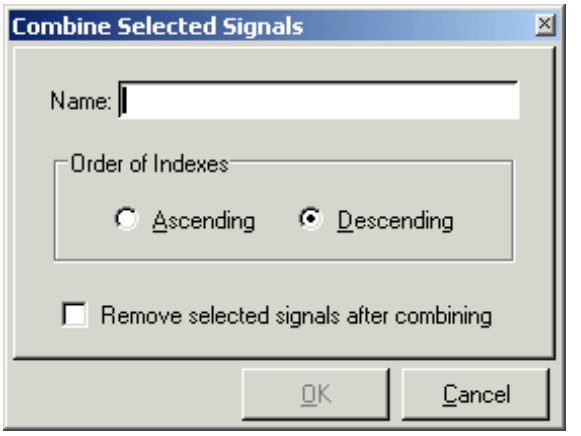
The zoom amount is displayed at the mouse cursor. A zoom operation must be more than 10 pixels to activate.

Keyboard shortcuts for zooming

Using the following keys when the mouse cursor is within the Wave window will cause the indicated actions:

Key	Action
i I or +	zoom in
o O or -	zoom out
f or F	zoom full
l or L	zoom last
r or R	zoom range
<arrow up>	scroll waveform display up
<arrow down>	scroll waveform display down
<arrow left>	scroll waveform display left
<arrow right>	scroll waveform display right
<page up>	scroll waveform display up by page
<page down>	scroll waveform display down by page
<tab>	searches forward (right) to the next transition on the selected signal
<shift-tab>	searches backward (left) to the previous transition on the selected signal
<Control-f> (Windows) <Control-s> (UNIX)	opens the find dialog box; searches within the specified field in the pathname pane for text strings

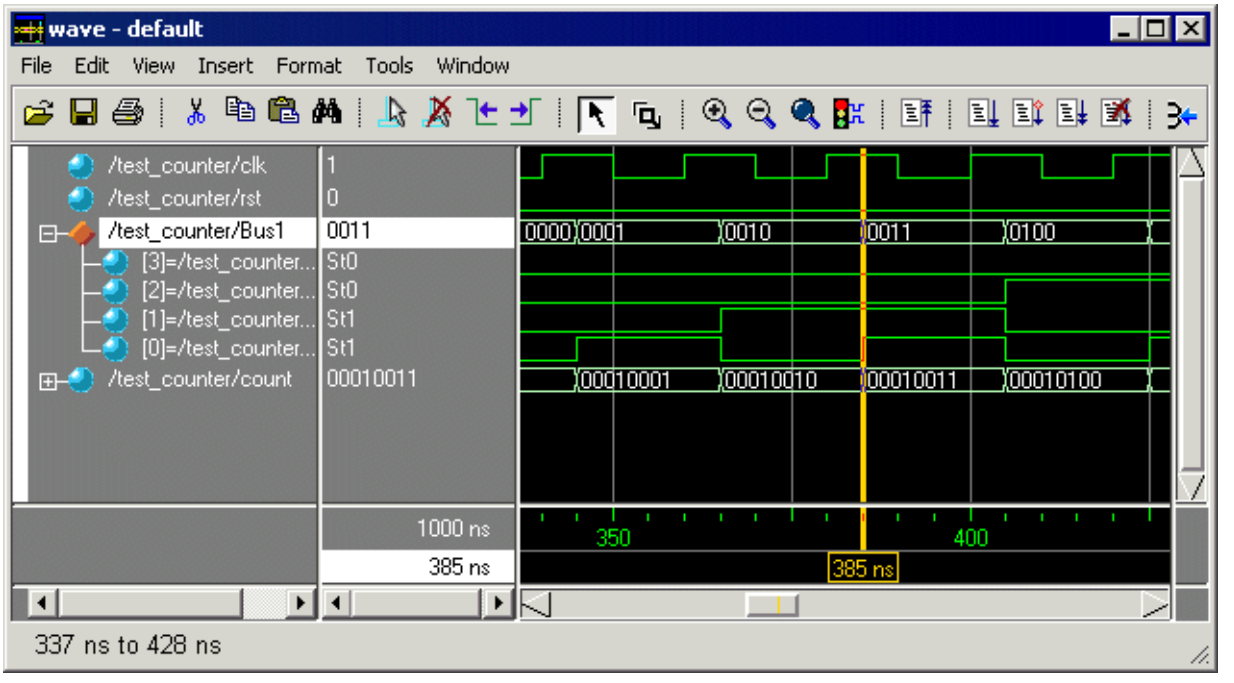
Combining items in the Wave window



The Wave window allows you to combine signals into buses. Select **Tools** > **Combine Signals** to open the Combine Selected Signals dialog.

A bus is a collection of signals concatenated in a specific order to create a new virtual signal with a specific value.

In the illustration below, four data signals have been combined to form a new bus called Bus1. Notice, the new bus has a value that is made up of the values of its component signals arranged in a specific order. Virtual objects are indicated by an orange diamond.



Creating and viewing datasets

Datasets allow you to view previous simulations or to compare simulations. To view a dataset, you must first save a ModelSim simulation to a WLF file (using the **vsim -wlf** command). Once you have saved a WLF file, you can open it as a view-mode dataset.

In this lesson you will compare two simple Verilog designs: a structural description and an RTL description of a 4-bit, binary counter. To begin, you will simulate the structural description and save it to a WLF file. Then you will simulate the RTL version. Finally, you will open the WLF file as a dataset and compare the two simulations in the Wave window.

Simulating the structural version

1. Start by creating a new working directory, making it the current directory, and copying the files from *\modeltech\examples\datasets* into it.

2. Use the **vlib** command to create a **work** library in the current directory.

   ```
   vlib work
   ```

   (MENU: File > New > Library)

3. Use the **vmap** command to map the work library to a physical directory. A *modelsim.ini* file will be written into the current directory.

   ```
   vmap work work
   ```

4. Compile the structural version of the counter.

   ```
   vlog cntr_struct.v
   ```

   (MENU: Compile > Compile) 

5. Load the design and save the simulation to a WLF file named *struct.wlf*.

   ```
   vsim -wlf struct.wlf work.cntr_struct
   ```

6. Now you will run a DO file that applies stimulus to the design, runs the simulation, and adds waves to the Wave window. Feel free to open the DO file and look at its contents.

   ```
   do stimulus.do
   ```

   (MENU: Tools > Execute Macro)

   The waves that appear in the Wave window are saved automatically into the *struct.wlf* file.

7. Quit the simulation.

   ```
   quit -sim
   ```

   (MENU: Simulate > End Simulation)

Simulating the RTL version

1 Compile the RTL version of the counter.

```
vlog cntr_rtl.v
```

2 Simulate the design.

```
vsim work.cntr_rtl
```

(MENU: Simulate > Simulate) 

3 Run the DO file to apply stimulus to the design.

```
do stimulus.do
```

Comparing the two designs

To compare the two simulations, we will create a second pane in the Wave window, open the *struct.wlf* file, and add the signals from the dataset to the new pane.

1 Add a second pane to the Wave window.

Wave MENU: Insert > Window Pane

Notice that a thick, white vertical bar at the left edge of the window indicates that the new pane is active.

2 Open *struct.wlf*.

```
dataset open struct.wlf
```

(Wave MENU: File > Open Dataset)

3 Add signals for the "struct" dataset.

```
add wave *
```

Notice that the pathname prefix for the signals you just added is the dataset name "struct". The pathname prefix for the active simulation is "sim".

The results for each simulation should be the same. You can continue experimenting with the two simulations or quit the simulation.

```
quit -sim
```

(Main MENU: Simulate > End Simulation)

Lesson 8 - Simulating with the Performance Analyzer

The goals for this lesson are:

- Run a simulation with Performance Analyzer turned on
- View the Hierarchical and Ranked Profile displays
- Use the Performance Analyzer statistics displayed in the Hierarchical Profile and the Ranked Profile to speed up simulation

Performance Analyzer identifies the percentage of simulation time spent in each section of your code. With this information, you can identify bottlenecks and reduce simulation time by optimizing your code. Users have reported up to 75% reductions in simulation time after using Performance Analyzer.

This lesson introduces the Performance Analyzer and shows you how to use the main Performance Analyzer commands.

Note: You must be using ModelSim SE to complete this lesson.

Compiling and loading the design

This lesson will use an example design that contains lower-level VHDL blocks in the files *control.vhd*, *retrieve.vhd*, and *store.vhd*; and top-level block, test bench and configuration files – *ringrtl.vhd*, *testring.vhd*, and *config_rtl.vhd*.

1. Start by creating a new working directory, making it the current directory, and copying the files from *\modeltech\examples\profiler* into it.

2. Use the **vlib** command to create a **work** library in the current directory.

   ```
   vlib work
   ```
   (MENU: File > New > Library)

3. Use the **vmap** command to map the work library to a physical directory. A *modelsim.ini* file will be written into the current directory.

   ```
   vmap work work
   ```

4. Compile the lower level blocks of the design.

   ```
   vcom control.vhd retrieve.vhd store.vhd
   ```
   (MENU: Compile > Compile) 

5. Compile the top-level block, test bench and configuration files.

   ```
   vcom ringrtl.vhd testring.vhd config_rtl.vhd
   ```
   (MENU: Compile > Compile) 

6. Use the **vsim** command to load the design configuration.

   ```
   vsim work.test_bench_rtl
   ```
   (MENU: Simulate > Simulate) 

Running the simulation

1 Turn on profiling prior to running the simulation.

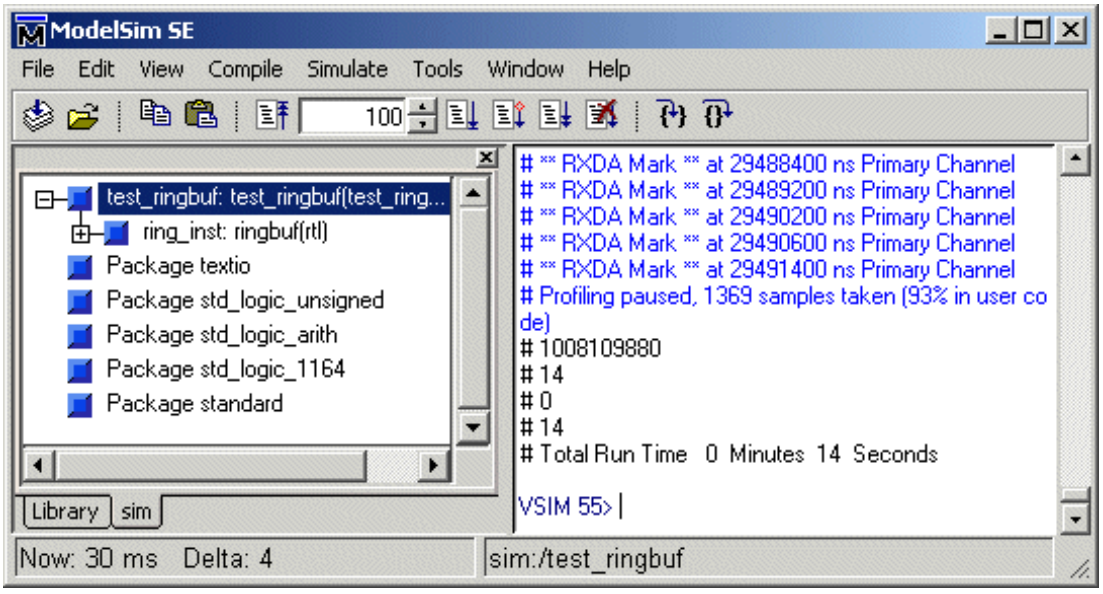
```
profile on
```

(MENU: Tools > Profile > Profile On)

2 We're going to run the simulation using a DO file that reports how long the simulation takes to run. Take a look at the commands in the *timerun.do* file. The *seconds* Tcl command is used to time the simulation.

```
do timerun.do
```

Notice as the simulation runs that the status bar shows how many profile samples are being taken.



Make a note of the run time reported in the Transcript window. We'll use it later to compare how much we've increased simulation speed. (Your times may differ from those shown here due to differing system configurations.)

3 Display the Hierarchical Profile output.

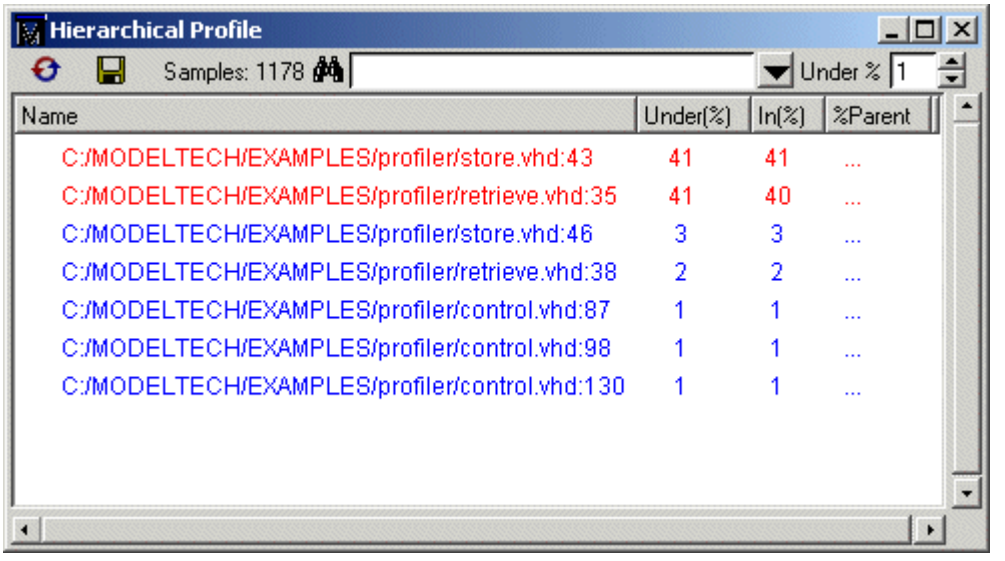
```
view_profile
```

(MENU: Tools > Profile > View hierarchical profile)

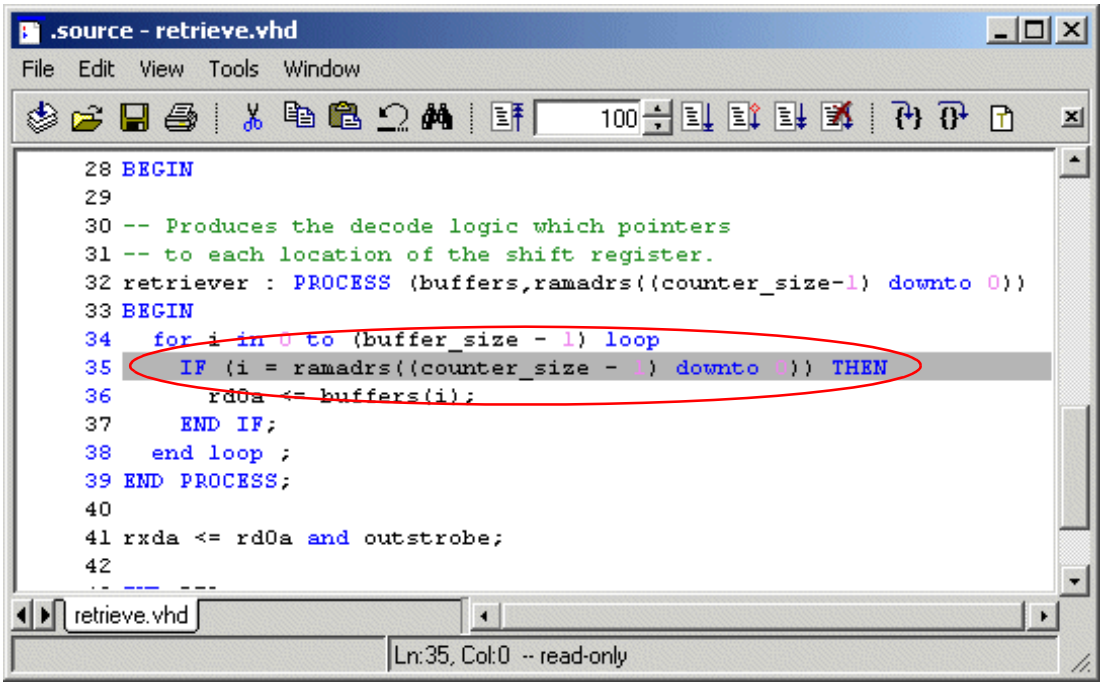
Note that two lines – *retrieve.vhd:35* and *store.vhd:43* – are taking the majority of the simulation time.

You can use the *$PrefProfile(hierCutoff)* Tcl control variable to filter out everything below a certain percentage. **hierCutoff** is the minimum percent usage that will be listed in the Hierarchical Profile display. The default value is 1%. Any usage less than 1% will not be displayed.

You can also filter the display with the Under % filter in the Hierarchical Profile window.



Double-clicking on any line in the Hierarchical Profile window will open the Source window and allow you to view the relevant source code for that line. The selected line will be highlighted in the Source window as shown below. (Here, we've double-clicked *retrive.vhd:35*.)



Speeding up the simulation

The information provided by the Performance Analyzer can be used to speed up the simulation. Double click the pathname for *store.vhd:43* and *retrieve.vhd:35* and view the source code. In both cases, the source includes a loop which could have an exit.

1. Modify the loops to include exits inside the *IF* statements, or compile the following files included for that purpose – *store_exit.vhd* and *retrieve_exit.vhd*.

   ```
   vcom retrieve_exit.vhd store_exit.vhd
   ```

   (MENU: Compile > Compile) 

2. Compile the top level blocks and configuration files again to account for the lower level changes.

   ```
   vcom ringrtl.vhd testring.vhd config_rtl.vhd
   ```

   (MENU: Compile > Compile) 

3. Reset the simulation to time zero and restart with the modified files.

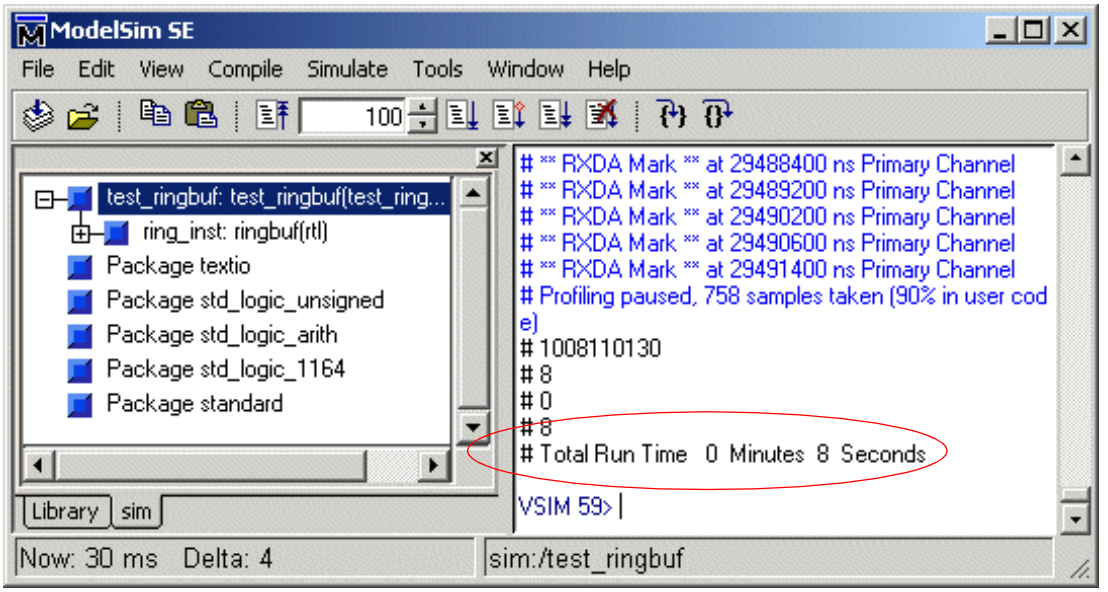
   ```
   restart -f
   ```

   (MENU: Simulate > Run > Restart) 

4 Run *timerun.do* again and note the difference in run time.

```
do timerun.do
```

Run time has been cut almost in half by inserting exits in the loops.

