Model*Sim*®

SE

Tutorial

Version 5.8d

Published: 8/Mar/04

The world's most popular HDL simulator

Table of Contents

Introduction

Topics

The following topics are covered in this chapter:

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, Verilog, and/or SystemC. 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 or in the following formats and locations:

Document	Format	How to get it
ModelSim SE Installation & Licensing Guide	paper	shipped with Model*Sim*
	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 Model*Sim*
	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**
Foreign Language Interface Reference	PDF, HTML	select **Main window > Help > SE Documentation**
Std_DevelopersKit User's Manual	PDF	www.model.com/support/documentation/BOOK/sdk_um.pdf The Standard Developer's Kit is for use with Mentor Graphics QuickHDL.
Command Help	ASCII	type `help [command name]` at the prompt in the Main window
Error message help	ASCII	type `verror <msgNum>` at the Main window or 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*
Technotes	HTML	select Technotes dropdown on www.model.com/support

Technical support and updates

Support

Model Technology online and email technical support options, maintenance renewal, and links to international support contacts:
www.model.com/support/default.asp

Mentor Graphics support:
www.mentor.com/supportnet

Updates

Access to the most current version of ModelSim:
www.model.com/downloads/default.asp

Latest version email

Place your name on our list for email notification of news and updates:
www.model.com/products/informant.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.)

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

Example designs

ModelSim comes with Verilog and VHDL versions of the designs used in these lessons. This allows you to do the tutorial regardless of which license type you have. Though we have tried to minimize the differences between the Verilog and VHDL versions, we could not do so in all cases. In cases where the designs differ (e.g., line numbers or syntax), you will find language-specific instructions. Follow the instructions that are appropriate for the language that you are using.

Lesson 1 - ModelSim conceptual overview

Topics

The following topics are covered in this chapter:

Introduction

ModelSim is a simulation and debugging tool for VHDL, Verilog, and mixed-language designs.

This lesson provides a brief conceptual overview of the ModelSim simulation environment. It is divided into four topics, which you will learn more about in subsequent lessons:

Topic	Additional information and practice
Basic simulation flow	*Lesson 2 - Basic simulation*
Project flow	*Lesson 3 - ModelSim projects*
Multiple library flow	*Lesson 4 - Working with multiple libraries*
Debugging tools	Remaining lessons

Basic simulation flow

The following diagram shows the basic steps for simulating a design in ModelSim.

Create a working library

↓

Compile design files

↓

Run simulation

↓

Debug results
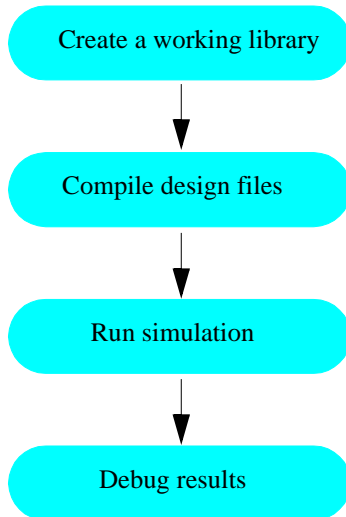
Creating the working library

In ModelSim, all designs, be they VHDL, Verilog, or a combination of the two, are compiled into a library. You typically start a new simulation in ModelSim by creating a working library called "work". "Work" is the library name used by the compiler as the default destination for compiled design units.

Compiling your design

After creating the working library, you compile your design units into it. The ModelSim library format is compatible across all supported platforms. You can simulate your design on any platform without having to recompile your design.

Running the simulation

With the design compiled, you invoke the simulator on a top-level module (Verilog) or a configuration or entity/architecture pair (VHDL). Assuming the design loads successfully, the simulation time is set to zero, and you enter a run command to begin simulation.

Debugging your results

If you don't get the results you expect, you can use ModelSim's robust debugging environment to track down the cause of the problem.

Project flow

A project is a collection mechanism for an HDL design under specification or test. Even though you don't have to use projects in ModelSim, they may ease interaction with the tool and are useful for organizing files and specifying simulation settings.

The following diagram shows the basic steps for simulating a design within a ModelSim project.

```
┌─────────────────────┐
│   Create a project   │
└─────────────────────┘
           │
           ▼
┌─────────────────────┐
│ Add files to the project │
└─────────────────────┘
           │
           ▼
┌─────────────────────┐
│  Compile design files  │
└─────────────────────┘
           │
           ▼
┌─────────────────────┐
│    Run simulation    │
└─────────────────────┘
           │
           ▼
┌─────────────────────┐
│    Debug results     │
└─────────────────────┘
```
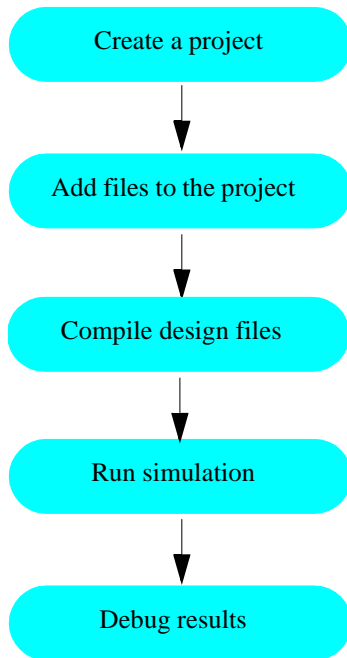
As you can see, the flow is similar to the basic simulation flow. However, there are two important differences:

• You do not have to create a working library in the project flow; it is done for you automatically.

• Projects are persistent. In other words, they will open every time you invoke ModelSim unless you specifically close them.

Multiple library flow

ModelSim uses libraries in two ways: 1) as a local working library that contains the compiled version of your design; 2) as a resource library. The contents of your working library will change as you update your design and recompile. A resour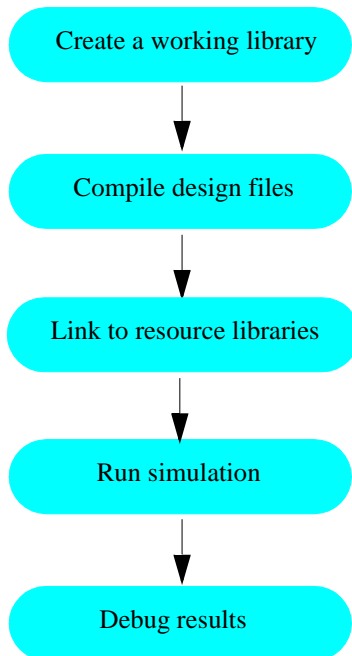ce library is typically static and serves as a parts source for your design. You can create your own resource libraries, or they may be supplied by another design team or a third party (e.g., a silicon vendor).

You specify which resource libraries will be used when the design is compiled, and there are rules to specify in which order they are searched. A common example of using both a working library and a resource library is one where your gate-level design and testbench are compiled into the working library, and the design references gate-level models in a separate resource library.

The diagram below shows the basic steps for simulating with multiple libraries.

```
Create a working library
          |
          v
Compile design files
          |
          v
Link to resource libraries
          |
          v
Run simulation
          |
          v
Debug results
```

You can also link to resource libraries from within a project. If you are using a project, you would replace the first step above with these two steps: create the project and add the testbench to the project.