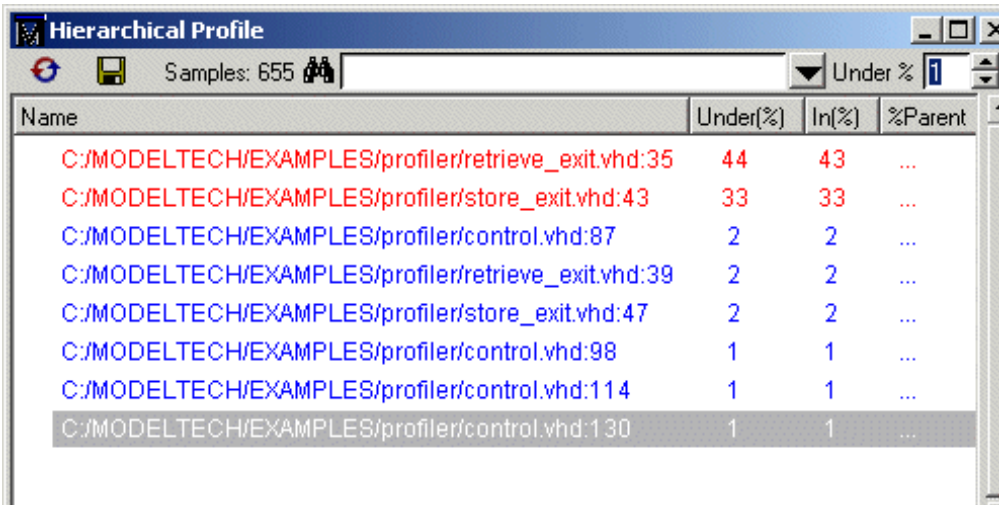


5 Take another look at the Performance Analyzer data.

```
view_profile
```

(MENU: Tools > Profile > View hierarchical profile)

A lot of time is still being spent in the loops. To further reduce simulation time, these loops can be replaced by indexing an array.



6 Remove the loops and add an array, or compile the following files which already contain the modifications.

```
vcom retrieve_array.vhd store_array.vhd
```

(MENU: Compile > Compile) 

7 Compile the top-level blocks and configuration files again.

```
vcom ringrtl.vhd testring.vhd config_rtl.vhd
```

(MENU: Compile > Compile) 

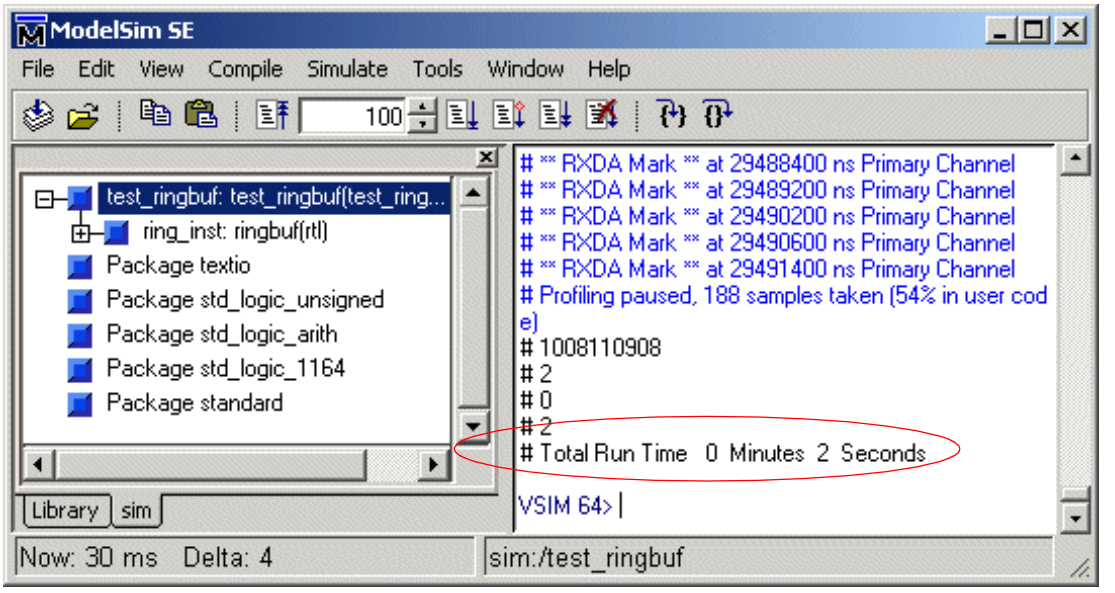
8 Restart the simulation with the modified files.

```
restart -f
```

(MENU: Simulate > Run > Restart) 

9 Run *timerun.do* again and note the difference in simulation run time. Your time may differ from that shown here, but the new run should be very fast – roughly ten times faster than the original simulation time.

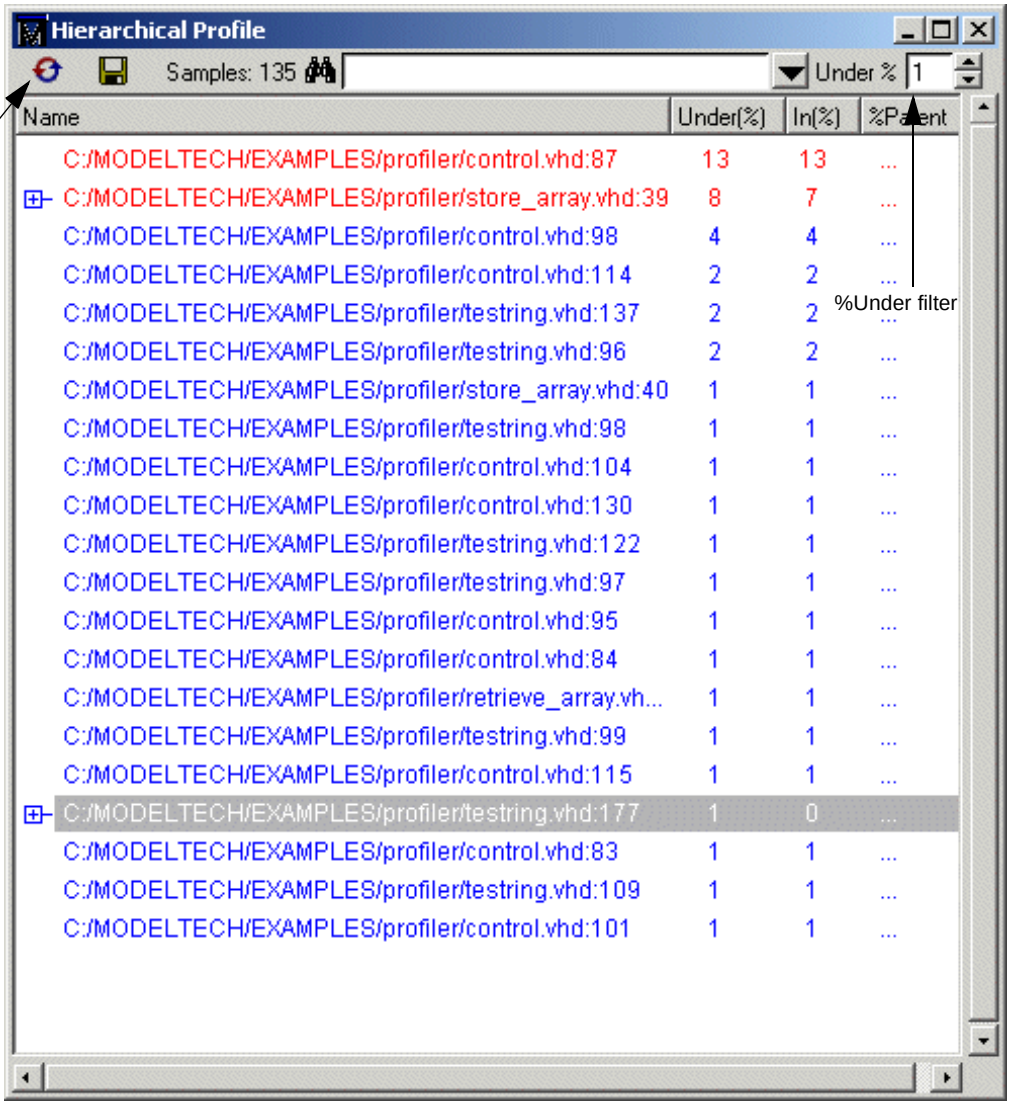
```
do timerun.do
```



10 Look again at the Hierarchical Profile and you will see more lines showing.

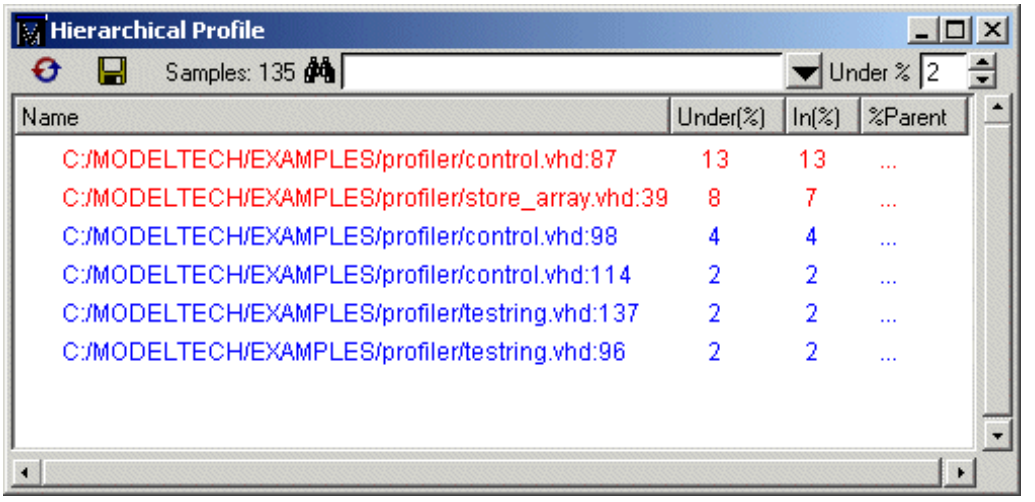
```
view_profile
```

(MENU: Tools > Profile > View hierarchical profile)



Note: You results may look slightly different as a result of the computer you're using and different system calls that occur during the simulation.

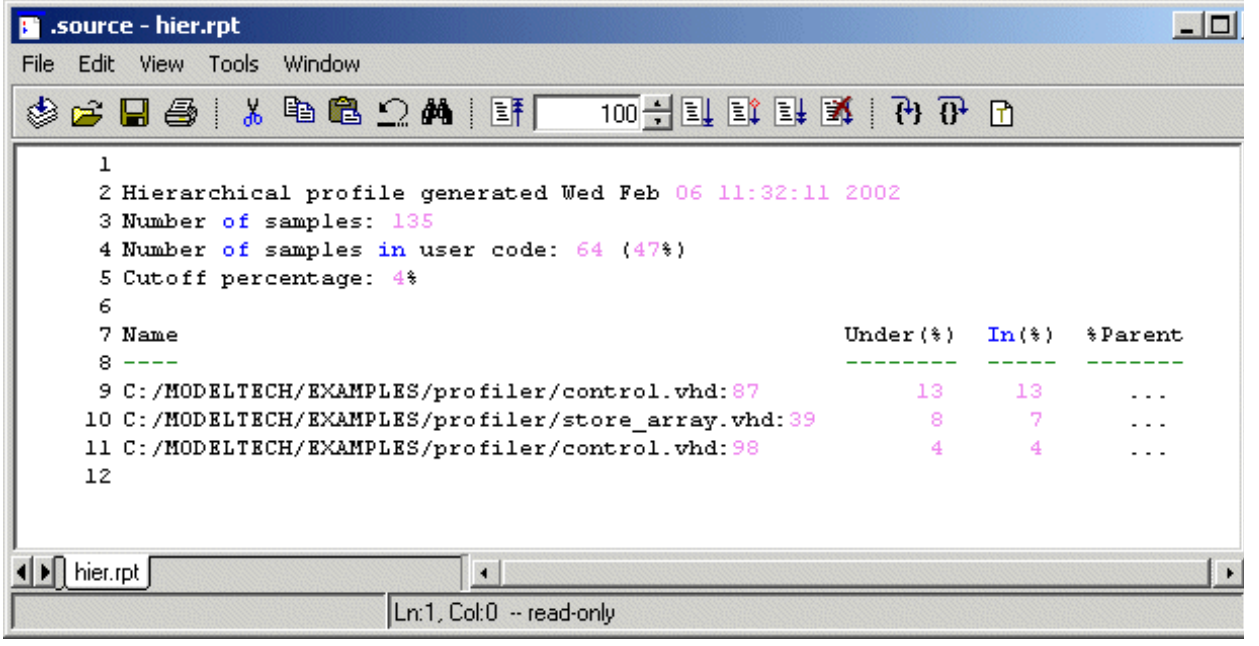
11 Set the Under% filter to "2" and click the Update icon. This will filter out all usage values below 2%.



12 Use the report command to output a file with the profile data.

```
profile report -hierarchical -file hier.rpt -cutoff 4
```

This command outputs a hierarchical profile of performance data with the file name *hier.rpt*.



13 Quit the simulation.

```
quit -f
```

Lesson 9 - Simulating with Code Coverage

The goals for this lesson are:

- Run a simulation with Code Coverage ON and examine the coverage_summary window
- Save line coverage information to a text file
- Exclude lines and files from the coverage statistics
- Append results from a previous simulation run onto the next one

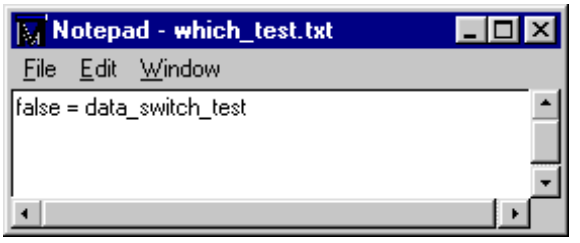
ModelSim Code Coverage allows you to identify which lines in your code are being covered by the testbench. You can merge the results of multiple tests, making it possible to use multiple testbenches or multiple stimulus files. It is non-intrusive (instrumented code is *not* required) and only minimally impacts simulation performance (<5%).

Running a simulation with Code Coverage

All commands are shown as entered on the ModelSim command line.

For this lesson, you'll use the same working directory used for the *Simulating with the Performance Analyzer* lesson. It is not necessary to recreate the work library if you completed the last lesson. See Lesson 8 - Simulating with the Performance Analyzer (T-65) if you need the details on these steps.

1. Prior to running the simulation, we need to check the *which_test.txt* file that was copied from the *modeltech\examples\profiler* directory to ensure it reads "false = data_switch_test". You can edit the file with **notepad** within ModelSim.



   This switch configures the test bench – the *testring.vhd* file. We'll change it later in the lesson when we merge the coverage results of two simulations.

2. Compile the lower-level blocks of the design.

```
vcom control.vhd retrieve_array.vhd store_array.vhd
```

   (MENU: Compile > Compile)

3. Compile the top-level block, test bench, and configuration files.

```
vcom ringrtl.vhd testring.vhd config_rtl.vhd
```

   (MENU: Compile > Compile) 

4. Use the **vsim -coverage** command to load the design configuration with Code Coverage.

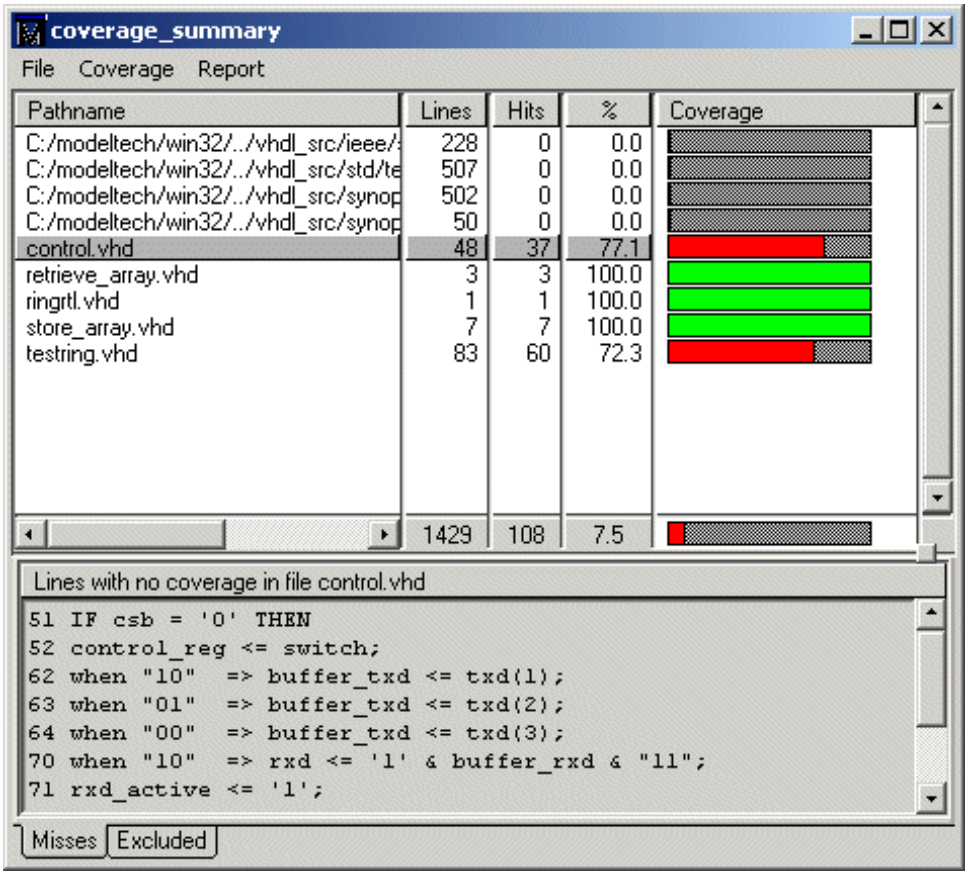
```
vsim -coverage work.test_bench_rtl
```

5. Run the simulator for 3 milliseconds.

```
run 3 ms
```

6 Display the coverage_summary window.

```
view_coverage
```
(MENU: Tools > Source Coverage)

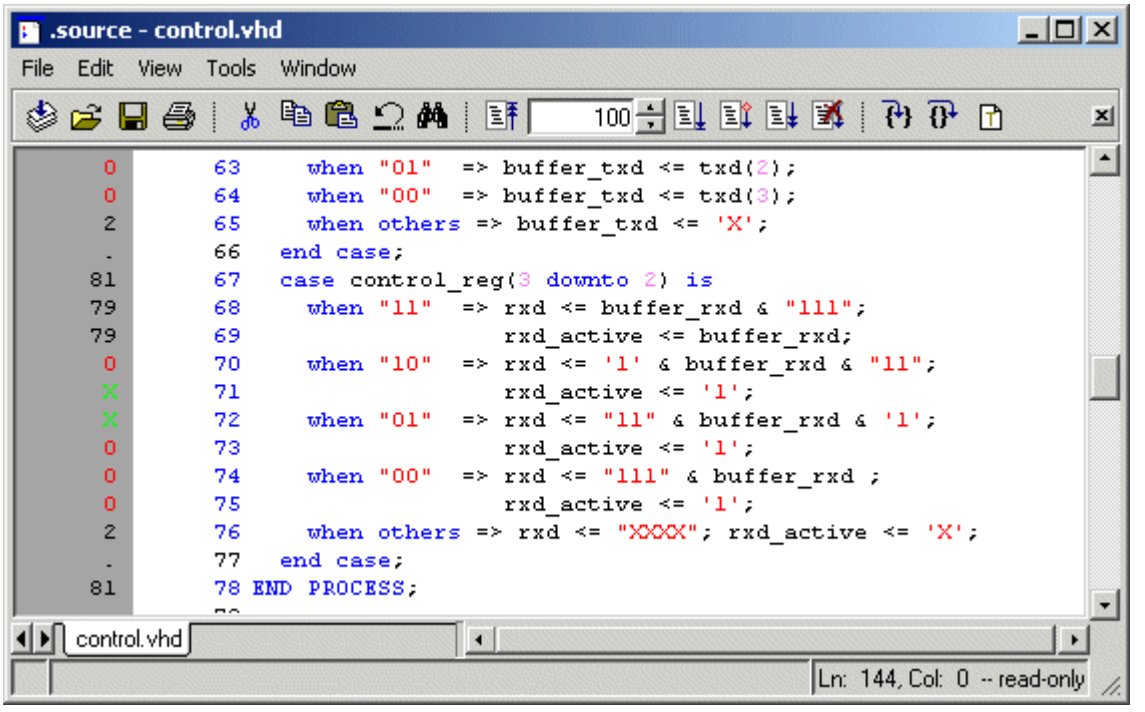


The top half of the window shows summary information on a per-file basis. If you select a file in the list, the bottom part of the window gives details about the lines in the file that have zero coverage.