Debugging tools

ModelSim offers numerous tools for debugging and analyzing your design. Several of these tools are covered in subsequent lessons, including:

- Setting breakpoints and stepping through the source code

- Viewing waveforms and measuring time

- Exploring the "physical" connectivity of your design

- Viewing and initializing memories

- Analyzing simulation performance

- Testing code coverage

- Comparing waveforms
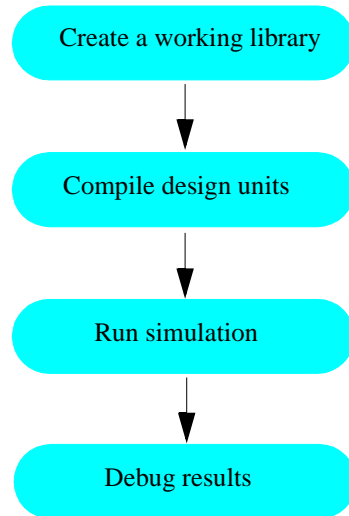
Lesson 2 - Basic simulation

Topics

The following topics are covered in this lesson:

Introduction

In this lesson you will go step-by-step through the basic simulation flow:

```
┌──────────────────────────┐
│  Create a working library │
└──────────────────────────┘
              │
              ▼
┌──────────────────────────┐
│    Compile design units   │
└──────────────────────────┘
              │
              ▼
┌──────────────────────────┐
│       Run simulation      │
└──────────────────────────┘
              │
              ▼
┌──────────────────────────┐
│       Debug results       │
└──────────────────────────┘
```

Design files for this lesson

The sample design for this lesson is a simple 8-bit, binary up-counter with an associated testbench. The pathnames are as follows:

Verilog – *<install_dir>/modeltech/examples/counter.v* and t*counter.v*

VHDL – *<install_dir>/modeltech/examples/counter.vhd* and *tcounter.vhd*

This lesson uses the Verilog files *counter.v* and *tcounter.v* in the examples. If you have a VHDL license, use *counter.vhd* and *tcounter.vhd* instead. Or, if you have a mixed license, feel free to use the Verilog testbench with the VHDL counter or vice versa.

Related reading

ModelSim User's Manual – Chapter 3 - Design libraries (UM-53), *Chapter 5 - Verilog simulation* (UM-105), *Chapter 4 - VHDL simulation* (UM-71)

ModelSim Command Reference (**vlib** (CR-344), **vmap** (CR-356), **vlog** (CR-345), **vcom** (CR-303), **vsim** (CR-357), **view** (CR-320), and **right** (CR-244) commands)

Creating the working design library

Before you can simulate a design, you must first create a library and compile the source code into that library.

1   Create a new directory and copy the tutorial files into it.

Start by creating a new directory for this exercise (in case other users will be working with these lessons). Create the directory, then copy *counter.v* and *tcounter.v* files from */<install_dir>/examples* to the new directory.

2   Start ModelSim if necessary.

a   Type **vsim** at a UNIX shell prompt or use the ModelSim icon in Windows.

Upon opening ModelSim for the first time, you will see the Welcome to ModelSim dialog (Figure 1). Click **Close**.

b   Select **File > Change Directory** and change to the directory you created in step 1.

3   Create the working library.

a   Select **File > New > Library**.

This opens a dialog where you specify physical and logical names for the library (Figure 2). You can create a new library or map to an existing library. We'll be doing the former.

b   Type **work** in the Library Name field if it isn't entered automatically.
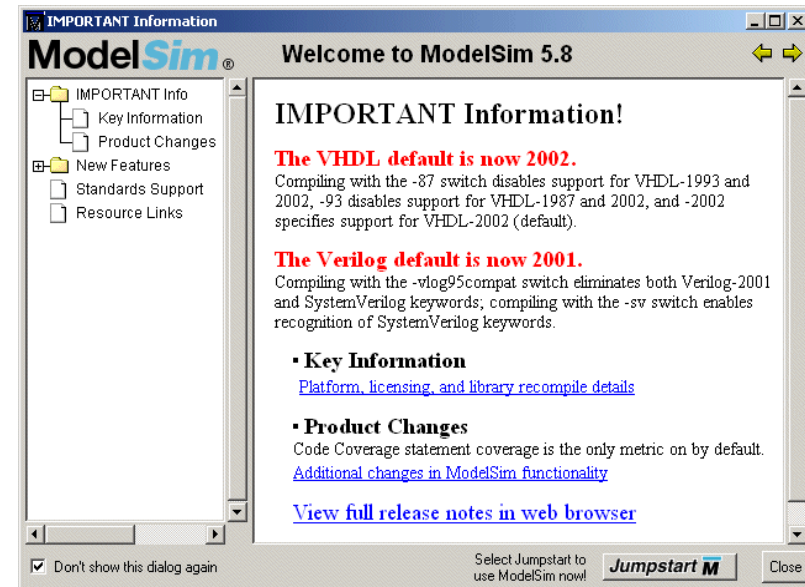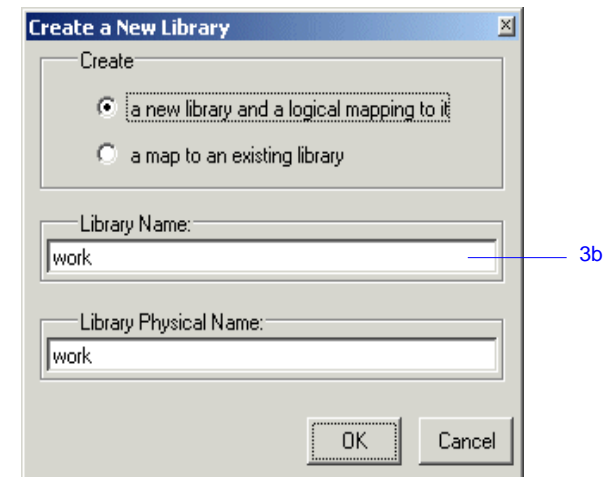
Figure 1: The Welcome Dialog

Figure 2: The Create a New Library dialog

c   Click **OK**.

ModelSim creates a directory called *work* and writes a specially-
formatted file named *_info* into that directory. The *_info* file must remain
in the directory to distinguish it as a ModelSim library. Do not edit the
folder contents from your operating system; all changes should be made
from within ModelSim.

ModelSim also adds the library to the list in the Workspace (Figure 3)
and records the library mapping for future reference in the ModelSim
initialization file (*modelsim.ini*).

When you pressed OK in step c above, three lines were printed to the Main
window Transcript pane:

```
vlib work
vmap work work
# Modifying modelsim.ini
```

The first two lines are the command-line equivalent of the menu commands you
invoked. Most menu driven functions will echo their command-line equivalents in
this fashion. The third line notifies you that the mapping has been recorded in the
ModelSim initialization file.

Figure 3: The newly created work library
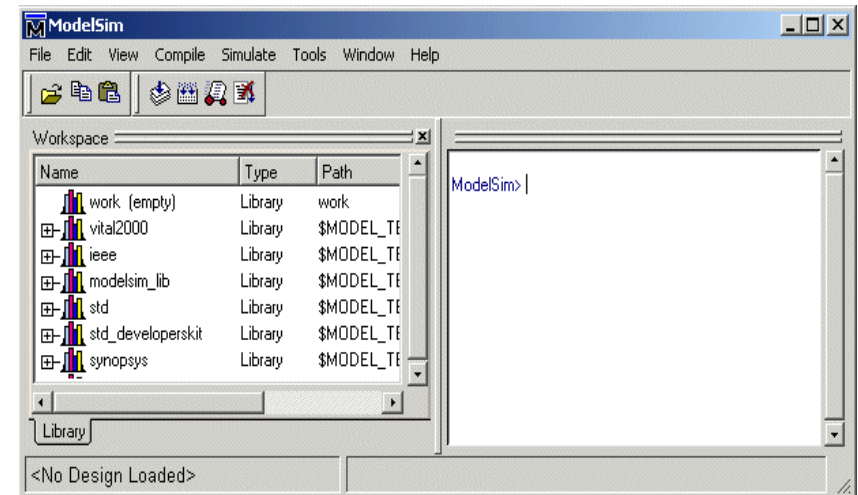
Compiling the design

With the working library created, you are ready to compile your source files.

1   Compile *counter.v* and *tcounter.v*.

    a   Select **Compile > Compile**.

       This opens the Compile Source Files dialog (Figure 4).

       If the Compile menu option is not available, you probably have a project open. If so, close the project by selecting **File > Close > Project**.

    b   Select *counter.v*, hold the <Ctrl> key down, and then select *tcounter.v*.

    c   With the two files selected, click **Compile**.

       The files are compiled into the *work* library.

    d   Click **Done**.

2   View the compiled design units.

    a   On the Library tab, click the '+' icon next to the *work* library and you will see two design units (Figure 5). If you scroll to the right, you will see their types (modules in this case) and the path to the underlying source files.
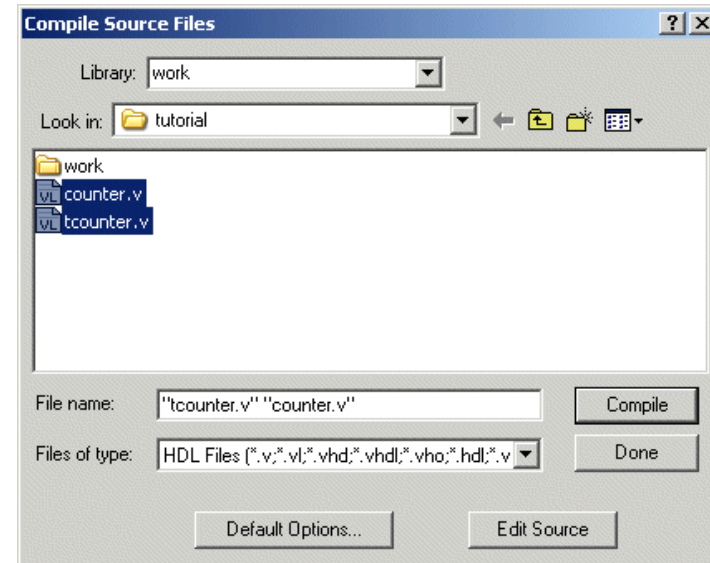
Figure 4: The Compile HDL Source Files dialog

Figure 5: Verilog modules compiled into the work library
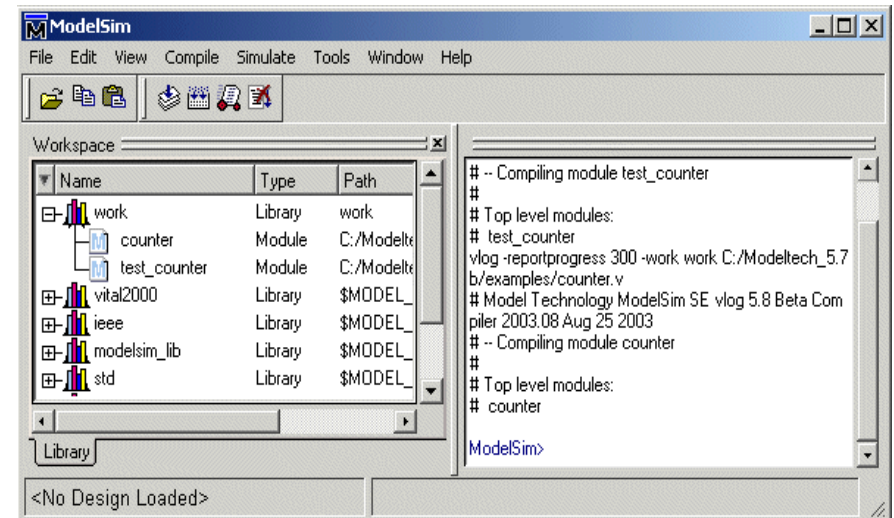
3   Load the *test_counter* module.

    b   Double-click *test_counter* to load the design.

        You will see a new tab named *sim* that displays the hierarchical structure of the design (Figure 6). You can navigate within the hierarchy by clicking on any line with a '+' (expand) or '-' (contract) icon. You will also see a tab named *Files* that displays all files included in the design.

Figure 6: Workspace tab showing a Verilog design
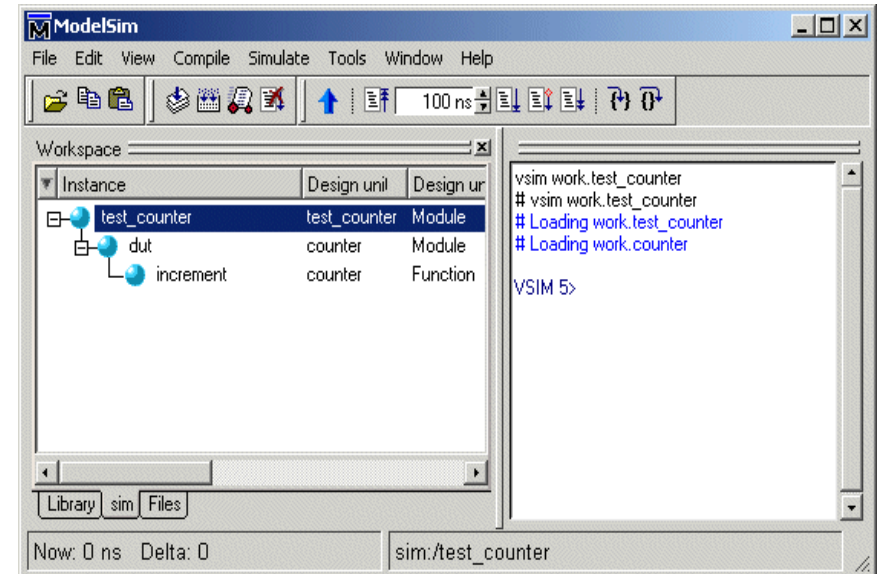
Running the simulation

Now you will run the simulation.

1   View all windows.

a   Select **View > All Windows**.

This opens all ModelSim windows, giving you different views of your design data and a variety of debugging tools. You may need to move or resize the windows to your liking.

2   Add signals to the Wave window.

a   In the Signals window, select **Add > Wave > Signals in Region** (Figure 7).

Three signals are added to the Wave window.

3   Run the simulation.

a   Click the Run icon on the Main, Wave, or Source window toolbar.

The simulation runs for 100 ns (the default simulation length) and waves are drawn in the Wave window (Figure 8).

b   Type **run 500** at the VSIM> prompt in the Main window.

The simulation advances another 500 ns for a total of 600 ns.