Note that both *testring.vhd* and *control.vhd* are below 90% and, therefore, shown in red in the Coverage bar graph. 90% is the default coverage threshold, and all coverage values below 90% will be shown red. The default coverage threshold can be changed with the Tcl control variable *$PrefCoverage(cutoff)*.

7 Click on the *control.vhd* pathname to display the source code for *control.vhd* in the Source window. With Code Coverage enabled, the Source window is displayed with an extra column that details the number of times each line has been executed. The green "Xs" in the graphic below denote lines that have been excluded. We'll discuss that later in the lesson.

Scroll the Source window to view the executable lines. As you can see some lines have red zeros next to them. This indicates a line that was not executed.



8 Save the line coverage information to a text file.

```
coverage report -file cover.dat -lines
```
(coverage_summary MENU: Report > Save Line Coverage)

Open the file *cover.dat* to see how the data is stored. **Notepad** works well to check text files such as this.

```
notepad cover.dat
```

Excluding lines and files

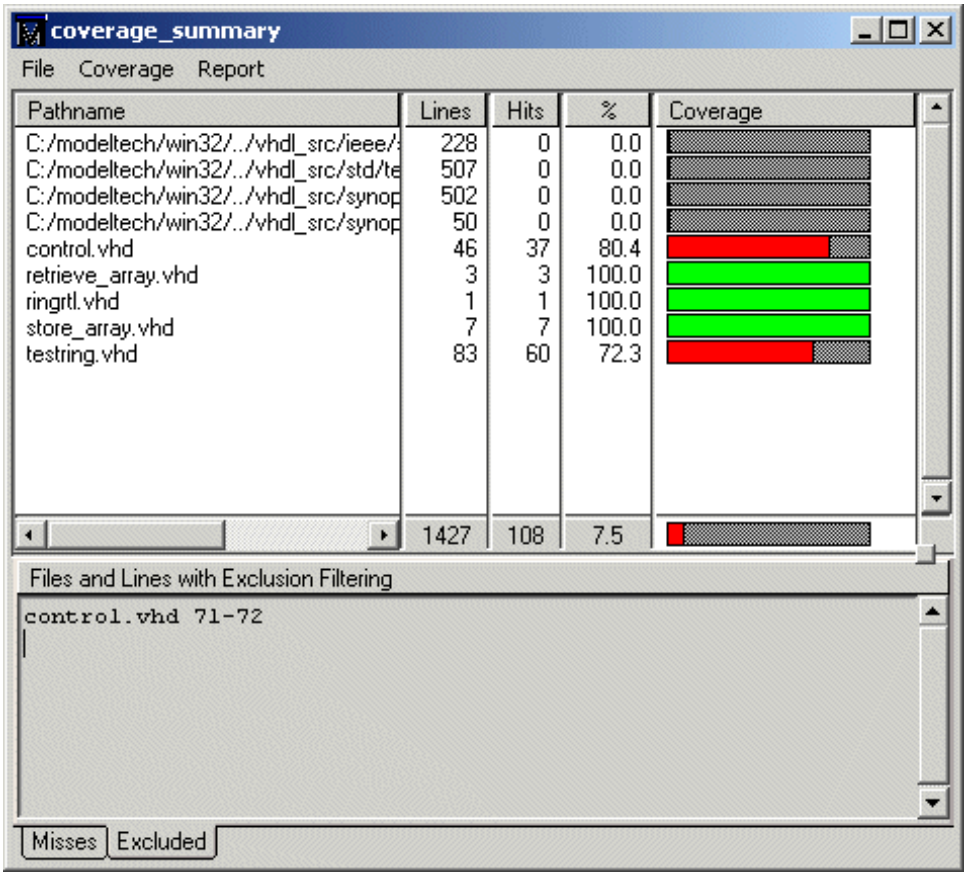
There may be a time when you want to exclude certain parts of your code from the analysis. You can exclude both lines and files from either the source or the coverage_summary windows.

1. Scroll to one of the executable lines in the Source window (noted with a blue line number).

2. With your mouse pointer over the column that shows coverage results (the left-most column), click your right-mouse button and select **Exclude Coverage Line #**.

3. The line will now be marked with a green "X" and the coverage statistics will be updated to exclude that line.

4. View the coverage_summary window and click the **Excluded** tab. This tab shows all the lines and files that are currently being excluded.



5 Select the line in the Excluded tab, click your right mouse button, and select **Include Entire Selected Files**. This removes the exclusion filter on any lines from the selected file.

You can continue experimenting with the various exclusion commands at this point if you want. However, before continuing with the tutorial, select **Coverage > Clear Out Current Filter** to ensure all lines and files are included for the next exercise.

Merging coverage results from two simulations

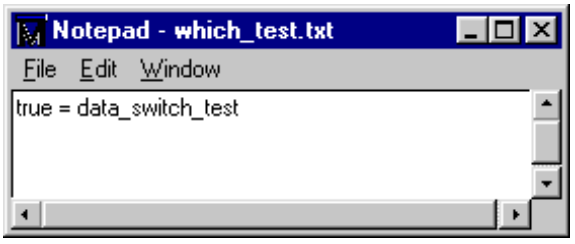
You can merge code coverage results from multiple simulations, making it possible to run multiple tests on a design and assess coverage across all tests. In this exercise, we'll change the test that is run by the testbench, resimulate, and then append the coverage statistics from our previous analysis to the new analysis.

1. Note how many times the clocked processes have been executed.

2. Next we'll edit the *which_test.txt* file. Changing this text file causes a different test to be run from the same testbench.

   Using ModelSim Notepad, edit the file so it reads "true = data_switch_test." Make sure the **Edit** > **read_only** switch is not on.

   ```
   notepad which_test.txt
   ```

3. Restart the simulation so the different test is run on the circuit.

   ```
   restart -f
   ```

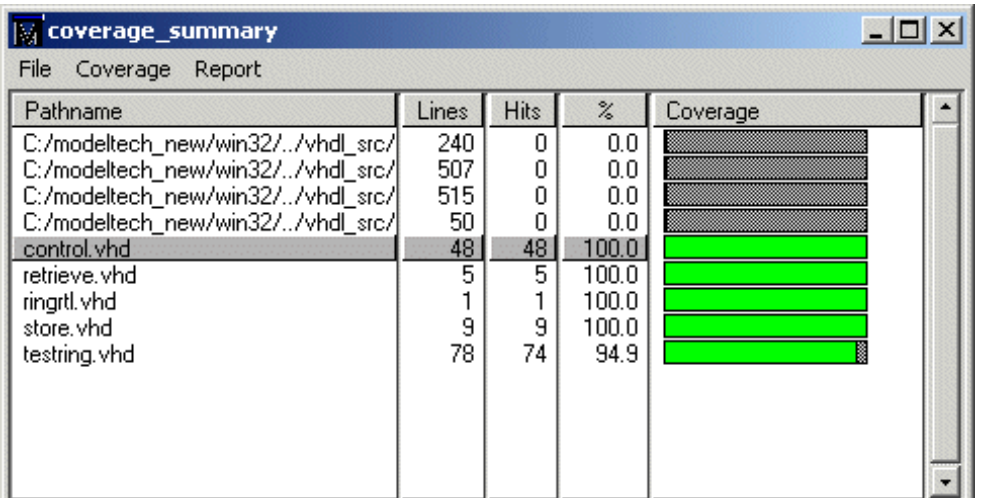
4. Restore the coverage data from the last simulation run so that its data can be appended to the current simulation.

   ```
   coverage reload cover.dat
   ```
   (coverage_summary MENU: File > Open > Coverage > Merge Coverage)

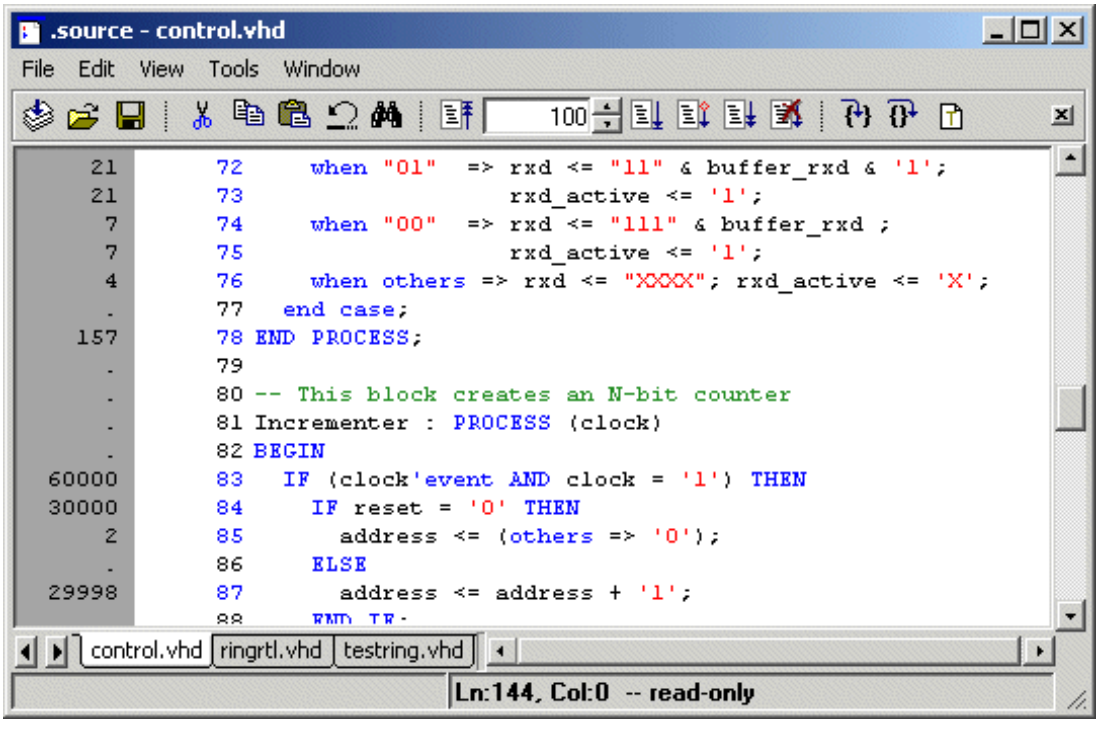
5. Run the simulator for 3 milliseconds as before.

   ```
   run 3 ms
   ```

   Note that now both *testring.vhd* and *control.vhd* are above 90% and therefore shown in green.



6 Click on the *control.vhd* pathname to bring up the Source window. You can see from the values in the first column that the line hits from this run have been added to the ones from the last run. The number of times the clocked processes have been run has doubled.



7 Quit the simulator.

```
quit -f
```

Lesson 10 - Comparing waveforms

The goals for this lesson are:

- Compare two simulations using the Comparison Wizard
- View comparison results and timing difference markers in the Wave window
- Use compare icons to jump to "previous" and "next" difference markers
- View comparison results in the List window
- Set an edge tolerance

Waveform Comparison computes timing differences between test signals and reference signals. In this exercise we're going to run and save the mixedHDL simulation, edit one of the source files, run the simulation again, and finally compare the two runs.

The general procedure for comparing waveforms has four main steps:

1 Selecting the simulations or datasets to compare

2 Specifying the signals or regions to compare

3 Running the comparison

4 Viewing the comparison results

Creating the reference dataset

We'll start by running a simulation and saving it to a dataset. This dataset will become the reference dataset when we set up the comparison.

1. Start by creating a new directory for this exercise. Create the directory and copy all of the files from *\<install_dir>\modeltech\examples\mixedHDL* to the new directory.

   Make sure the new directory is the current directory. Do this by invoking ModelSim from the new directory or by selecting the **File > Change Directory** command from the ModelSim Main window.

2. At the ModelSim prompt in the Transcript pane, run the compare.do DO file.

   ```
   do compare.do
   ```

   This DO file does the following:

   - Creates and maps the work library
   - Compiles the Verilog and VHDL files
   - Runs the simulation and saves the results to a dataset named "gold.wlf"

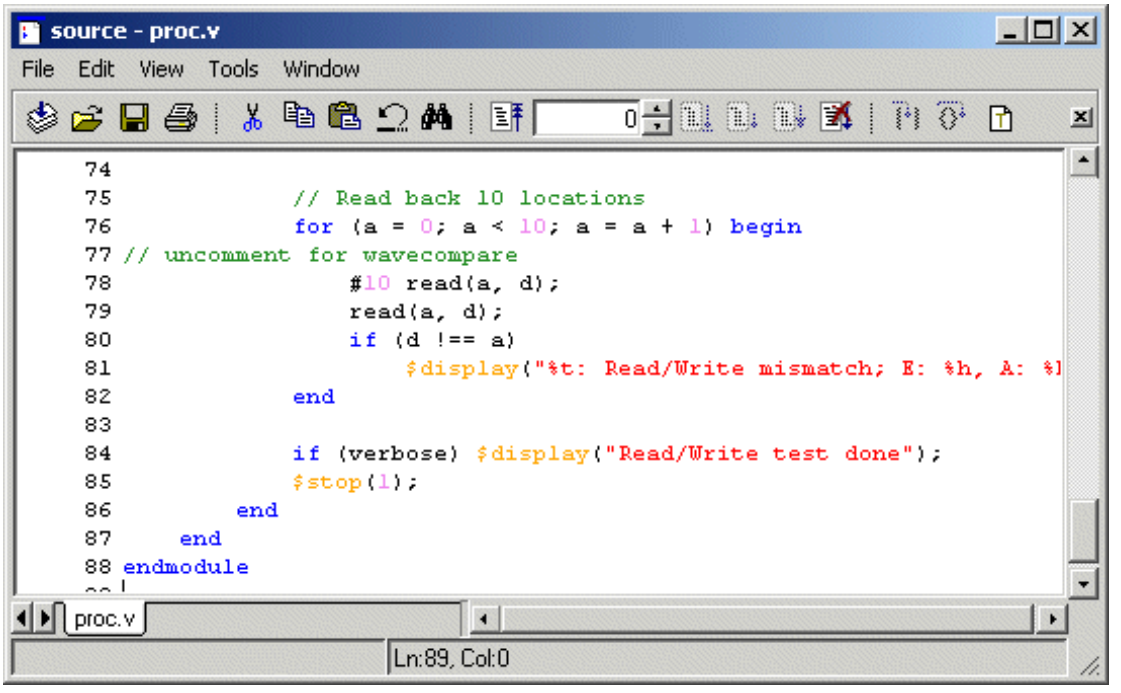
   Feel free to open the DO file and take a look at its contents.

Editing a source file and re-running the simulation

In the last step, we ran the default mixed HDL simulation and saved it to the *gold.wlf* dataset. Now we'll edit one of the source files and re-run the simulation.

1. Edit the *proc.v* file by opening it in the Source window. Make sure the **Edit > read only** flag isn't selected.

2. Scroll down and un-comment the read cycle on line 78. Your source file should look like the following:



3. Save the file in the Source window.

   (MENU: File > Save) 

4. Re-compile the *proc.v* file.

   (PROMPT: vlog proc.v)

   (Main MENU: Compile > Compile) 

5 Load the top design unit.

(PROMPT: vsim work.top)

(MENU: Simulate > Simulate) 

6 Add the waves to the Wave window and run the simulation.

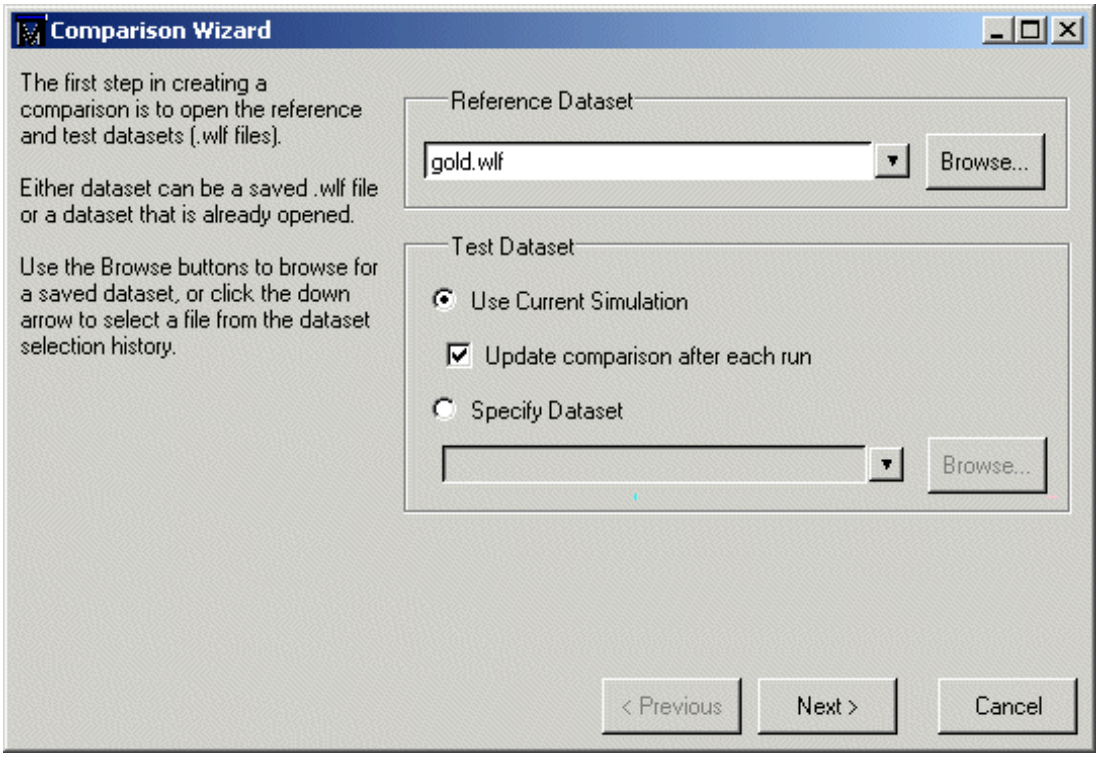
```
add wave *
run -all
```

Comparing the simulation runs

ModelSim includes a Comparison Wizard that walks you through the steps of setting up a waveform comparison. You can also do it manually with menu or command line commands.