Figure 7: Adding signals to the Wave window

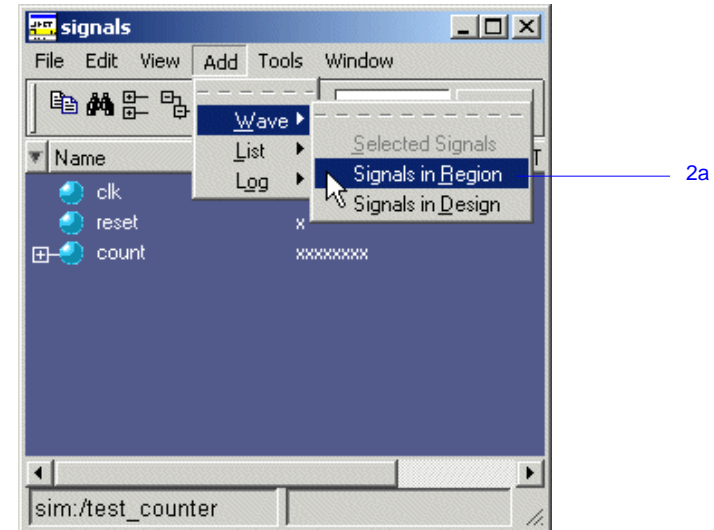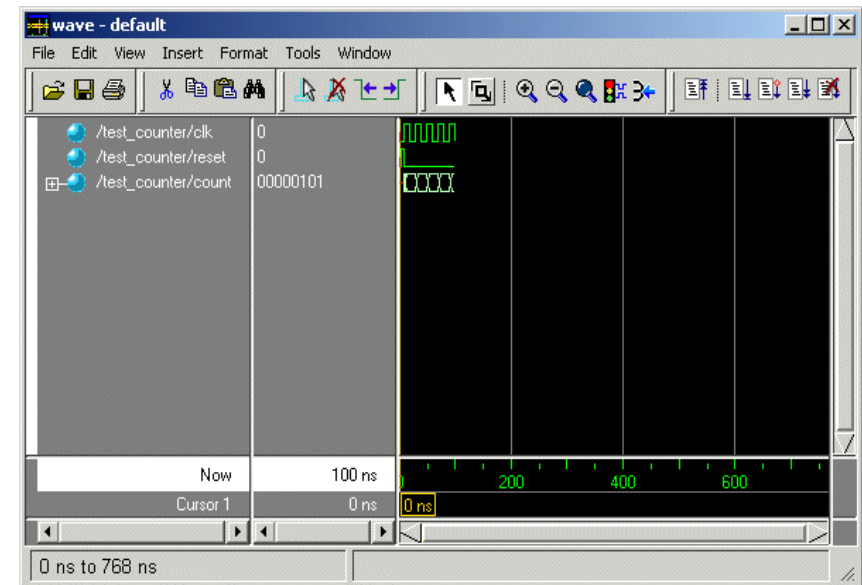**Figure 8: Waves being drawn in the Wave window**

c   Click the Run -All icon on the Main, Wave, or Source window toolbar.

The simulation continues running until you execute a break command or it hits a statement in your code (e.g., a Verilog $stop statement) that halts the simulation.

d   Click the Break icon.

The simulation stops running.

Setting breakpoints and stepping in the Source window

Next you will take a brief look at one interactive debugging feature of the ModelSim environment. You will set a breakpoint in the Source window, run the simulation, and then step through the design under test.

Breakpoints can be set only on lines with blue line numbers.

1 Set a breakpoint on line 28 of *counter.v* (if you are simulating the VHDL files, use line 30 instead).

    a Select *dut* in the *sim* tab of the Main window Workspace.

      This opens *counter.v* in the Source window.

    b Scroll to line 28 and click on or to the left of the line number.

      A red diamond appears next to the line (Figure 9).

2 Disable, enable, and delete the breakpoint.

    a Click the red diamond to disable the breakpoint.

    b Click the red diamond again to re-enable the breakpoint.

    c Click the red diamond with your right mouse button and select **Remove Breakpoint 28**.

    d Click on line 28 again to re-create the breakpoint.

3 Restart the simulation.

    a Click the Restart icon to reload the design elements and reset the simulation time to zero.

      The Restart dialog that appears gives you options on what to retain during the restart (Figure 10).

    b Click **Restart** in the Restart dialog.
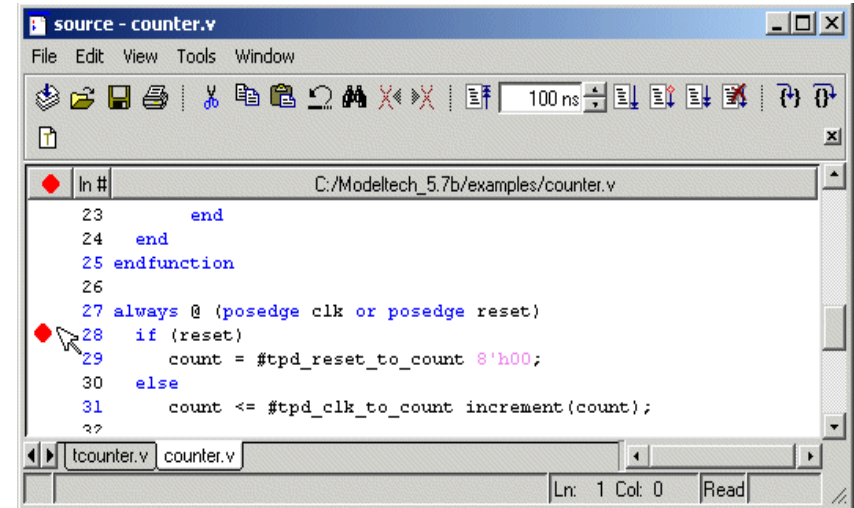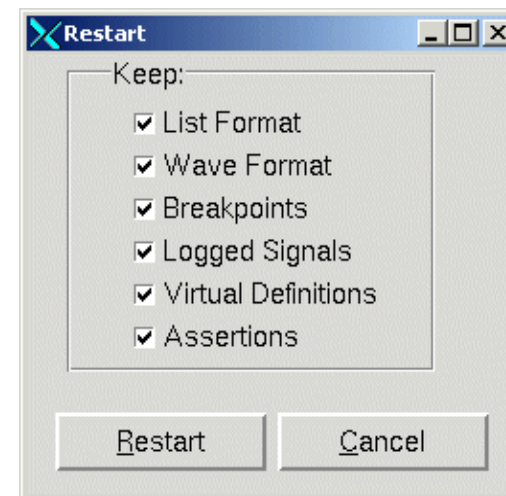
Figure 9: A breakpoint in the Source window

Figure 10: The Restart dialog

c   Click the Run -All icon on the Main, Wave, or Source window toolbar.

Figure 11: Resting the mouse pointer on a variable in the Source window

The simulation runs until the breakpoint is hit. When the simulation hits the breakpoint, it stops running, highlights the line with an arrow in the Source window (Figure 11), and issues a Break message in the Main window.

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

- look at the values shown in the Signals window

- set your mouse pointer over the *count* variable in the Source window, and a "balloon" will pop up with the value (Figure 11)

- highlight the *count* variable in the Source window, right-click it, and select Examine from the pop-up menu

- use the examine command to output the value to the Main window Transcript (i.e., `examine count`)

4   Try out the step commands.

a   Click the Step icon on the Source window toolbar.

This single-steps the debugger.

Experiment on your own. Set and clear breakpoints and use the Step, Step Over, and Continue Run commands until you feel comfortable with their operation.

b   When you are done experimenting, close the Source window by selecting **File > Close**.

Lesson wrap-up

This concludes this lesson. Before continuing we need to end the current simulation.

1   Select **Simulate > End Simulation**.

2   Click **Yes** when prompted to confirm that you wish to quit simulating.

Lesson 3 - ModelSim projects

Topics

The following topics are covered in this lesson:

Introduction

In this lesson you will practice creating a project. 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

This lesson uses the Verilog files *tcounter.v* and *counter.v* in the examples. If you have a VHDL license, use *tcounter.vhd* and *counter.vhd* instead.

Related reading

ModelSim User's Manual, Chapter 2 - Projects (UM-31)

Creating a new project

1   If you just finished the previous lesson, ModelSim should already be running.
     If not, start ModelSim.

   a   Type **vsim** at a UNIX shell prompt or use the ModelSim icon in
        Windows.

2   Create a new project.

   a   Select **Create a Project** from the Welcome dialog *or* **File > New >
        Project** (Main window) from the menu bar.

        This opens a dialog where you enter a Project Name, Project Location
        (i.e., directory), and Default Library Name (Figure 12). The default
        library is where compiled design units will reside.

   b   Type **test** in the Project Name field.

   c   Click **Browse** to select a directory where the project file will be stored.

   d   Leave the Default Library Name set to *work*.

   e   Click **OK**.

Figure 12: The Create Project dialog
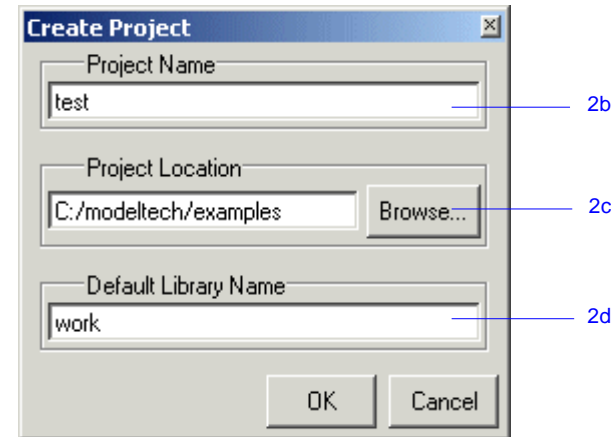
Adding items to the project

Once you click OK to accept the new project settings, you will see a blank Project tab in the workspace area of the Main window and the Add items to the Project dialog will appear (Figure 13). From this dialog you can create a new design file, add an existing file, add a folder for organization purposes, or create a simulation configuration (discussed below).

1   Add two existing files.

   a   Click **Add Existing File**.

        This opens the Add file to Project dialog (Figure 14). This dialog lets you browse to find files, specify the file type, specify which folder to add the file to, and identify whether to leave the file in its current location or to copy it to the project directory.
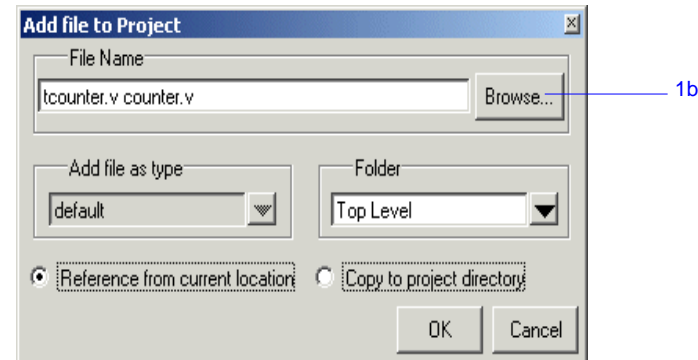
   b   Click **Browse**.

   c   Open the *examples* directory in your ModelSim installation tree.

   d   Select *counter.v*, hold the <Ctrl> key down, and then select *tcounter.v*.

   e   Click **Open** and then **OK**.

   f   Click **Close** to dismiss the Add items to the Project dialog.
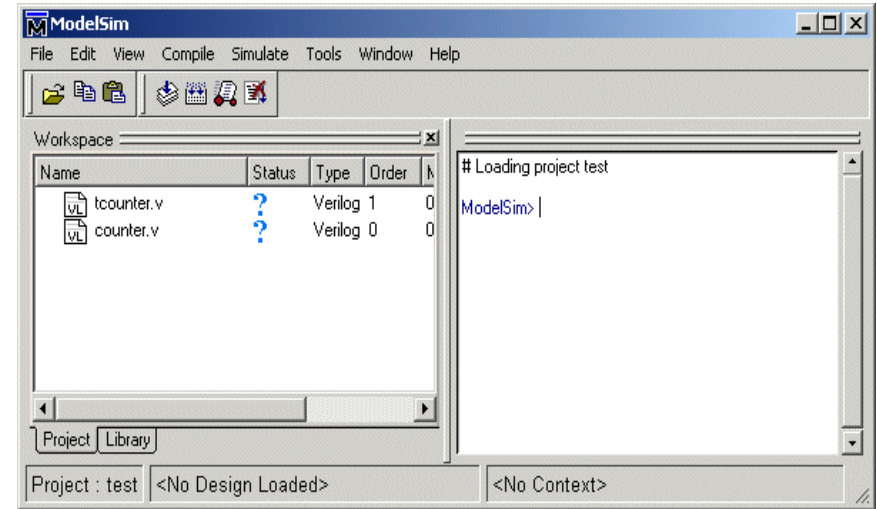
Figure 13: Adding new items to a project
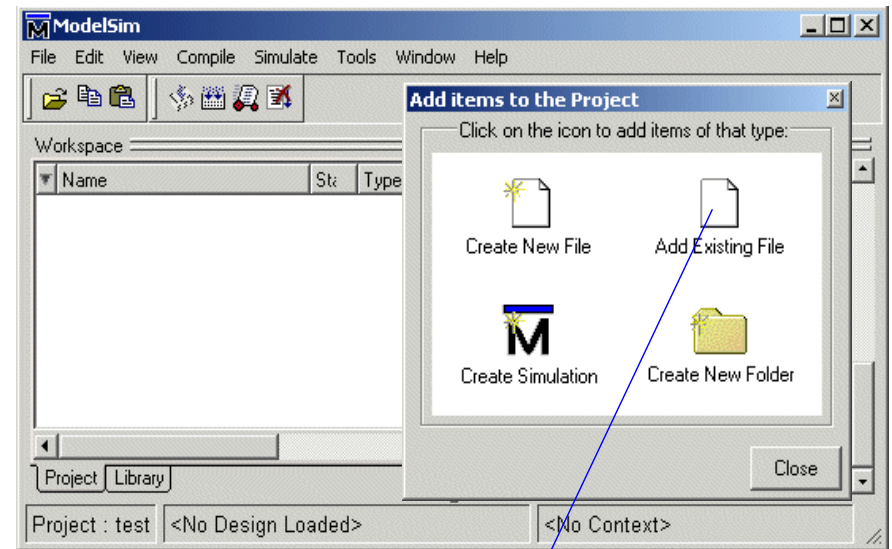
Figure 14: The Add File to Project dialog

You should now see two files listed in the Project tab of the Main window workspace (Figure 15).

Question mark icons (?) in the Status column mean the file hasn't been compiled or the source file has changed since the last successful compile. The other columns identify file type (e.g., Verilog or VHDL), compilation order, and modified date.

Changing compile order (VHDL)

Compilation order is important in VHDL designs. Follow these steps to change compilation order within a project.

1   Change the compile order.

   a   Select **Compile > Compile Order**.

      This opens the Compile Order dialog box (Figure 16).

   b   Click the **Auto Generate** button.

      ModelSim "determines" the compile order by making multiple passes over the files. It starts compiling from the top; if a file fails to compile due to dependencies, it moves that file to the bottom and then recompiles it after compiling the rest of the files. It continues in this manner until all files compile successfully or until a file(s) can't be compiled for reasons other than dependency.

      Alternatively, you can select a file and use the Move Up and Move Down buttons to put the files in the correct order.

   c   Click **OK** to close the Compile Order dialog.