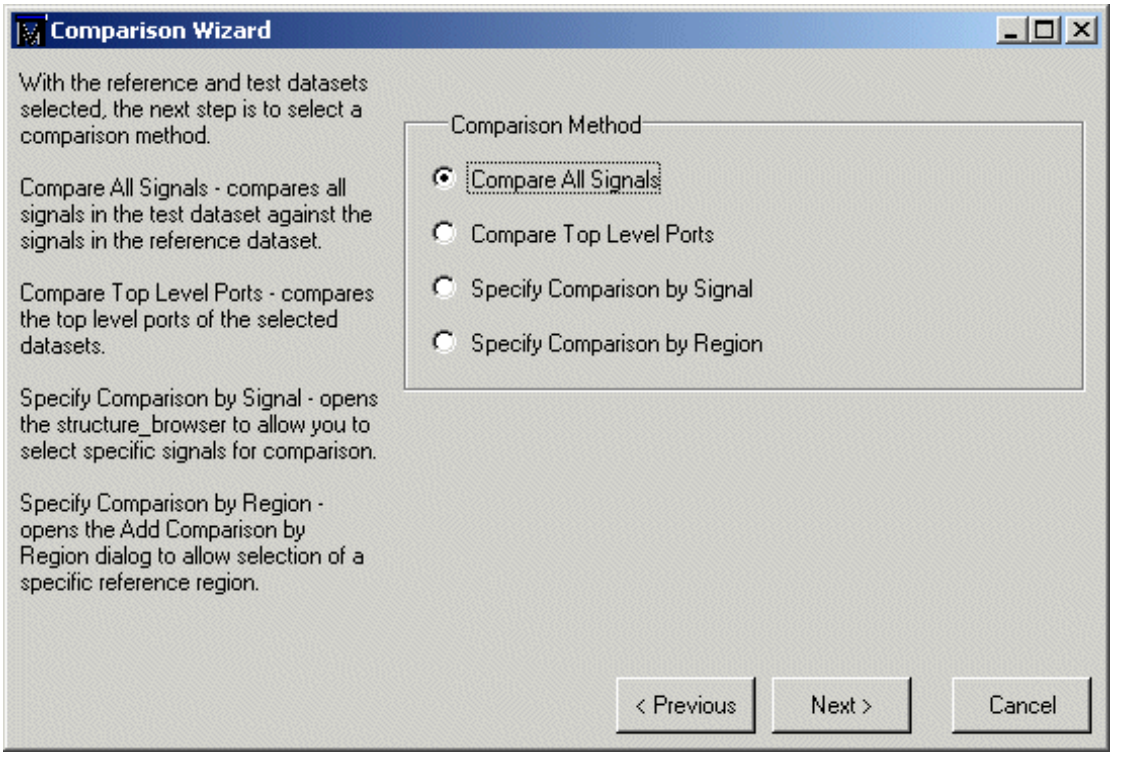
1. Select **Tools > Waveform Compare > Comparison Wizard** from the Wave or Main window.

2. Click the browse button and select *gold.wlf* as the Reference Dataset. Recall that this dataset is from the first simulation run prior to adding the 10 time unit delay.



Leave the Test Dataset set to **Use Current Simulation**, and then click Next.

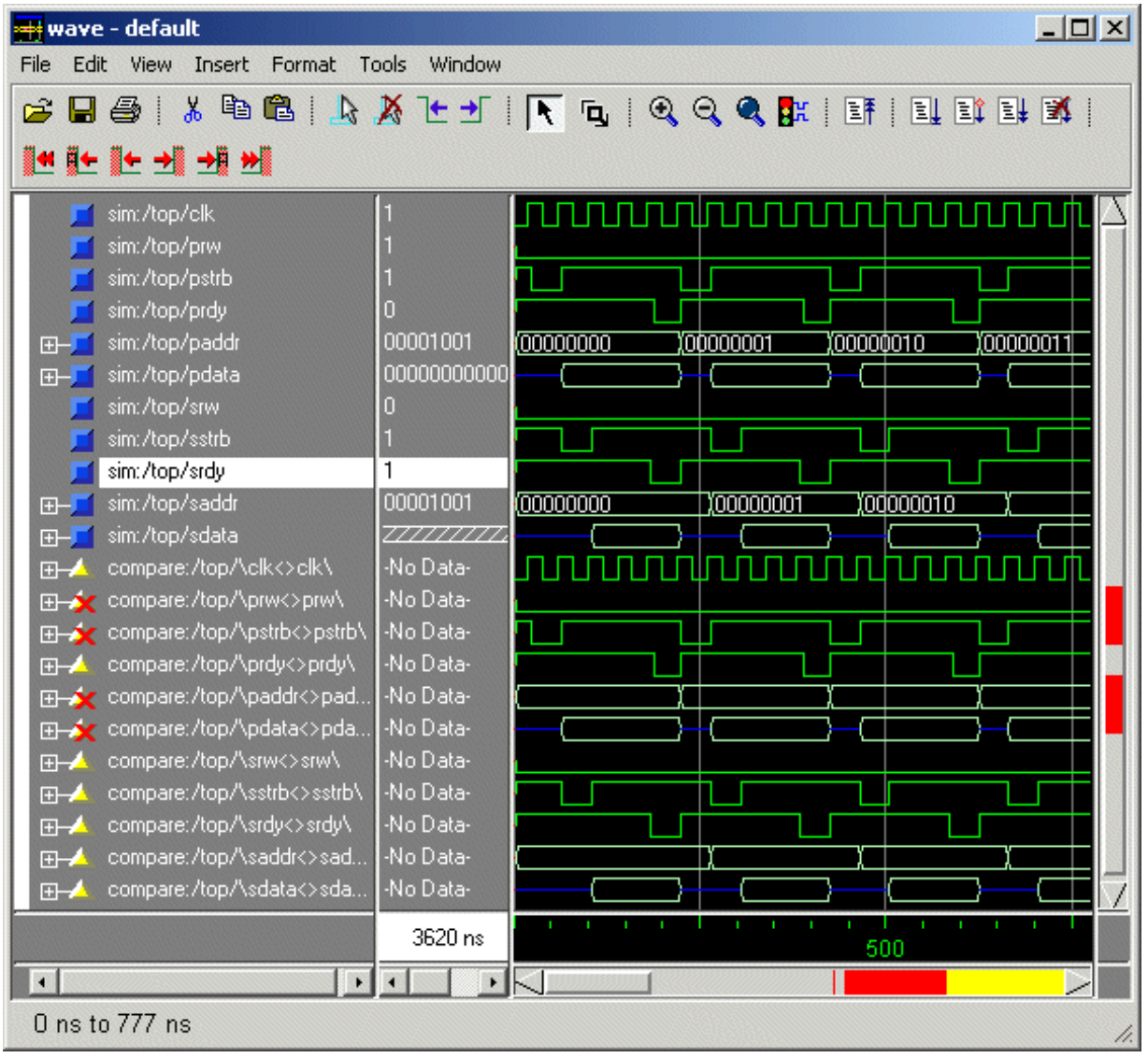
3 Select **Compare All Signals** in the second dialog, and then click Next.



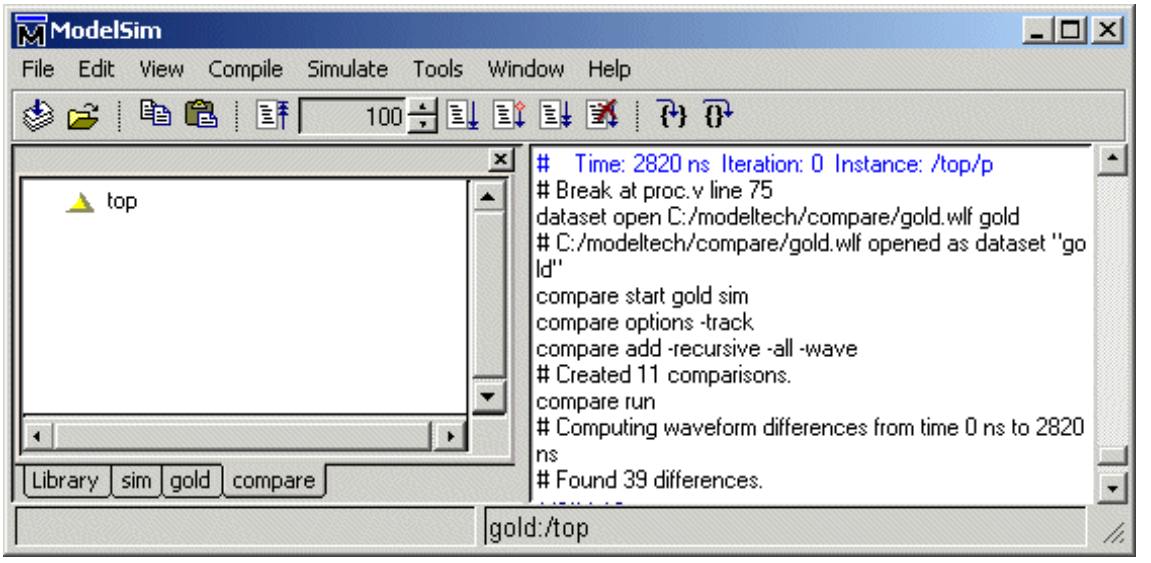
4 In the next three dialogs, click Next, Compute Differences Now, and Finish, respectively.

Viewing and saving the comparison data

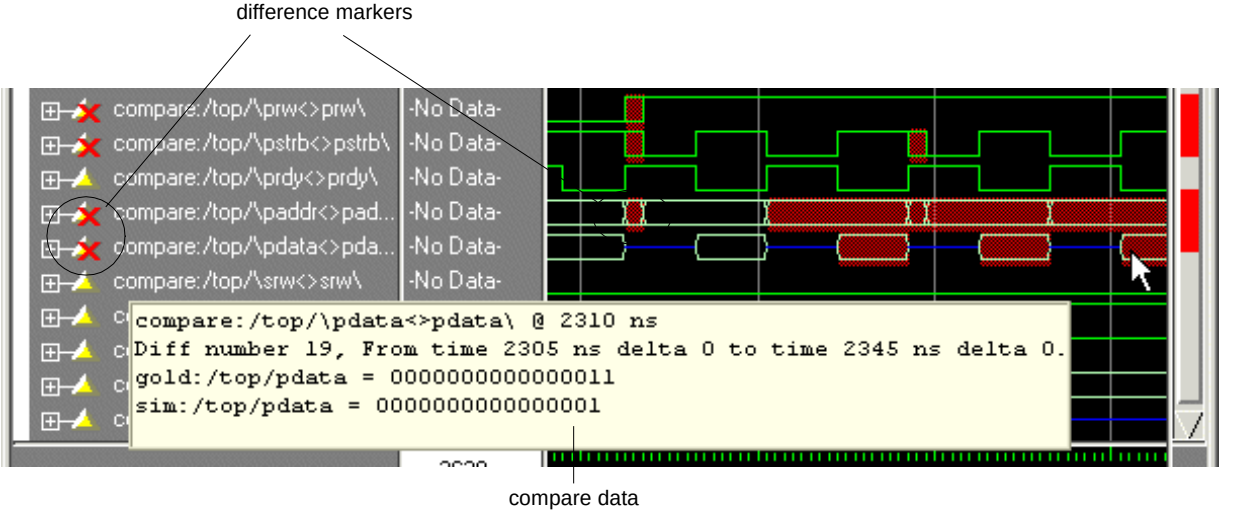
ModelSim performs the comparison and displays the compared signals in the Wave window.



The Compare tab in the Main window shows the region that was compared, and the transcript area shows the number of differences found between the timing of the Reference and Test datasets.



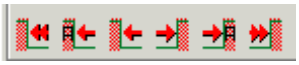
In the Wave window, a signal that contains timing differences between the two simulations is denoted by a red X over its yellow triangle. Red difference markers in the waveform display area show the location of the timing differences on the waveforms, as do the red lines in the horizontal scrollbar at the bottom of the window.



Hover your mouse pointer over a difference marker to display a popup containing data about that timing difference. Also note that when you place a waveform cursor over a difference, the values column displays the text "diff."

Compare icons

The Wave window includes six waveform comparison icons that let you quickly jump between differences. From left to right, the icons do the following: find first difference, find previous annotated difference, find previous difference, find next difference, find next annotated difference, find last difference. Use these icons to move the selected cursor.



The next and previous buttons cycle through differences on all signals. To view differences for just the selected signal, use <tab> and <shift> - <tab>.

Saving the comparison

You can save the comparison for later viewing, either in a text file or in files that can be reloaded into ModelSim.

To save the difference information to a text file, select **Tools > Waveform Compare > Differences > Write Report**.

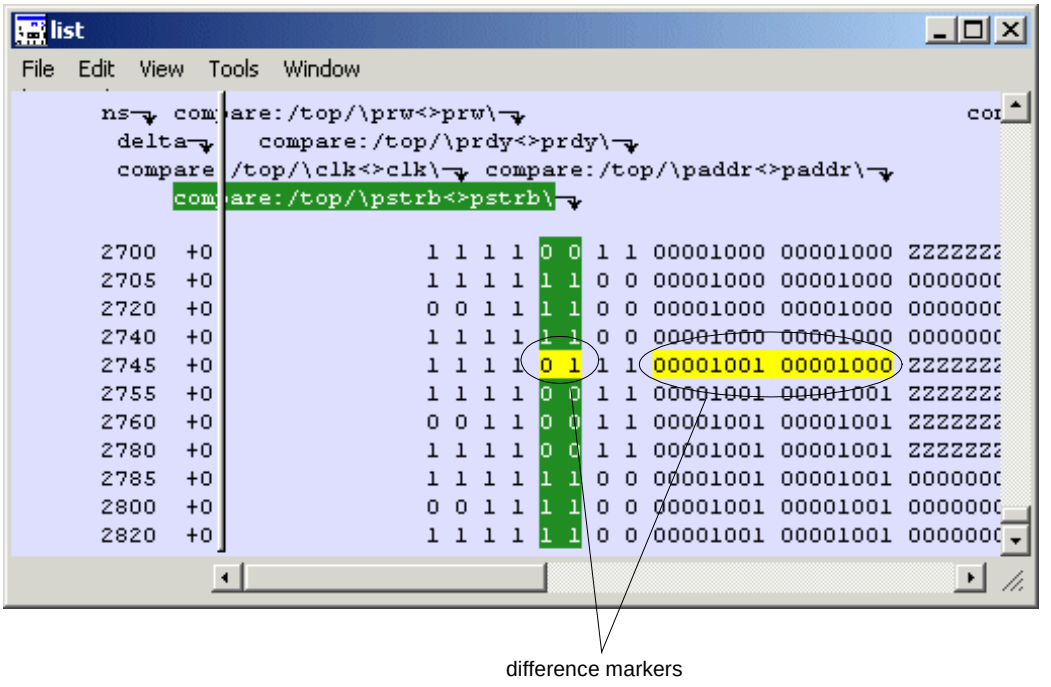
To save the comparison so it can be reloaded into ModelSim, you must save two files. Select **Tools > Waveform Compare > Differences > Save** to save the computed differences. Next, select **Tools > Waveform Compare > Rules > Save** to save the comparison configuration rules. To reload the comparison later, you would start a comparison and then use the **Tools > Waveform Compare > Reload** command.

Viewing comparison results in the List window

You can also view the results of your waveform comparison in the List window.

1. Select **View** > **List** to open the List window.

2. Drag the region from the Compare tab in the Main window to the List window. This will load the compared signals into the List window. Scroll down the window, and you'll see differences shown in yellow.



Specifying tolerances

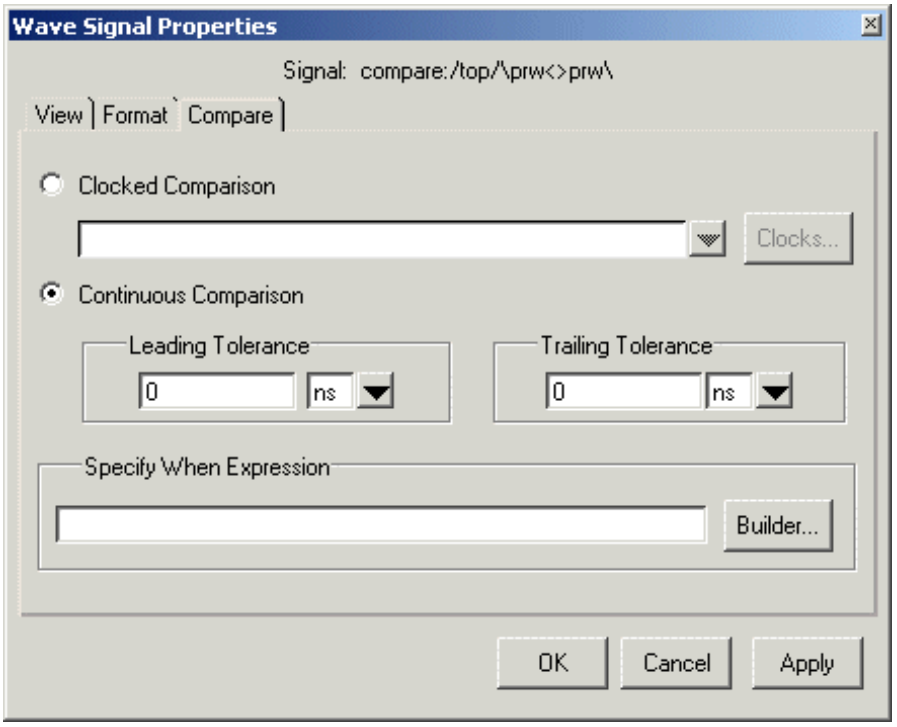
There may be times you want to allow for leading or trailing tolerances in the test dataset signals. You can do this easily by modifying the signal properties of a comparison object in the Wave window.