Figure 15: Newly added project files display a '?' for status

Figure 16: The Compile Order dialog box
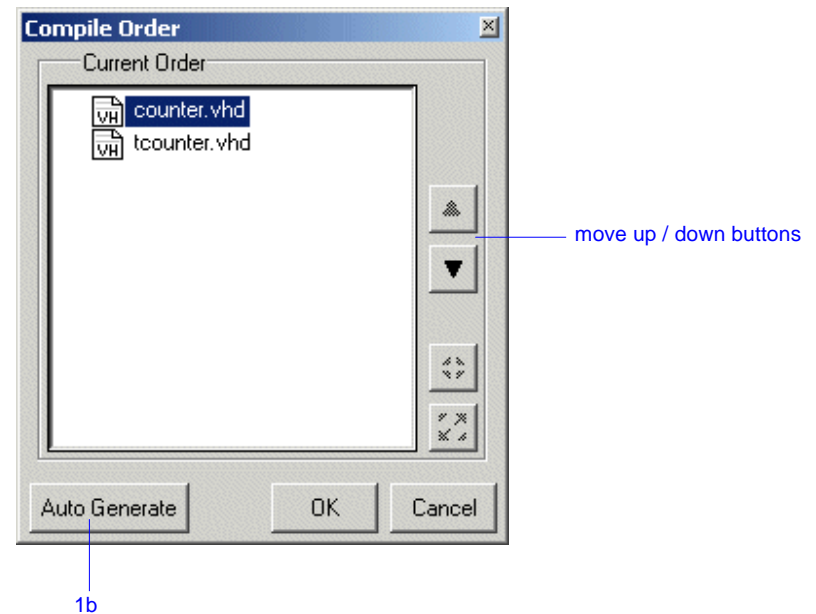
Compiling and loading a design

1   Compile the files.

   a   Right-click anywhere in the Project tab and select **Compile > Compile All** from the pop-up menu.

      ModelSim compiles both files and changes the symbol in the Status column to a check mark. A check mark means the compile succeeded. If the compile had failed, the symbol would be a red 'X', and you would see an error message in the Transcript window on the right.

2   View the design units.

   a   Click the **Library** tab in the workspace.

   b   Click the "+" icon next to the *work* library.

      You should see two compiled design units, their types (modules in this case), and the path to the underlying source files (Figure 17).

3   Load the *test_counter* design unit.

   a   Double-click the *test_counter* design unit.

      You should see a new tab named *sim* that displays the structure of the *test_counter* design unit (Figure 18). A fourth tab named *Files* contains information about the underlying source files.

      At this point you would generally run the simulation and analyze or debug your design like you did in the previous lesson. For now, you'll continue working with the project. However, first you need to end the simulation that started when you loaded *test_counter*.

4   End the simulation.

   a   Select **Simulate > End Simulation**.

   b   Click **Yes**.

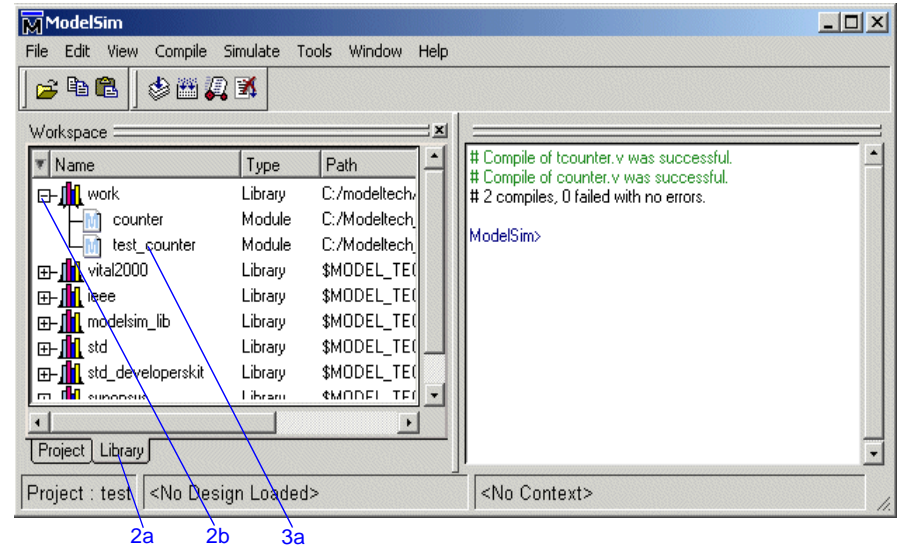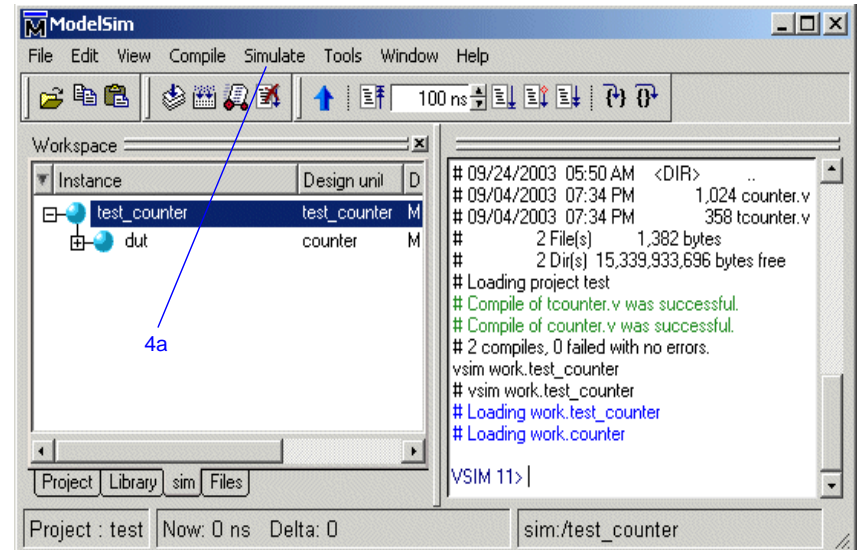**Figure 17: The Library tab with an expanded library**

Figure 18: The structure tab for the *counter* design unit

Organizing projects with folders

If you have a lot of files to add to a project, you may want to organize them in folders. You can create folders either before or after adding your files. If you create a folder before adding files, you can specify in which folder you want a file placed at the time you add the file (see Folder drop-down in Figure 14). If you create a folder after adding files, you edit the file properties to move it to that folder.

Adding folders

As shown previously in Figure 13, the Add items to the Project dialog has an option for adding folders. If you have already closed that dialog, you can use a menu command to add a folder.

1   Add a new folder.

a   Select **File > Add to Project > Folder**.

b   Type **Design Files** in the **Folder Name** field (Figure 19).

c   Click **OK**.

You'll now see a folder in the Project tab (Figure 20).

2   Add a sub-folder.

a   Right-click anywhere in the Project tab and select **Add to Project > Folder**.

b   Type **HDL** in the **Folder Name** field (Figure 21).

c   Click the **Folder Location** drop-down arrow and select *Design Files*.

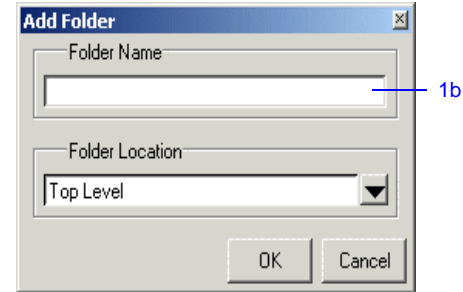d   Click OK.

Figure 19: Adding a new folder to the project

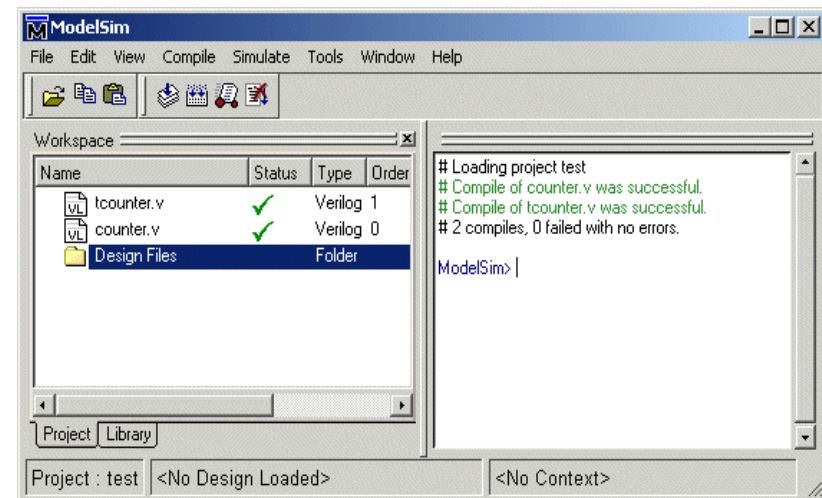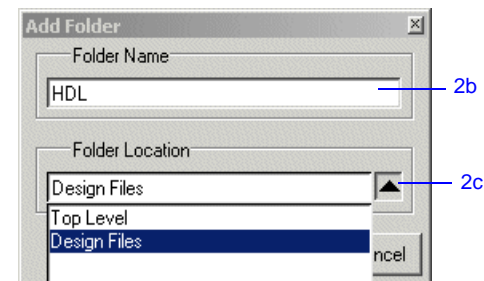**Figure 20: A folder in a project**

Figure 21: Creating a subfolder

ModelSim SE Tutorial

You'll now see a '+' icon next to the *Design Files* folder in the Project tab (Figure 22).

e   Click the '+' icon to see the *HDL* sub-folder.

Moving files to folders

Now that you have folders, you can move the files into them. If you are running on a Windows platform, you can simply drag-and-drop the files into the folder. On Unix platforms, you either have to place the files in a folder when you add the files to the project, or you have to move them using the properties dialog.

1   Move *tcounter.v* and *counter.v* to the *HDL* folder.

a   Select *counter.v*, hold the <Ctrl> key down, and then select *tcounter.v*.

b   Right-click either file and select **Properties**.

This opens the Project Compiler Settings dialog (Figure 23), which lets you set a variety of options on your design files.

c   Click the **Place In Folder** drop-down arrow and select *HDL*.

d   Click OK.

The two files are moved into the HDL folder. Click the '+' icons on the folders to see the files.

The files are now marked with a '?' icon. Because you moved the files, the project no longer knows if the previous compilation is still valid.
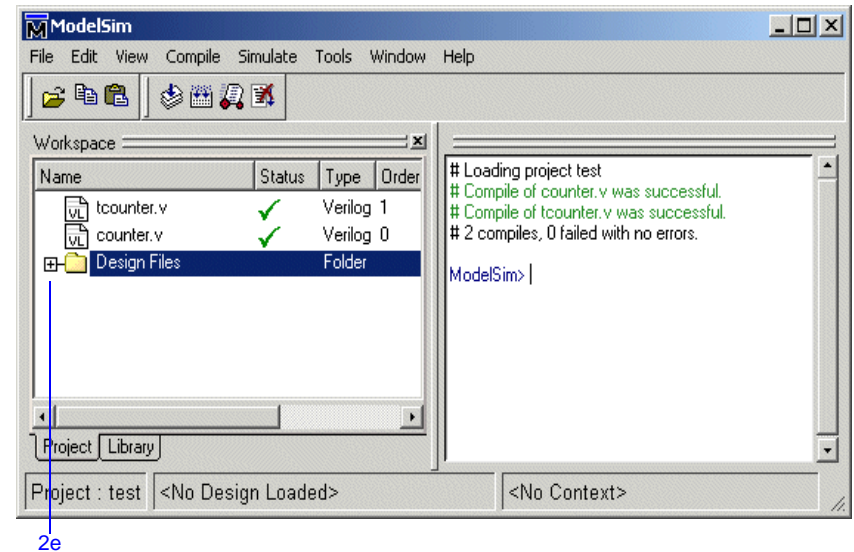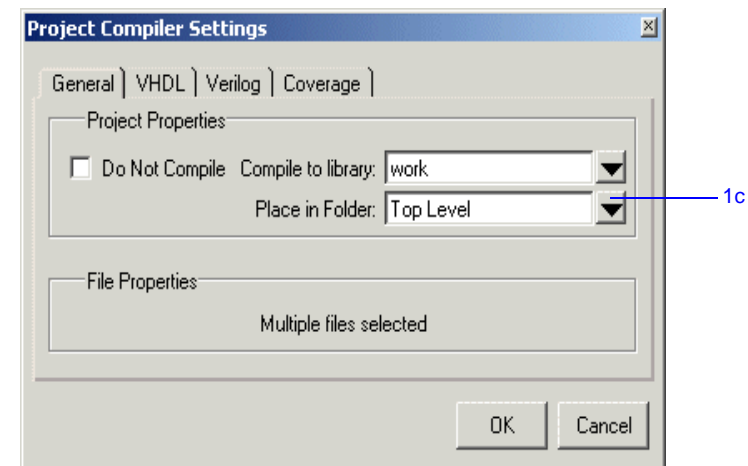
Figure 22: A folder with a sub-folder

Figure 23: Changing file location via the project settings dialog

Simulation Configurations

A Simulation Configuration associates a design unit(s) and its simulation options. For example, say every time you load *tcounter.v* you want to set the simulator resolution to picoseconds (ps) and enable event order hazard checking. Ordinarily you would have to specify those options each time you load the design. With a Simulation Configuration, you specify options for a design and then save a "configuration" that associates the design and its options. The configuration is then listed in the Project tab and you can double-click it to load *counter.v* along with its options.

1   Create a new Simulation Configuration.

   a   Select **File > Add to Project > Simulation Configuration**.

       This opens the Simulate dialog (Figure 24). The tabs in this dialog present a myriad of simulation options. You may want to explore the tabs to see what's available. You can consult the ModelSim User's Manual to get a description of each option.

   b   Type **counter** in the **Simulation Configuration Name** field.

   c   Select *HDL* from the **Place in Folder** drop-down.

   d   Click the '+' icon next to the *work* library and select *test_counter*.

   e   Click the **Resolution** drop-down and select *ps*.

   f   For Verilog, click the Verilog tab and check **Enable Hazard Checking**.

   g   Click **OK**.

       The Project tab now shows a Simulation Configuration named *counter* (Figure 25).