1 Click the Find Next Difference icon until you can see the differences at 2025 ns.

(KEYBOARD: Tab) 

2 Select "compare:/top/\prw<>prw\" in the signals list and then right-click to open the Signal Properties dialog. Select the Compare tab.

(MENU: View > Signal Properties)



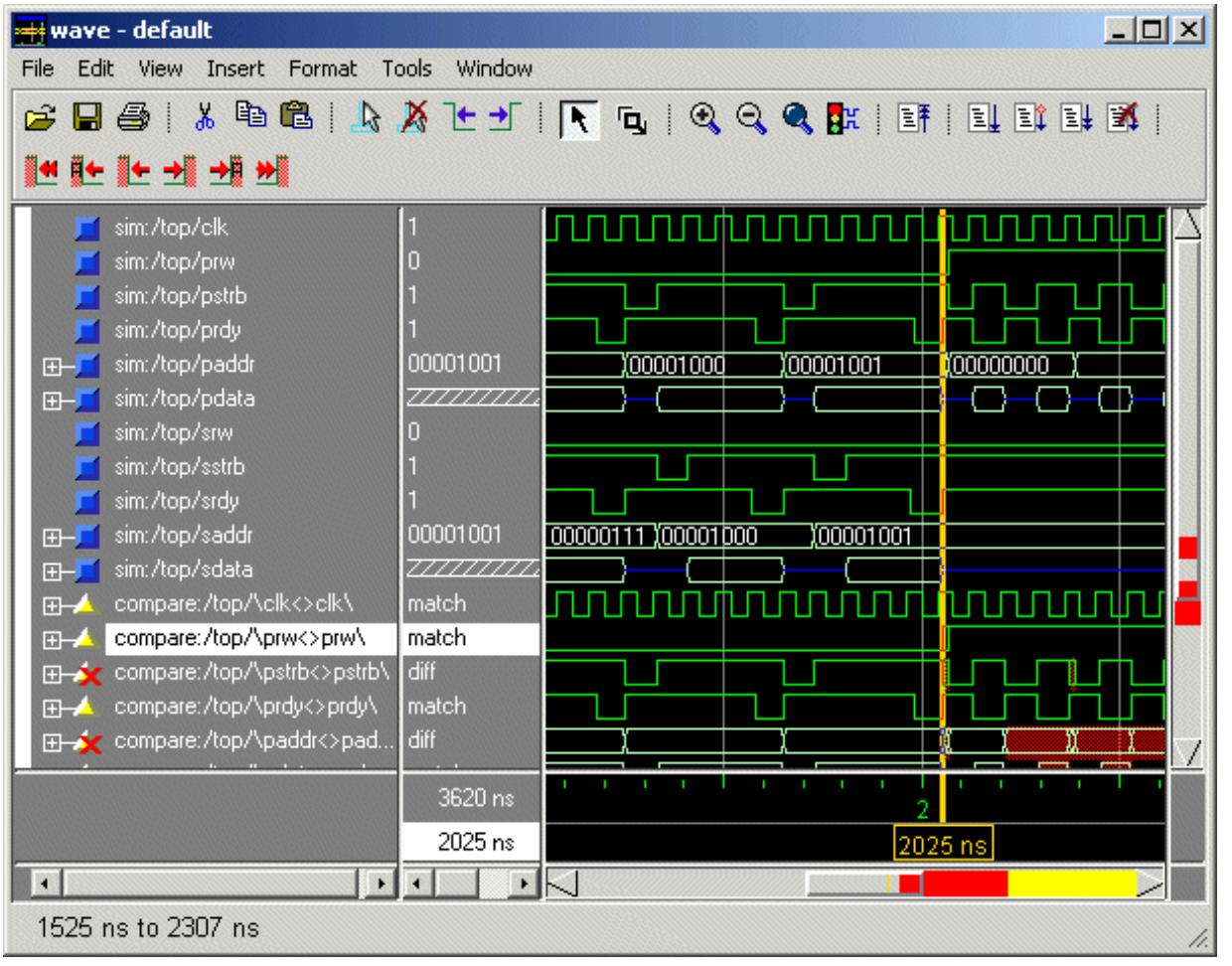
Recall that we delayed the read cycle in *proc.v* by 10 time units. Therefore, if we specify a trailing tolerance of 10 ns, the differences on the comparison object should disappear.

3 Specify 10 ns for the Trailing Tolerance and then click OK.

4 Rerun the comparison.

(MENU: Tools > Waveform Compare > Run Comparison)

5 Notice that the difference markers have disappeared for the /top/prw comparison object.



6 Quit the simulator.

```
quit -f
```

Lesson 11 - Debugging with the Dataflow window

The goals for this lesson are:

- Log signals so you have information necessary for debugging
- Explore the connectivity of your design
- Trace an event
- Trace an X (unknown) value
- Jump to the source of an unknown
- View hierarchy in the Dataflow window
- Zoom and pan the Dataflow window

The Dataflow window allows you trace VHDL signals or Verilog nets and registers through your design. ModelSim versions 5.6 and later include enhanced Dataflow functionality that expands your debugging options.

Compiling and loading the design

We'll start by compiling and loading a mixed design that we'll use for subsequent examples.

1. Create a new working directory, make it the current directory, and then copy the files from *\modeltech\examples\mixedHDL* into it.

2. Use the **vlib** command to create a **work** library in the current directory.

   ```
   vlib work
   ```
   (MENU: File > New > Library)

3. Use the **vmap** command to map the work library to a physical directory. A *modelsim.ini* file will be written into the **work** directory.

   ```
   vmap work work
   ```

4. Compile the Verilog files.

   ```
   vlog cache.v memory.v proc.v
   ```
   (MENU: Compile > Compile) 

5. Compile the VHDL files.

   ```
   vcom util.vhd set.vhd top.vhd
   ```
   (MENU: Compile > Compile) 

6. Load the top level of the design.

   ```
   vsim top
   ```
   (MENU: Simulate > Simulate) 

7. Log all signals in the design so we have all information for debugging.

   ```
   log -r /*
   ```

8. Run the design for 500 ns.

   ```
   run 500 ns
   ```

Exploring connectivity

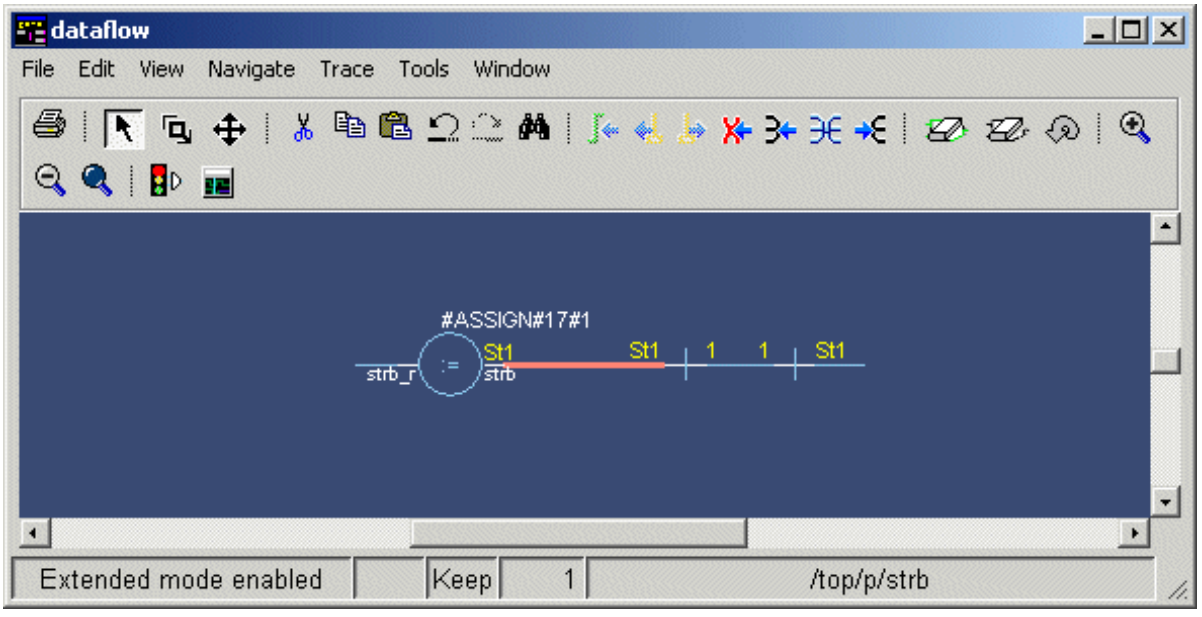
A primary use of the Dataflow window is exploring the "physical" connectivity of your design. You do this by expanding the view from process to process. This allows you to see the drivers/receivers of a particular signal, net, or register.

1. Select *p: proc* in the sim tab of the Main window.

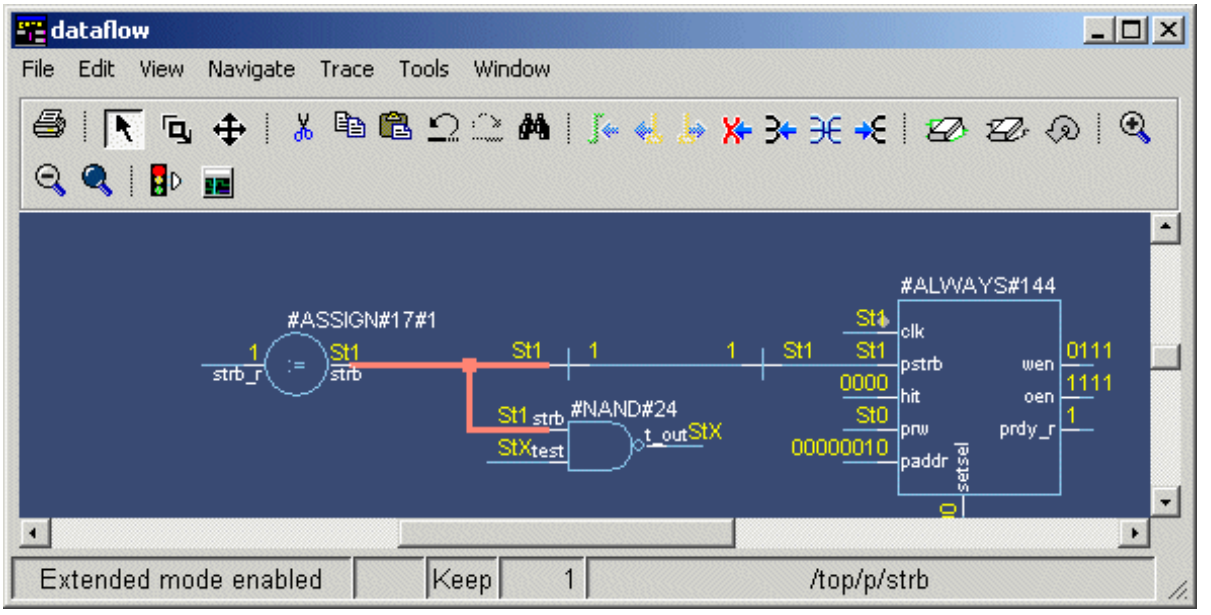
2. Open the Signals and Dataflow windows.

   ```
   view si d
   ```
   (MENU: View > Signals, View > Dataflow)

3. Drag signal *strb* from the Signals window to the Dataflow window.



4 Double click the net that is highlighted in red. The view expands to display the processes that are connected to *strb*.



5 Select signal *test* on process *#AND#24* and expand the view to show its drivers.

(MENU: Navigate > Expand net to drivers) 

Notice that after the display expands, the signal line for *strb* is highlighted in green. This highlighting lets you know the path you have traversed in the design.

6 Select signal *oen* on process *#ALWAYS#144*, and expand the view to show its readers.

(MENU: Navigate > Expand net to readers) 

7 Continue exploring if you wish. When you are done, clear the Dataflow window before moving on to the next exercise.

(MENU: Edit > Erase all) 

Tracing events

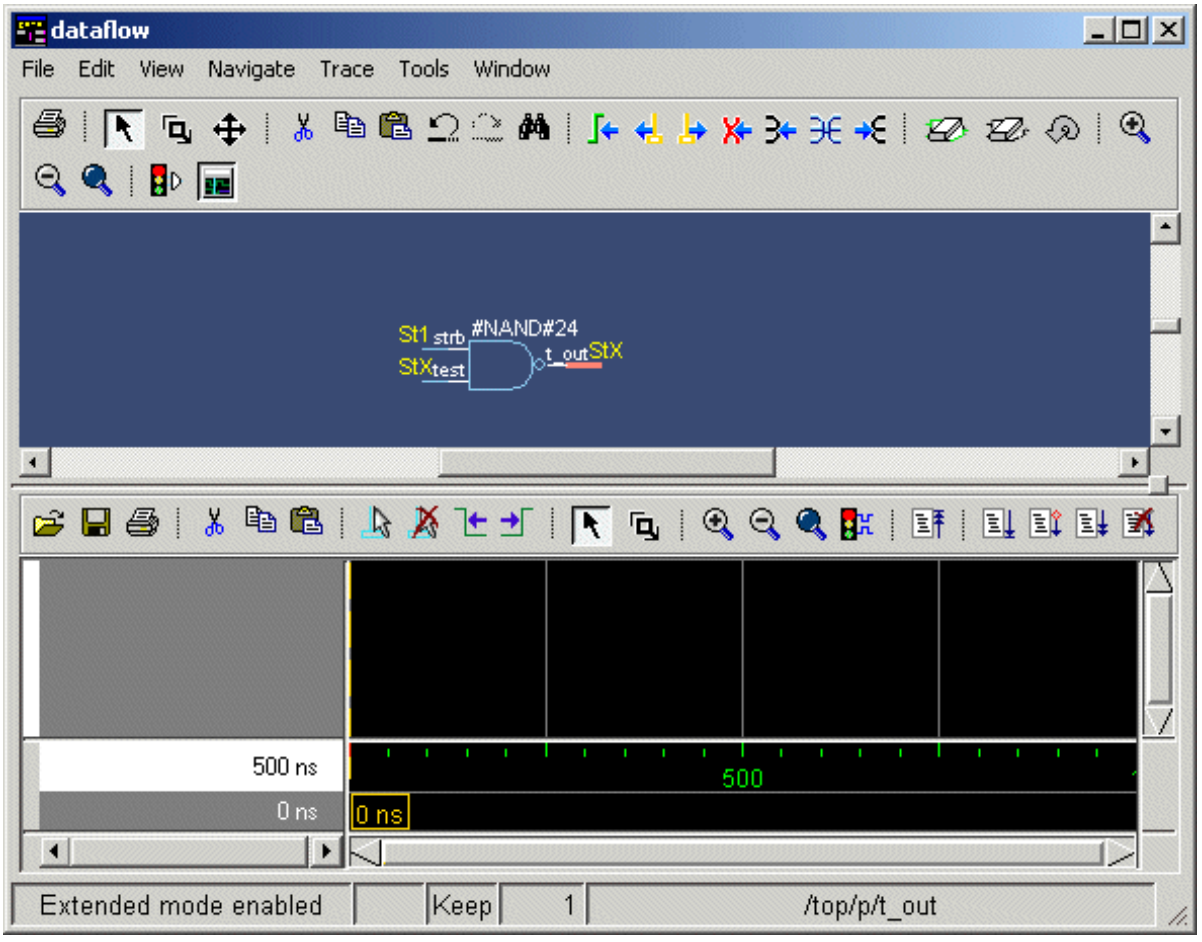
Another useful debugging feature is tracing events that contribute to an unexpected output value. Using the Dataflow window's embedded wave viewer, you can trace backward from a transition to see what process or signal is causing the unexpected output.

1 If you didn't do so in the last exercise, clear the Dataflow window.

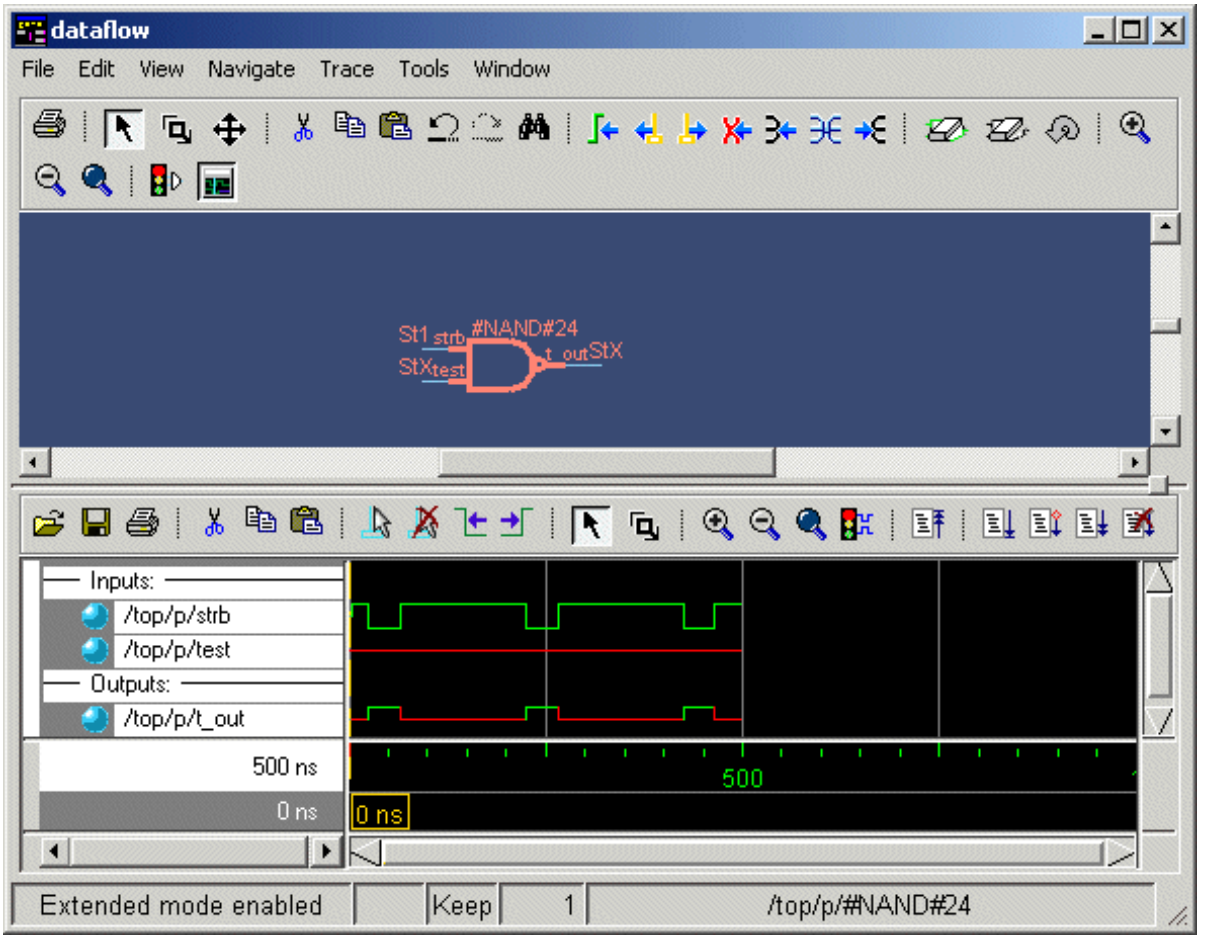
2 Select *p: proc* in the sim tab of the Main window, and then drag signal *t_out* from the Signals window into the Dataflow window.

3 Open the embedded wave viewer and increase the size of the window.

(MENU: View > Show Wave) 



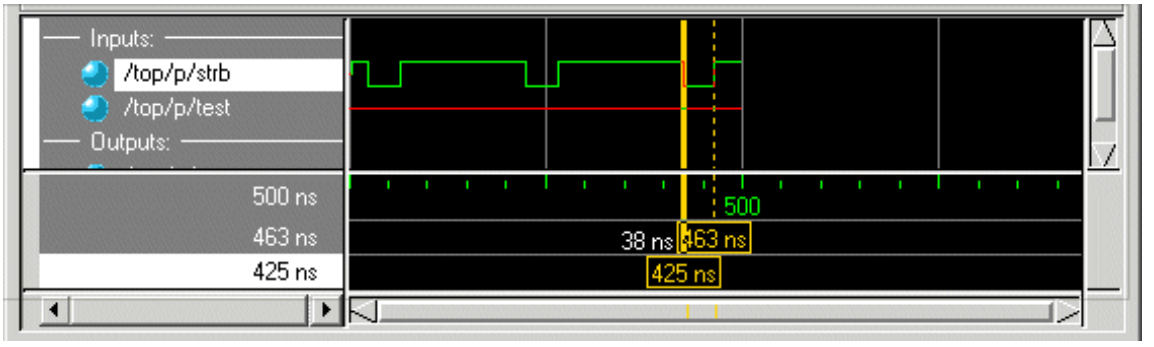
4 Select process *#NAND#24* in the dataflow pane. Notice that all input and output signals of the process are displayed automatically in the wave viewer.



5 Set a time cursor in the wave viewer at the last transition of signal *t_out* (465 ns). See "Making cursor measurements" (T-59) for more information on setting cursors.

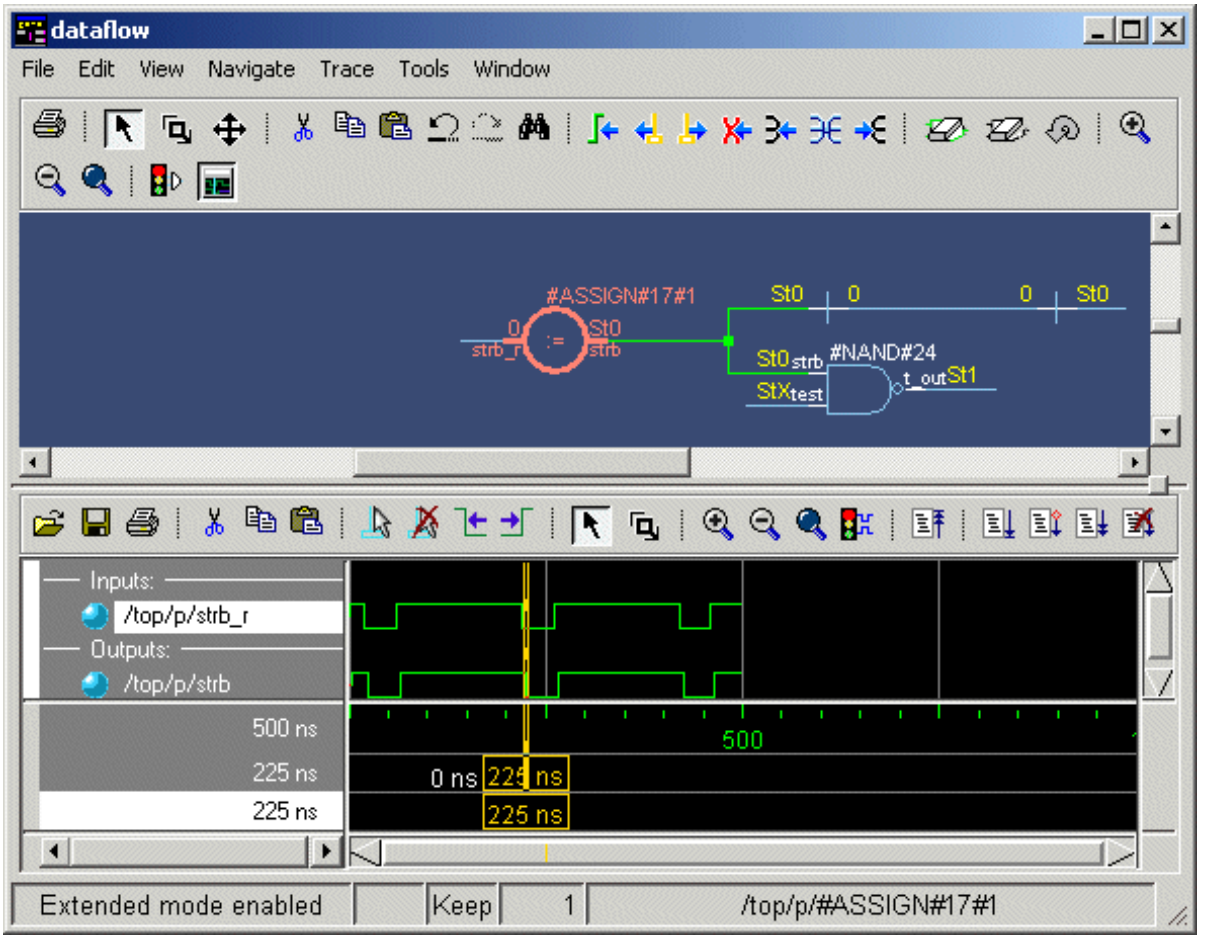
6 To trace to the first contributing event, select **Trace** > **Trace next event**.

A new cursor is added to the wave viewer marking the last event, the transition of the strobe to 0, which caused the output of 0 on *t_out*.



7 Trace the next event two more times and then select **Trace** > **Trace event set**.

The dataflow pane sprouts to the preceding process and shows the input driver of signal *strb*. Notice too that the wave viewer now shows the input and output signals of the newly selected process.



You can continue tracing events through the design in this manner: select **Trace next event** until you get to a transition of interest in the wave viewer, and then select **Trace event set** to update the dataflow pane.

8 Clear the Dataflow window before moving on to the next exercise. Also, close the wave viewer pane.

(MENU: View > Show Wave)

Tracing an 'X' (unknown)

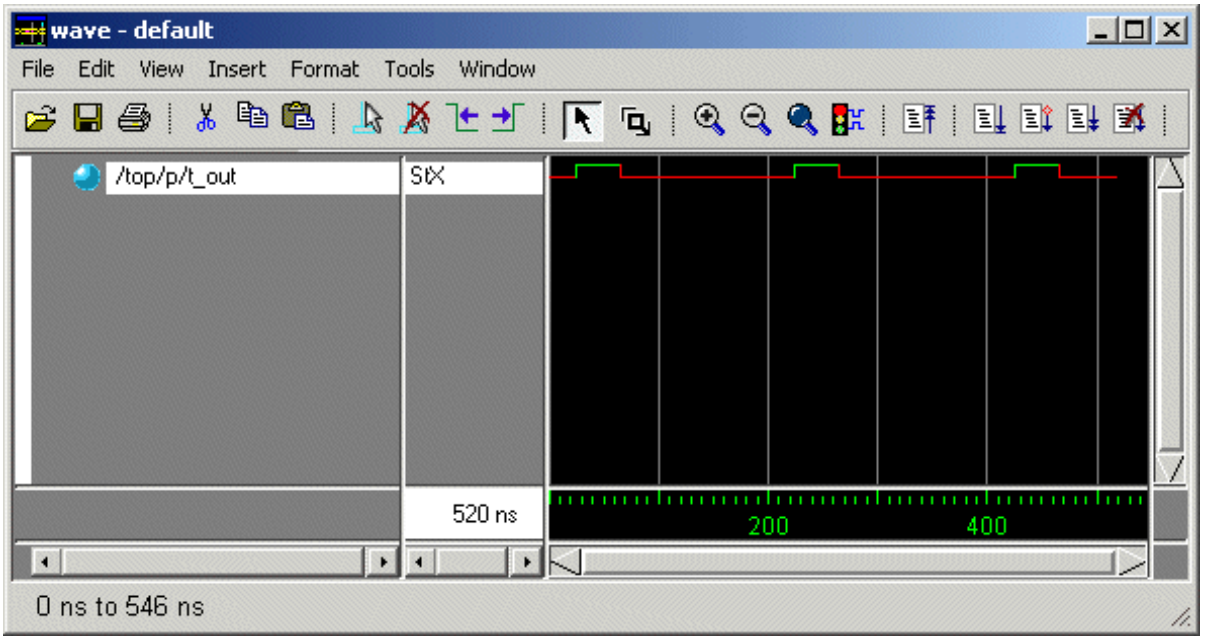
The Dataflow window lets you easily track an unknown value (X) as it propagates through the design. The Dataflow window is linked to the stand-alone Wave window, so you can view signals in the Wave window and then use the Dataflow window to track the source of a problem. As you traverse your design in the Dataflow window, appropriate signals will be added automatically to the Wave window.

1 Open the Wave window and add a signal.

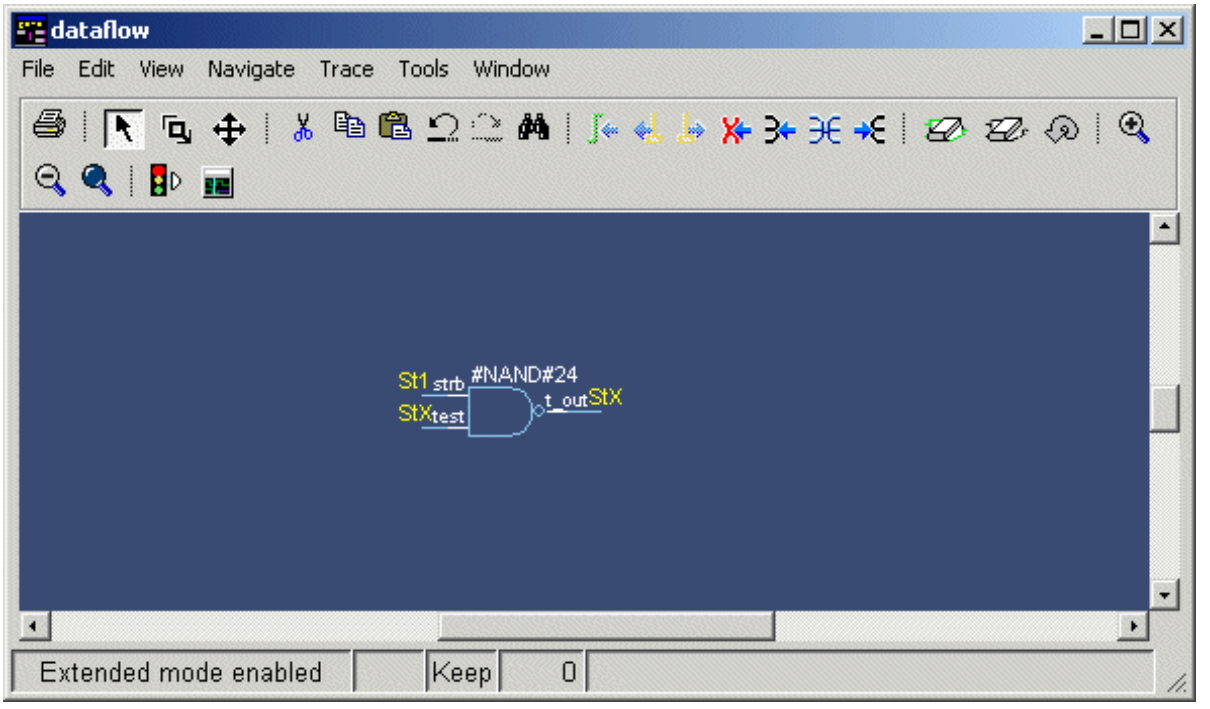
```
view wave
add wave /top/p/t_out
```

(MENU: View > Wave)
(GUI: Open Signals window and drag signal to Wave window)

Note that *t_out* goes to an unknown state (StX) at time 0 and continues transitioning to StX throughout the run. The red color of the waveform indicates an unknown value.



2 Drag *t_out* from the Wave window to the Dataflow window.



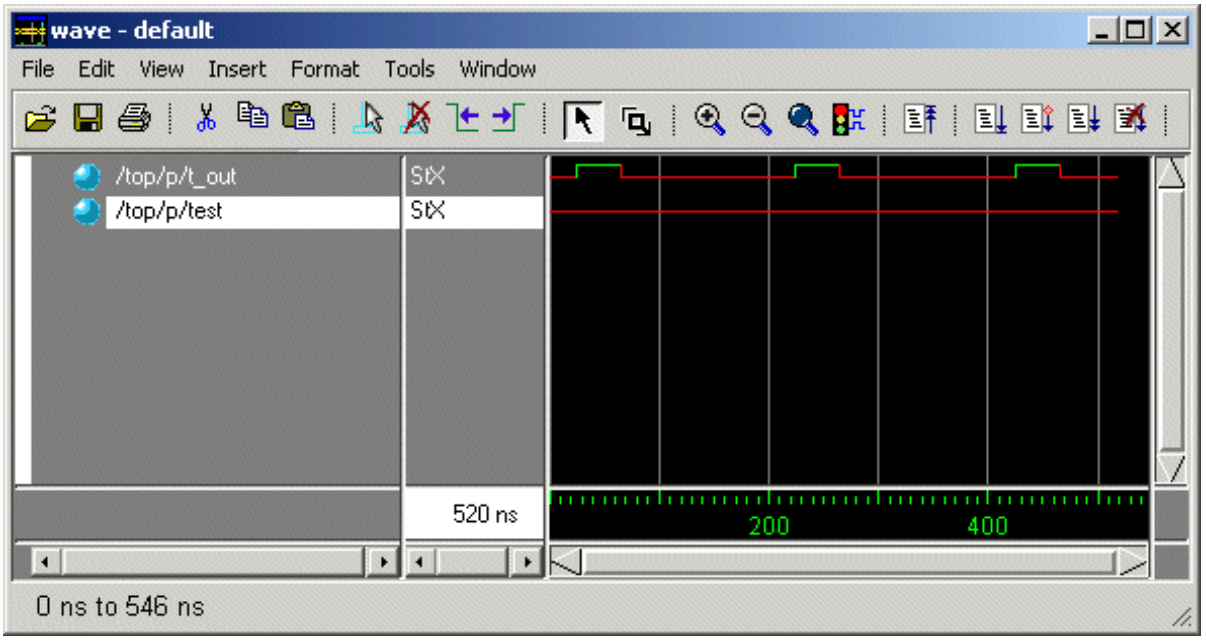
You must click somewhere in the Dataflow window to get the yellow signal values to appear.

As previously mentioned the Wave and Dataflow windows are designed to work together. Try moving the cursor in the Wave window (see "Making cursor measurements" (T-59) for details), and you'll see that the value of *t_out* changes in the Dataflow window. We'll look at other links between the windows as we work through the tutorial.

3 Move the Wave window cursor back to a time when *t_out* is unknown. Then, with *t_out* selected in the Dataflow window, trace the unknown.

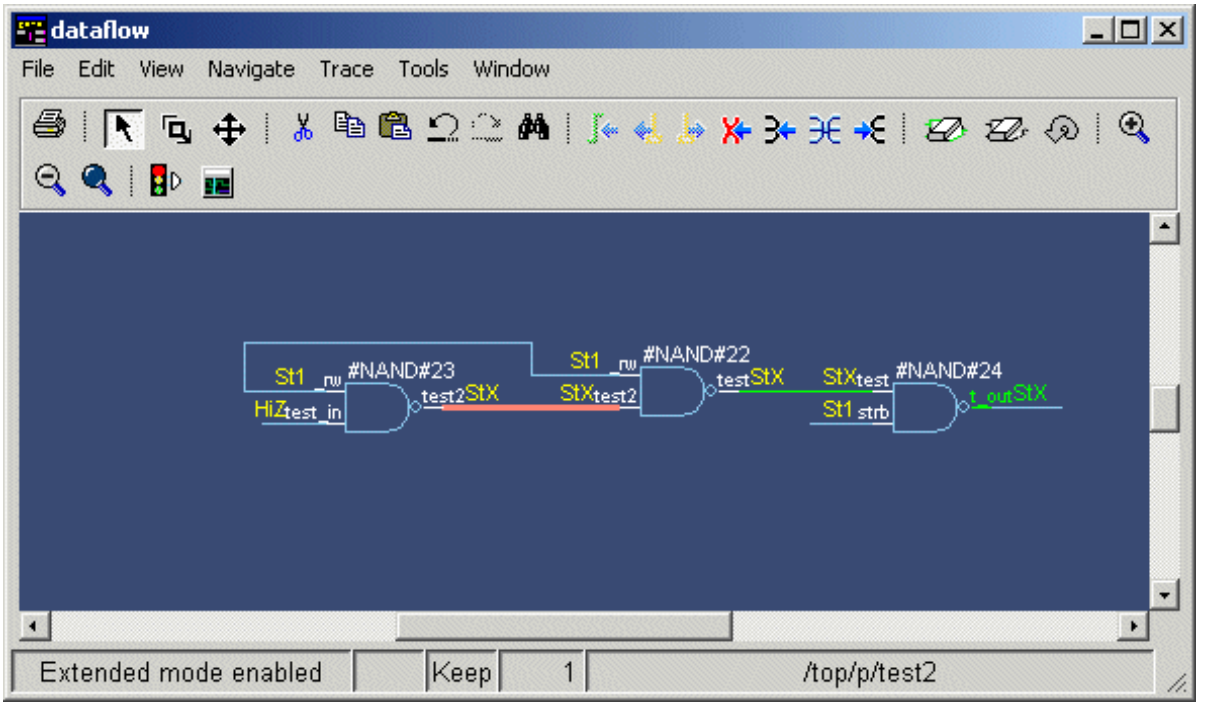
(MENU: Trace > TraceX) 

The input signal *test* is selected in the Dataflow window, and it is also added automatically to the Wave window.



4 Continue tracing back to the source of the unknown. Select **Trace** > **TraceX** again. This time signal *test2* is highlighted in the Dataflow window, and it is also added to the Wave window.

5 Select **Trace > TraceX** once more, and you'll discover the source of the unknown. In this case there is a HiZ on input signal *test_in* and a 1 on input signal *_rw*, so output signal *test2* resolves to an 'X'.



6 Clear the Dataflow window.

(MENU: Edit > Erase All) 

Jumping to the source of an X

In the last exercise you traced an unknown, from process to process, until you identified the source. You can speed this up by jumping directly to the source in one step.

1. Drag *t_out* from the Wave window to the Dataflow window as you did in the last exercise.
2. Select **Trace > ChaseX**.
3. The design expands to show the source of the unknown.
4. Clear the Dataflow window.

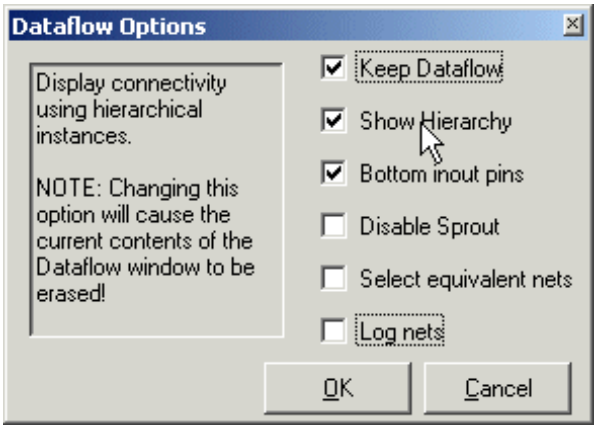
   (MENU: Edit > Erase All) 

Displaying hierarchy in the Dataflow window

You can display connectivity in the Dataflow window using hierarchical instances. You enable this by modifying the options prior to adding items to the window.

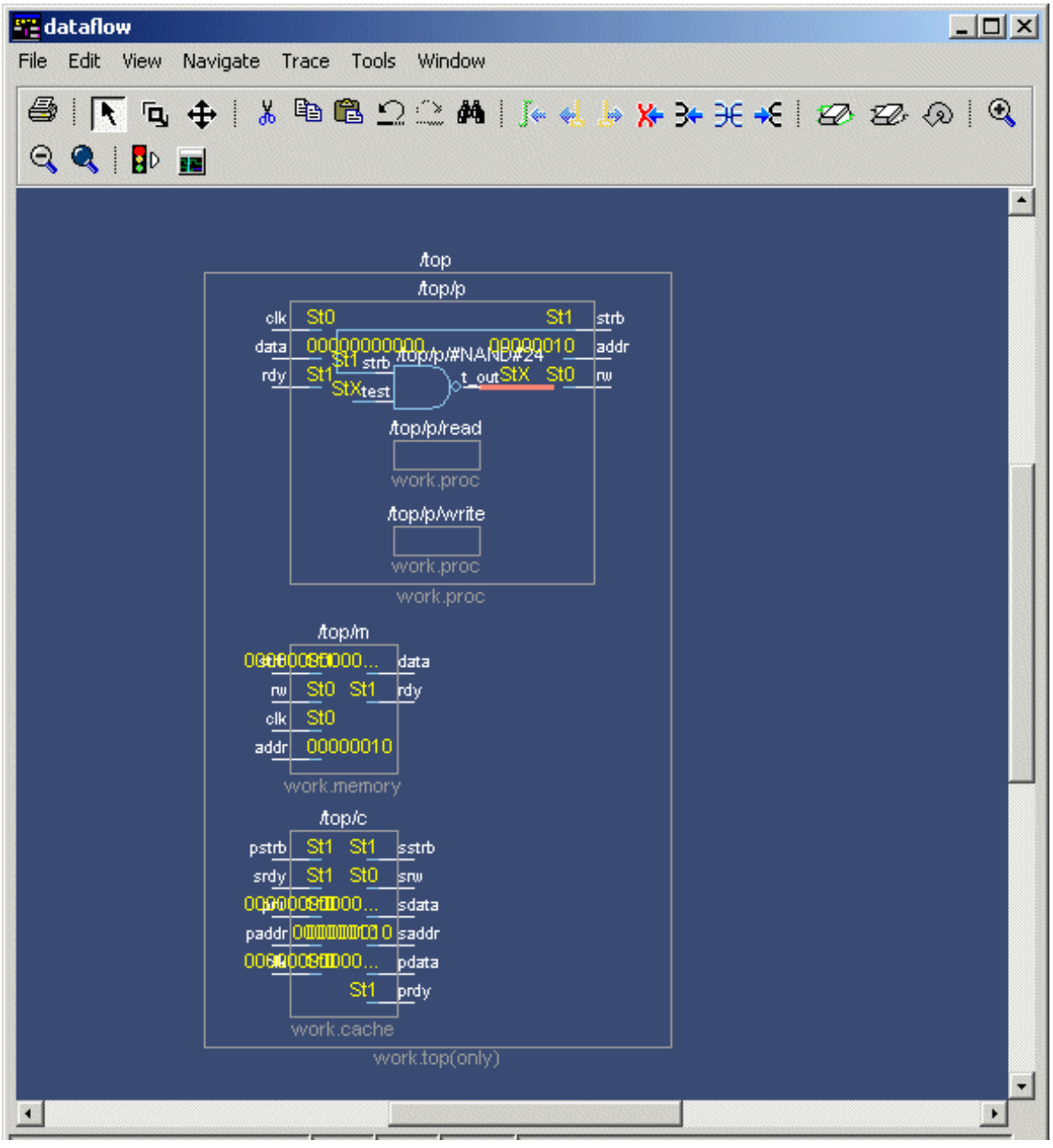
1 Select **Tools > Options** from the Dataflow window menu bar.

2 Check **Show Hierarchy** and then select OK.



3 Add signal *t_out* to the Dataflow window.

```
add dataflow /top/p/t_out
```



Zooming and panning

The Dataflow window offers several tools for zooming and panning the display. After reviewing the options below, try them out on the *cache* module design.

Zooming with toolbar buttons

These zoom buttons are available on the toolbar:

 **Zoom in 2x** zoom in by a factor of two from the current view	 **Zoom out 2x** zoom out by a factor of two from current view
 **Zoom Full** zoom out to view the entire schematic	

Zooming with the mouse

To zoom with the mouse, you can either use the middle mouse button or enter Zoom Mode by selecting **View > Zoom** and then use the left mouse button.

4 zoom options are possible by clicking and dragging in different directions:

- Down-Right: Zoom Area (In)
- Up-Right: Zoom Out (zoom amount is displayed at the mouse cursor)
- Down-Left: Zoom Selected
- Up-Left: Zoom Full

The zoom amount is displayed at the mouse cursor. A zoom operation must be more than 10 pixels to activate.

Panning with the mouse

To pan with the mouse you must enter Pan Mode by selecting **View > Pan**.

Now click and drag with the left mouse button to pan the design.

Lesson 12 - Running a batch-mode simulation

The goals for this lesson are:

- Run a batch-mode VHDL simulation
- Execute a macro (DO) file
- View a saved simulation

Batch-mode allows you to execute several commands that are written in a text file. You create a text file with the list of commands you wish to run, and then specify that file when you start ModelSim. This is particularly useful when you need to run a simulation or a set of commands repeatedly.

Important: Batch-mode simulations must be run from a DOS or UNIX prompt. Unless directed otherwise, enter all commands in this lesson at a DOS or UNIX prompt. Additionally, this lesson assumes you have added ModelSim to your PATH. If you did not, you'll need to specify full paths to the tools (i.e., vlib, vmap, vcom, and vsim) that are used in the lesson.

1. To set up for this lesson, create a new directory and copy this file into it:

   ```
   \<install_dir>\modeltech\examples\counter.vhd
   ```

2. Create a new design library (again, enter these commands at a DOS or UNIX prompt in the new directory you created in step 1.):

   ```
   vlib work
   ```

3. Map the library:

   ```
   vmap work work
   ```

4. Then compile the source file:

   ```
   <install_dir>/<platform>/vcom counter.vhd
   ```

5. You will use a macro file that provides stimulus for the counter. For your convenience, a macro file has been provided with ModelSim. You need to copy this macro file from the installation directory to the current directory:

   ```
   <install_dir>\modeltech\examples\stim.do
   ```

6. Create a batch file using an editor; name it *yourfile*. With the editor, put the following on separate lines in the file:

   ```
   add list -decimal *
   do stim.do
   write list counter.lst
   quit -f
   ```

   and save to the current directory.

7. To run the batch-mode simulation, enter the following at the command prompt:

   ```
   vsim -do yourfile -wlf saved.wlf counter -c
   ```

   This is what you just did in Step 7:

   - invoked the VSIM simulator on a design unit called "counter"
   - instructed the simulator to save the simulation results in a log file named *saved.wlf* by using the **-wlf** switch
   - used the contents of *yourfile* to specify that values are to be listed in decimal, to execute a stimulus file called *stim.do*, and to write the results to a file named *counter.lst*, the default for a design named *counter*

8. Since you saved the simulation results in *saved.wlf*, you can view the simulation results by starting up VSIM with its **-view** switch:

   ```
   vsim -view saved.wlf
   ```

9 Open these windows with the **View** menu in the Main window, or the equivalent command at the ModelSim prompt:

```
view signals list wave
```

Note: If you open the Process or Variables windows they will be empty. You are looking at a saved simulation, not examining one interactively; the logfile saved in *saved.wlf* was used to reconstruct the current windows.

10 Now that you have the windows open, put the signals in them:

```
add wave *
add list *
```

11 Use the available windows to experiment with the saved simulation results and quit when you are ready:

```
quit -f
```

For additional information on the batch and command line modes, please refer to the *ModelSim User's Manual*.

Lesson 13 - Executing commands at load time

The goals for this lesson are:

- Specify the design unit to be simulated on the command line
- Edit the *modelsim.ini* file
- Execute commands at load time with a DO file

Important: Start this lesson from either the UNIX or DOS prompt in the same directory in which you completed *Chapter Lesson 12 - Running a batch-mode simulation.*

1 For this lesson, you will use a macro (DO) file that executes whenever you load a design. For convenience, a startup file has been provided with the ModelSim program. You need to copy this DO file from the installation directory to your current directory:

```
\<install_dir>\modeltech\examples\startup.do
```

2 Next, you will edit the *modelsim.ini* file in the *\modeltech* directory (or the *modelsim.ini* file in your current directory if one exists) to point at this file. To do this, open *<install_dir>\modeltech\modelsim.ini* using a text editor and uncomment the following line (by deleting the leading ;) in the [vsim] section of the file:

```
Startup = do startup.do
```

Then save *modelsim.ini*.

Note: The *modelsim.ini* file must be write-enabled for this change to take place. Using MS Explorer, right-click on *\<install_dir>\modeltech\modelsim.ini*, then click Properties. In the dialog box, uncheck the Read-only box and click OK. (You can also copy the file to your current directory.)

3 Take a look at the DO file. It uses the predefined variable **$entity** to do different things when loading different designs.

4 Start the simulator and specify the top-level design unit to be simulated by entering the following command at the UNIX/DOS prompt:

```
vsim counter
```

Notice that the simulator loads the design unit without displaying the Load Design dialog box. This is handy if you are simulating the same design unit over and over. Also notice that all the windows are open. This is because the **view *** command is included in the startup macro.

5 If you plan to continue with the following practice sessions, keep ModelSim running. If you would like to quit the simulator, enter the following command at the VSIM prompt:

```
quit -f
```

6 You won't need the *startup.do* file for any other examples, so use your text editor to comment out the "Startup" line in *modelsim.ini*.

Lesson 14 - Tcl/Tk and ModelSim

The goals for this lesson are:

- Create a "hello world" button widget
- Execute a procedure using a push button
- Simulate an intersection with traffic lights
- Draw a state machine that represents the simulation

This lesson is divided into several Tcl examples intended to give you a sense of Tcl/Tk's function within ModelSim. The examples include a custom simulation interface created with Tcl/Tk (the code is already written).

Note: You must be using ModelSim SE-VHDL or ModelSim SE/MIXED to complete these exercises.

More information on Tcl/Tk

Sources of information about Tcl include *Tcl and the Tk Toolkit* by John K. Ousterhout, published by Addison-Wesley Publishing Company, Inc., and *Practical Programming in Tcl and Tk by* Brent Welch published by Prentice Hall.

Or, consult one of the following online Tcl references:

- Select **Help > Tcl Man Pages** (Main window) within ModelSim.
- http://dev.scriptics.com has many useful references
- The Model Technology web site lists a variety of Tcl resources: www.model.com/resources/tcltk.asp

How Tcl/Tk works with ModelSim

ModelSim incorporates Tcl as an embedded library package. The Tcl library consists of a parser for the Tcl language, routines to implement the Tcl built-in commands, and procedures that allow Tcl to be extended with additional commands specific to ModelSim.

ModelSim generates Tcl commands and passes them to the Tcl parser for execution. Commands may be generated by reading characters from an input source, or by associating command strings with ModelSim's user interface features, such as menu entries, buttons, or keystrokes.

When the Tcl interpreter receives commands it parses them into component fields and executes built-in commands directly. For commands implemented by ModelSim, Tcl calls back to the application to execute the commands. In many cases commands will invoke recursive invocations of the Tcl interpreter by passing in additional strings to execute (procedures, looping commands, and conditional commands all work in this way).

ModelSim gains a programming advantage by using Tcl for its command language. ModelSim can focus on simulation-specific commands, while Tcl provides many utility commands, graphic interface features, and a general programming interface for building up complex command procedures.

By using Tcl, ModelSim need not re-implement these features, a benefit that allows its graphic interface to remain consistent on all platforms. (The only vestige of the host platform's graphic interface is the window frame manager.)

The custom traffic-light interface

The subject of our main Tcl/Tk lesson is a simple traffic-light controller. The system is comprised of three primary components: a state machine, a pair of traffic lights, and a pair of traffic sensors. The components are described in three VHDL files: traffic.vhd (the state machine), queue.vhd (the traffic arrival queue) and tb_traffic.vhd (the testbench).

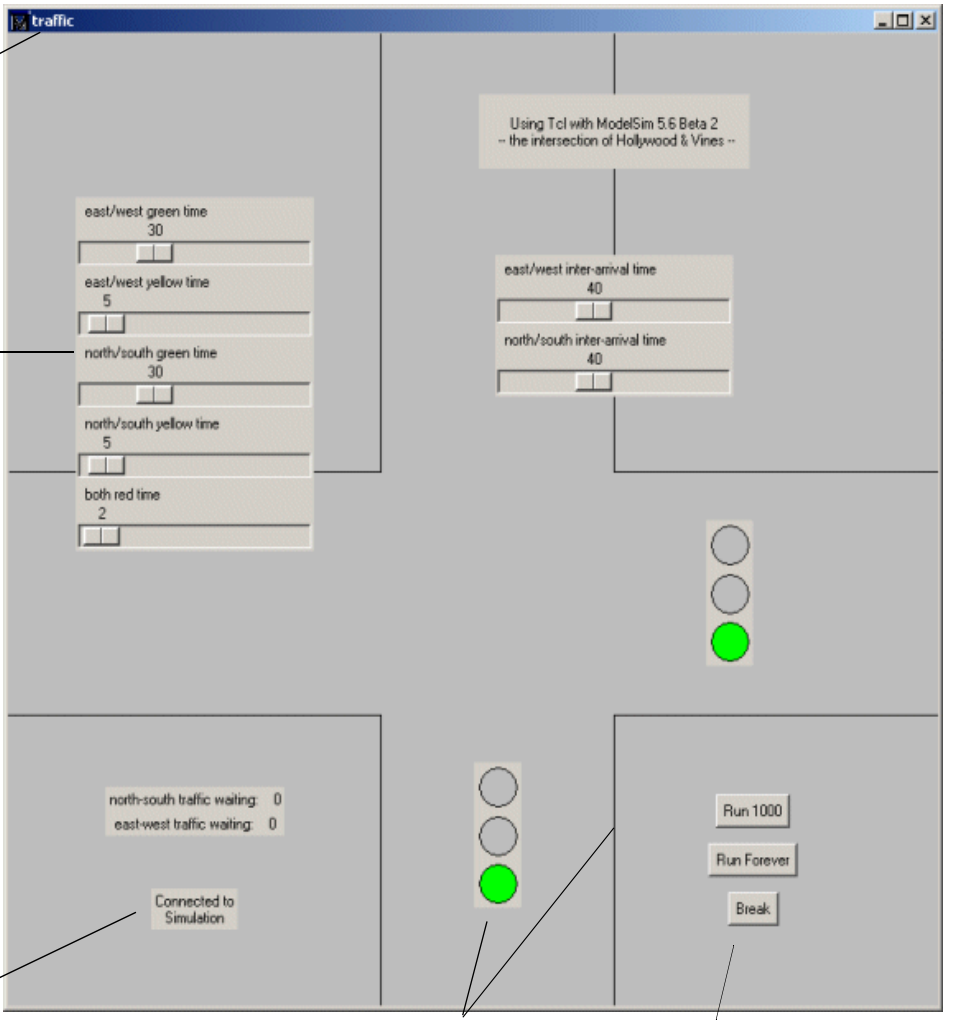
You could, of course, simulate this system with ModelSim's familiar interface, but Tcl/Tk provides us the option to try something different. Since we're simulating something most of us have seen and experienced before, we can create an intuitive interface unique to the simulation.

Overview

The table below summarizes the source files, procedures, and commands used to simulate the traffic-light controller.

VHDL source files describe the system	Tcl procedures create and connect the interface, plus the source files, to ModelSim	ModelSim commands are run via the new interface using the Tcl procedures
	draw_intersection	
traffic.vhd queue.vhd tb_traffic.vhd	connect_lights	vsim -lib vhdl/work tb_traffic examine -value <light_timing>
	draw_queues	
	draw_controls	force -freeze $var $val ns

The result is a traffic intersection interface similar to this illustration:



wm widget

Calls to the operating system window manager to create the "traffic" window.

frame and scale widget

A scale widget within a frame widget creates an analog entry device for a minimum to a maximum value and invokes the force command

label widget

A static label is added to the end of the connect_lights procedure to indicate connection to the simulator.

canvas widget

The background, lines and traffic lights are created with the canvas widget.

button widgets

Each button invokes the indicated run or break command.

Tk widgets

The intersection illustration points out several Tcl/Tk "widgets." A widget is simply a user interface element, like a menu or scrolled list. Tk widgets are referenced within Tcl procedures to create graphic interface objects. The Tk tool box comes with several widgets, additional widgets can be created using these as a base.

Controlling the simulation

The components of the intersection interface have the following effect within ModelSim:

Intersection control used	Effect in ModelSim
Run 1000 button	invokes the run command for 1000 ns
Run Forever button	invokes the run -all command
Break button	invokes the break command
light timing control	invokes the force command with the arguments for the indicated signal and time
arrival time control	invokes the force command with the arguments for the indicated direction and time
waiting queue	any time you change a control the examine command is invoked to display the value of the waiting queue

Saving time

Since several intersection controls invoke a command and arguments with a single action (such as the movement of a slider), this custom interface saves time compared to invoking the commands from the command line or ModelSim menus.

Copies of the original example files

Copies of the Tcl example files from these exercises are located in the *\<install_dir\>\modeltech\examples\tcl_tutorial\originals* directory.

Solutions to the examples

Throughout the traffic intersection examples you will be modifying Tcl files to complete the final intersection. You will find a completed set of intersection examples ready-to-run in the *tcl_tutorial\solutions* directory. Invoke these commands from the ModelSim prompt to run the intersection:

```
cd solutions
do traffic.do
```

Viewing files

If you would like to view the source for any of the Tcl files in our examples, use the **notepad** command at either the ModelSim or VSIM prompt.

```
notepad <filename>
```

Most files are opened in read-only mode by default; you can edit the file by deselecting **read only** from the notepad **Edit** menu.

The Tcl source command

The Tcl **source** command reads the Tcl file into the Tcl interpreter, which parses the procedures for use within the current environment. Once sourced, a Tcl procedure can be called from the ModelSim prompt as shown in the syntax below. ModelSim executes the instructions within the procedure.

Syntax

```
source <tcl filename>
<tcl procedure name>
```

Arguments

```
<tcl filename>
```

The Tcl file read into the ModelSim Tcl interpreter with the source command.

```
<tcl procedure name>
```

The Tcl procedure defined within <tcl filename>, called from the ModelSim prompt, and executed by ModelSim.

The *traffic.do* file is a good example of the **source** command syntax (the file is a macro that runs the traffic light simulation). View it with Notepad:

```
notepad traffic.do
```

Shortcuts

To save some typing, copy the commands from the PDF version of these instructions and paste them at the ModelSim prompt. Paste with the right (2 button mouse), or middle (3 button mouse). You can also select a ModelSim or VSIM prompt from the Main transcript to paste a previous command to the current command line.

Make a transcript DO file

You can rerun the commands executed during the current session with a DO file created from the Main transcript. Make the DO file by saving the transcript with the **File > Transcript > Save Transcript As** menu selection at any time during the exercises. Run the DO file to repeat the commands (do <do filename>).

Initial setup

Important: These steps must be completed before running the Tcl examples.

1. Create, and change to a new working directory for the Tcl/Tk exercises. Copy the lesson files in the following directory (include all subdirectories and files) to your new directory:

   ```
   <install_dir>\modeltech\examples\tcl_tutorial
   ```

2. Make the new directory the current directory, then invoke ModelSim:

   for UNIX

   ```
   vsim
   ```

   for Windows (from a shortcut or Start > Run, etc.)

   ```
   modelsim.exe
   ```

3. At the ModelSim prompt, create a **work** library in the */vhdl* directory:

   ```
   vlib vhdl/work
   ```

4. Map the **work** library.

   ```
   vmap work vhdl/work
   ```

5. Compile the VHDL example files with these commands (or the Compile dialog box):

   ```
   vcom vhdl/traffic.vhd
   vcom vhdl/queue.vhd
   vcom vhdl/tb_traffic.vhd
   ```

Example 1 - Create a "Hello World" button widget

Before you begin the examples make sure you have completed "Initial setup" (T-123).

In this example you will study a "hello world" button that prints a message when pressed.

1 Source the Tcl file from the ModelSim prompt:

```
source hello.tcl
```

then run the procedure defined within *hello.tcl*:

```
hello_example
```

The file *hello.tcl* was read into the ModelSim Tcl interpreter. The instructions in the *hello_example* procedure were then executed by ModelSim, and "Hello World" was printed to the Main transcript (or invoking shell on UNIX). Selecting the button will print the message again.

You've just created your first top-level widget!

2 Invoke the *hello_example* procedure again and notice how the new button replaces the original button. The procedure destroyed the first button and created the new one. Get a closer look at the source Tcl file with the **notepad**:

```
notepad hello.tcl
```

Close the hello_example window when you're done.

Example 2 - Execute a procedure using a push button

Before you begin this example make sure you have completed "Initial setup" (T-123).

This example will display all of the gif images in the images directory. Each button has a binding attached to it for "enter" events, and a binding for a mouse button press. When the mouse enters the button graphic, the image file name is printed to the Main window (or invoking shell on UNIX). When the mouse button is pushed, its "widget" name will be printed to the Main window (or invoking shell on UNIX).

1. Build an image viewer by invoking this command, and calling this procedure:

   ```
   source images.tcl
   image_example
   ```

2. Drag the mouse across the buttons and notice what happens in the Main transcript (or invoking shell on UNIX).

   Push one of the buttons; you will see an error dialog box. You can solve this problem by modifying the *images.tcl* file.

3. To view the source file press the **See Source Code** button at the bottom of the image display or invoke **notepad** at the ModelSim prompt:

   ```
   notepad images.tcl
   ```

   You'll find that the *pushme* procedure is missing; it's commented out in *images.tcl*.

4. Search for "proc push" using the **Edit > Find** menu selection in the notepad.

   Remove the comments (the "#" symbols) to return the function to your source, use **File > Save** to save the changes, then close the image window with the **Destroy** button.

5. Once the *pushme* procedure is in place it will print its one parameter, the object name, to the transcript.

   After you have added the *pushme* procedure to your source, you need to resource and rerun the Tcl procedure with these commands (use the up arrow to scroll through the commands or do !source):

   ```
   source images.tcl
   image_example
   ```

   Press all the buttons and notice the object names in the Main transcript. Close the image example window when you're done.

Example 3 - Simulate an intersection with traffic lights

In this example you'll simulate an intersection with traffic lights. The simulation interface you create allows you to run "what if" scenarios efficiently.

Introduction of the traffic intersection widget

This portion of our example introduces the traffic intersection widget. You'll add other widgets to the intersection to create a custom traffic simulation environment.

Once again, make sure you have completed "Initial setup" (T-123) before working this example.

1. Draw the intersection by invoking this command and procedure at the ModelSim prompt:

```
source intersection.tcl
draw_intersection
```

2. From the ModelSim prompt, use the procedure set_light_state to change the color of the lights:

```
set_light_state green .traffic.i.ns_light

set_light_state green .traffic.i.ew_light
```

   You can use the Copy and Paste buttons on the Main toolbar to help build instructions from previous commands.

3. View the source code with this command at the ModelSim prompt:

```
notepad intersection.tcl
```

   You can locate the *set_light_state* procedure with **Edit > Find** from the Notepad menu (the procedure is located toward the middle of the file).

Connect traffic lights to the simulation

Using the intersection widget, you will add *when* statements to connect the lights to the real simulation. Once the connection is made, you will simulate the traffic light controller and watch the lights change.

We'll use ModelSim *when* statements to condition the simulation to call our Tcl program when a desired simulation condition happens.

For our example, the desired condition is the state of the lights. Whenever the lights in the simulation change states, we want to change the color of the lights on the screen.

4. Load the VHDL libraries you compiled in preparation for these examples using this command at the ModelSim prompt:

```
vsim tb_traffic
```

   Be sure you invoke this command before the start of the connect_lights procedure, if you don't load the libraries, you won't have a design to simulate.

5 Connect the lights to the simulation with this command and procedure:

```
source lights.tcl
connect_lights
```

Try running the simulation now; select either run button on the intersection. Select **Break** if you used the **Run Forever** button. Notice how the Source window opens and indicates the next line to be executed. (If the simulator is not evaluating an executable process when the break occurs, the Source window will not open.) Only the East/West lights are working. You can make both lights work by editing the *lights.tcl* file.

6 Edit *lights.tcl* with the **notepad** to add a *when* statement for the North/South light.

```
notepad lights.tcl
```

You need to add this because the current statement is for the East/West light only. You'll find the solution commented. (Remember to change the read-only status of the file so you can edit it.)

You'll find the code commented-out toward the end of the file (use **Edit >Find** and look for light_ns).

7 After you have made the changes, reload and run the simulation again.

```
source lights.tcl
connect_lights
```

Both lights are now working.

Note: Remember, if you need to return to the original Tcl files (maybe you've edited the file and it doesn't work right) you'll find the files in the *tcl_tutorial\originals* directory.

Add widgets to display simulation information

Running the lights may be interesting, but not very useful - let's add some displays that will tell us what's happening to the cars at the intersection.

Now you will add queue widgets to display the sum of the length of each pair of queues as we simulate.

8 The East/West widget for displaying the total East/West queue length is already provided. Let's edit the source to add a display for the North/South direction. Use the **notepad**:

```
notepad queues.tcl
```

The solution is commented out in *queues.tcl*.

The Queue Display widget consists of an enclosing frame with two label widgets. The first label is a simple text string. The second label is the value of the queue length. The text in the second label will be updated whenever the queue lengths change.

9 After you have added your North/South widget, run your program by invoking this command:

```
source queues.tcl
draw_queues
```

According to the traffic indicators, the cars are leaving the intersection at the same rate. That seems fair, but if you are designing an intersection that responds to the traffic flow into the intersection you might want to change the light cycles. Perhaps one of the directions has more incoming traffic than the other.

Adding controls, in the form of scale widgets, allows you to quickly change the assumptions about traffic flow into the intersection.

Add "scale" widgets to control the simulation

Next you will add Tk "scale" widgets that will control the arrival rates and the length of the lights.

10 The East/West widget for controlling the East/West queue inter-arrival time is provided. You'll edit the source code to add controls for the North/South direction. Use this command:

```
notepad controls.tcl
```

You can remove the comments in the code to make this change.

Similarly, add the North/South widget for controlling the length of the lights. The East/West widget for light control is provided. (You can remove the comments in the code to make this change as well.)

These control widgets are implemented using the Tk "scale" widgets, enclosed in a frame.

When the value of the scale widget changes, it calls the command specified with the **-command** option on each scale.

11 After you have added your North/South widgets, run your program with this command:

```
source controls.tcl
draw_controls
```

Now you have a complete intersection interface. Try the run buttons and the slider scales.

You can view the simulation with ModelSim's GUI. Check the Source window to view the VHDL files, and add signals to a Wave window (**add wave ***).

You can also change the run length in the Main window. Try using the Run buttons in the Main window and the intersection window.

Keep the intersection simulation running to complete the next example. If you want to recreate the final intersection environment quickly, invoke these commands from the ModelSim prompt (after "Initial setup" (T-123)):

```
cd solutions
vmap work work
do traffic.do
```

Example 4 - Draw a state machine that represents the simulation

In this final example you will draw a state machine representing the simulation, and connect it to the state signal inside the traffic light controller. Each transition that the controller makes is displayed as it happens.

The intersection environment from the previous example needs to be running for this example. To get it running quickly, invoke these commands from the ModelSim prompt (after "Initial setup" (T-123)).

```
cd solutions
do traffic.do
```

1. Run the state machine with these commands:

   ```
   source state-machine.tcl
   draw_state_machine
   ```

   Click on one of the Run buttons.

2. Now we'll make some changes to the light colors and transition arrows. Open the source file with this command:

   ```
   notepad state-machine.tcl
   ```

   Note the "ModelSim EXAMPLE part 1" comments in the file. You can change "both_red" state coordinates from x = 125 and y = 50 to any coordinates. (You may need to uncheck the **read only** selection in the Edit menu before making changes.)

3. Note the "ModelSim EXAMPLE part 2" comments in the file. You can change the transition arrow coordinates to correspond with the new "both_red" state coordinates.

4. Note the "ModelSim EXAMPLE part 3" comments in the file. Change the active color from "black" to "purple".

5. Reuse the original commands when you're ready to run the state machine (remember, to copy a previous command to the current command line, select the previous ModelSim prompt):

   ```
   source state-machine.tcl
   draw_state_machine
   ```

   Click on one of the Run buttons.

   Notice the changes. Try some additional changes if you wish.

   This is the end of the Tcl/Tk examples. Continue to modify and test the examples if you wish; you can recover the original files at any time in the *tcl_tutorial\originals* directory.

License Agreement

IMPORTANT – USE OF THIS SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS

CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE SOFTWARE

This license is a legal “Agreement” concerning the use of Software between you, the end user, either individually or as an authorized representative of the company purchasing the license, and Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Mentor Graphics (Singapore) Private Limited, and their majority-owned subsidiaries (“Mentor Graphics”). USE OF SOFTWARE INDICATES YOUR COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. If you do not agree to these terms and conditions, promptly return or, if received electronically, certify destruction of Software and all accompanying items within 10 days after receipt of Software and receive a full refund of any license fee paid.

END USER LICENSE AGREEMENT

1. GRANT OF LICENSE. The software programs you are installing, downloading, or have acquired with this Agreement, including any updates, modifications, revisions, copies, and documentation (“Software”) are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics or its authorized distributor grants to you, subject to payment of appropriate license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form; (b) for your internal business purposes; and (c) on the computer hardware or at the site for which an applicable license fee is paid, or as authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Mentor Graphics’ then-current standard policies, which vary depending on Software, license fees paid or service plan purchased, apply to the following and are subject to change: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be communicated and technically implemented through the use of authorization codes or similar devices); (c) eligibility to receive updates, modifications, and revisions; and (d) support services provided. Current standard policies are available upon request.

2. ESD SOFTWARE. If you purchased a license to use embedded software development (“ESD”) Software, Mentor Graphics or its authorized distributor grants to you a nontransferable, nonexclusive license to reproduce and distribute executable files created using ESD compilers, including the ESD run-time libraries distributed with ESD C and C++ compiler Software that are linked into a composite program as an integral part of your compiled computer program, provided that you distribute these files only in conjunction with your compiled computer program. Mentor Graphics does NOT grant you any right to duplicate or incorporate copies of Mentor Graphics' real-time operating systems or other ESD Software, except those explicitly granted in this section, into your products without first signing a separate agreement with Mentor Graphics for such purpose.

3. BETA CODE.

3.1 Portions or all of certain Software may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to you a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and your use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.

3.2 If Mentor Graphics authorizes you to use the Beta Code, you agree to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. You will contact Mentor Graphics periodically during your use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of your evaluation and testing, you will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.

3.3 You agree that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceives or makes during or subsequent to this Agreement, including those based partly or wholly on your feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this subsection shall survive termination or expiration of this Agreement.

4. RESTRICTIONS ON USE. You may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. You shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. You shall not make Software available in any form to any person other than your employer's employees and contractors, excluding Mentor Graphics' competitors, whose job performance requires access. You shall take appropriate action to protect the confidentiality of Software and ensure that any person permitted access to Software does not disclose it or use it except as permitted by this Agreement. Except as otherwise permitted for purposes of interoperability as specified by the European Union Software Directive or local law, you shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive from Software any source code. You may not sublicense, assign or otherwise transfer Software, this Agreement or the rights under it without Mentor Graphics' prior written consent. The provisions of this section shall survive the termination or expiration of this Agreement.

5. LIMITED WARRANTY.

5.1 Mentor Graphics warrants that during the warranty period Software, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Software will

meet your requirements or that operation of Software will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. You must notify Mentor Graphics in writing of any nonconformity within the warranty period. This warranty shall not be valid if Software has been subject to misuse, unauthorized modification or installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND YOUR EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF SOFTWARE TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF SOFTWARE THAT DOES NOT MEET THIS LIMITED WARRANTY, PROVIDED YOU HAVE OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) SOFTWARE WHICH IS LOANED TO YOU FOR A LIMITED TERM OR AT NO COST; OR (C) EXPERIMENTAL BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

5.2 THE WARRANTIES SET FORTH IN THIS SECTION 5 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO SOFTWARE OR OTHER MATERIAL PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

6. LIMITATION OF LIABILITY. EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE STATUTE OR REGULATION, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT PAID BY YOU FOR THE SOFTWARE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER.

7. LIFE ENDANGERING ACTIVITIES. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF SOFTWARE IN ANY APPLICATION WHERE THE FAILURE OR INACCURACY OF THE SOFTWARE MIGHT RESULT IN DEATH OR PERSONAL INJURY. YOU AGREE TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE, OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH SUCH USE.

8. INFRINGEMENT.

8.1 Mentor Graphics will defend or settle, at its option and expense, any action brought against you alleging that Software infringes a patent or copyright in the United States, Canada, Japan, Switzerland, Norway, Israel, Egypt, or the

European Union. Mentor Graphics will pay any costs and damages finally awarded against you that are attributable to the claim, provided that you: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the claim; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the claim.

8.2 If an infringement claim is made, Mentor Graphics may, at its option and expense, either (a) replace or modify Software so that it becomes noninfringing, or (b) procure for you the right to continue using Software. If Mentor Graphics determines that neither of those alternatives is financially practical or otherwise reasonably available, Mentor Graphics may require the return of Software and refund to you any license fee paid, less a reasonable allowance for use.

8.3 Mentor Graphics has no liability to you if the alleged infringement is based upon: (a) the combination of Software with any product not furnished by Mentor Graphics; (b) the modification of Software other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of Software as part of an infringing process; (e) a product that you design or market; (f) any Beta Code contained in Software; or (g) any Software provided by Mentor Graphics' licensors which do not provide such indemnification to Mentor Graphics' customers.

8.4 THIS SECTION 8 STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS AND YOUR SOLE AND EXCLUSIVE REMEDY WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT BY ANY SOFTWARE LICENSED UNDER THIS AGREEMENT.

9. TERM. This Agreement remains effective until expiration or termination. This Agreement will automatically terminate if you fail to comply with any term or condition of this Agreement or if you fail to pay for the license when due and such failure to pay continues for a period of 30 days after written notice from Mentor Graphics. If Software was provided for limited term use, this Agreement will automatically expire at the end of the authorized term. Upon any termination or expiration, you agree to cease all use of Software and return it to Mentor Graphics or certify deletion and destruction of Software, including all copies, to Mentor Graphics' reasonable satisfaction.

10. EXPORT. Software is subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products, information about the products, and direct products of the products to certain countries and certain persons. You agree that you will not export in any manner any Software or direct product of Software, without first obtaining all necessary approval from appropriate local and United States government agencies.

11. RESTRICTED RIGHTS NOTICE. Software has been developed entirely at private expense and is commercial computer software provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the U.S. Government or a U.S. Government subcontractor is subject to the restrictions set forth in the license agreement under which Software was obtained pursuant to DFARS 227.7202-3(a) or as set forth in subparagraphs (c)(1) and (2) of the Commercial

Computer Software - Restricted Rights clause at FAR 52.227-19, as applicable. Contractor/manufacturer is Mentor Graphics Corporation, 8005 Boeckman Road, Wilsonville, Oregon 97070-7777 USA.

12. THIRD PARTY BENEFICIARY. For any Software under this Agreement licensed by Mentor Graphics from Microsoft or other licensors, Microsoft or the applicable licensor is a third party beneficiary of this Agreement with the right to enforce the obligations set forth in this Agreement.

13. CONTROLLING LAW. This Agreement shall be governed by and construed under the laws of Ireland if the Software is licensed for use in Israel, Egypt, Switzerland, Norway, South Africa, or the European Union, the laws of Japan if the Software is licensed for use in Japan, the laws of Singapore if the Software is licensed for use in Singapore, People's Republic of China, Republic of China, India, or Korea, and the laws of the state of Oregon if the Software is licensed for use in the United States of America, Canada, Mexico, South America or anywhere else worldwide not provided for in this section.

14. SEVERABILITY. If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.

15. MISCELLANEOUS. This Agreement contains the entire understanding between the parties relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions, except valid license agreements related to the subject matter of this Agreement which are physically signed by you and an authorized agent of Mentor Graphics. This Agreement may only be modified by a physically signed writing between you and an authorized agent of Mentor Graphics. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse. The prevailing party in any legal action regarding the subject matter of this Agreement shall be entitled to recover, in addition to other relief, reasonable attorneys' fees and expenses.

Rev. 03/00

Index

Numerics

A

B

C

D

E

F

H

I

K

L

W

Z