Figure 24: The Simulation Configuration dialog
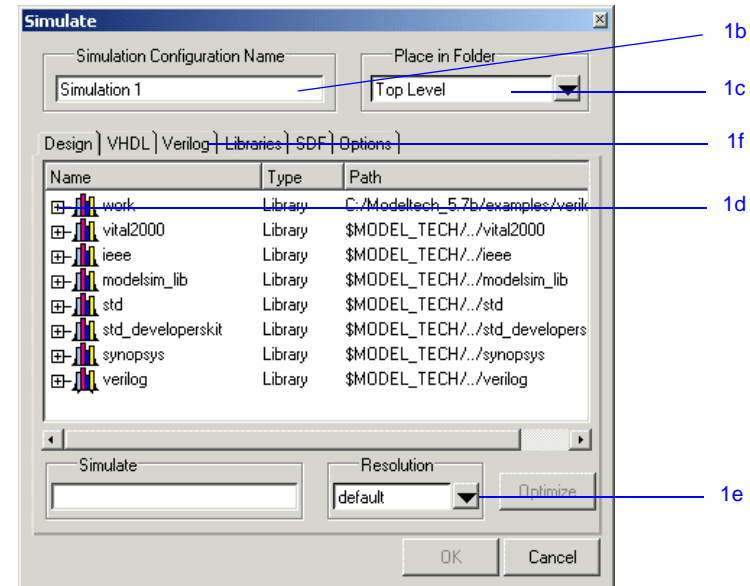
Figure 25: A Simulation Configuration in the Project tab
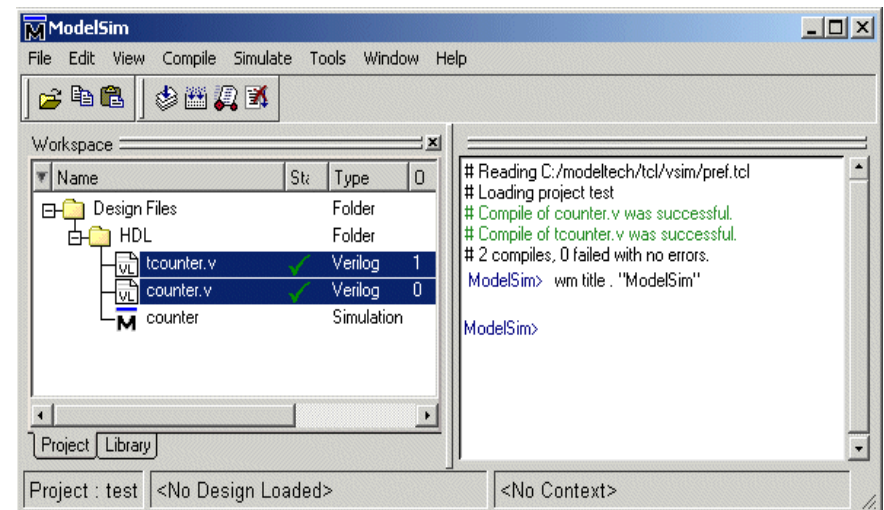
2   Load the Simulation Configuration.

   a   Double-click the *counter* Simulation Configuration in the Project tab.

      In the Transcript pane of the Main window, the **vsim** (the ModelSim simulator) invocation shows the **-hazards** and **-t ps** switches (Figure 26). These are the command-line equivalents of the options you specified in the Simulate dialog.

Figure 26: Transcript shows options used for Simulation Configuration
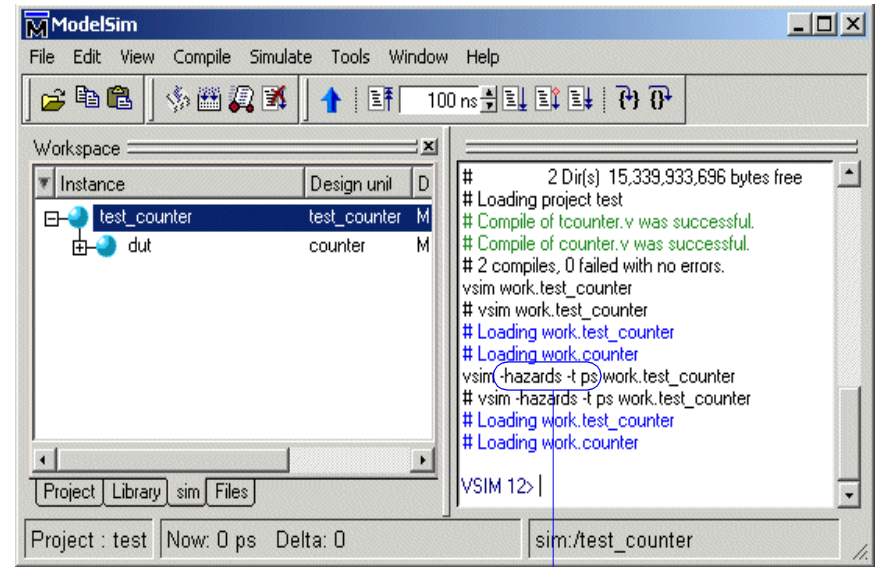
command-line switches

Lesson wrap-up

This concludes this lesson. Before continuing you need to end the current simulation and close the current project.

1   Select **Simulate > End Simulation**. Click Yes.

2   Select **File > Close > Project**. Click OK.

   If you do not close the project, it will open automatically next time you start ModelSim.

Lesson 4 - Working with multiple libraries

Topics

The following topics are covered in this lesson:

Introduction

In this lesson you will practice working with multiple libraries. As discussed in *Lesson 1 - ModelSim conceptual overview*, you might have multiple libraries to organize your design, to access IP from a third-party source, or to share common parts between simulations.

You will start the lesson by creating a resource library that contains the *counter* design unit. Next, you will create a project and compile the testbench into it. Finally, you will link to the library containing the counter and then run the simulation.

Design files for this lesson

The sample design for this lesson is a simple 8-bit, binary up-counter with an associated testbench. The pathnames are as follows:

Verilog – *<install_dir>/modeltech/examples/counter.v* and t*counter.v*

VHDL – *<install_dir>/modeltech/examples/counter.vhd* and *tcounter.vhd*

This lesson uses the Verilog files *tcounter.v* and *counter.v* in the examples. If you have a VHDL license, use *tcounter.vhd* and *counter.vhd* instead.

Related reading

ModelSim User's Manual, *3 - Design libraries* (UM-53)

Creating the resource library

1 Create a directory for the resource library.

   Create the directory that will hold the resource library. Copy *counter.v* from *<install_dir>/modeltech/examples* to the new directory.

2 Create a directory for the testbench.

   Create a new the directory that will hold the testbench and project files. Copy *tcounter.v* from *<install_dir>/modeltech/examples* to the new directory.

   You are creating two directories in this lesson to mimic the situation where you receive a resource library from a third-party. As noted earlier, we will link to the resource library in the first directory later in the lesson.

3 Start ModelSim and change to the exercise directory.

   If you just finished the previous lesson, ModelSim should already be running. If not, start ModelSim.

   a Type **vsim** at a UNIX shell prompt or use the ModelSim icon in Windows.

      If the Welcome to ModelSim dialog appears, click **Close**.

   b Select **File > Change Directory** and change to the directory you created in step 1.

4 Create the resource library.

   a Select **File > New > Library**.

   b Type **parts_lib** in the Library Name field (Figure 27).

      The Library Physical Name field is filled out automatically.

      Once you click OK, ModelSim creates a directory for the library, lists it in the Library tab of the Workspace, and modifies the *modelsim.ini* file to record this new library for the future.

Figure 27: Creating the new resource library
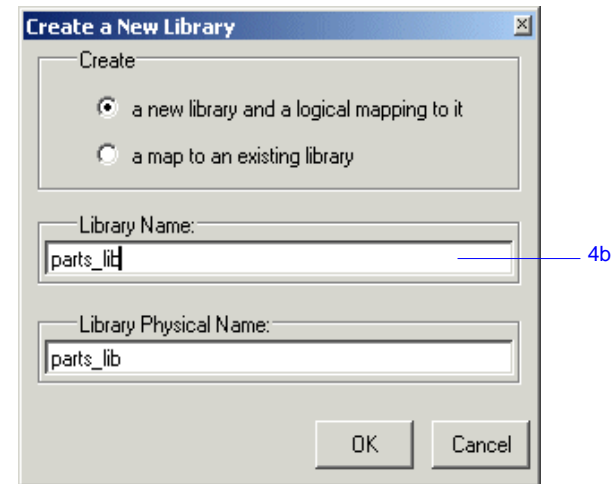
5   Compile the counter into the resource library.

    a   Click the Compile icon on the Main window toolbar.

Figure 28: Compiling into the resource library
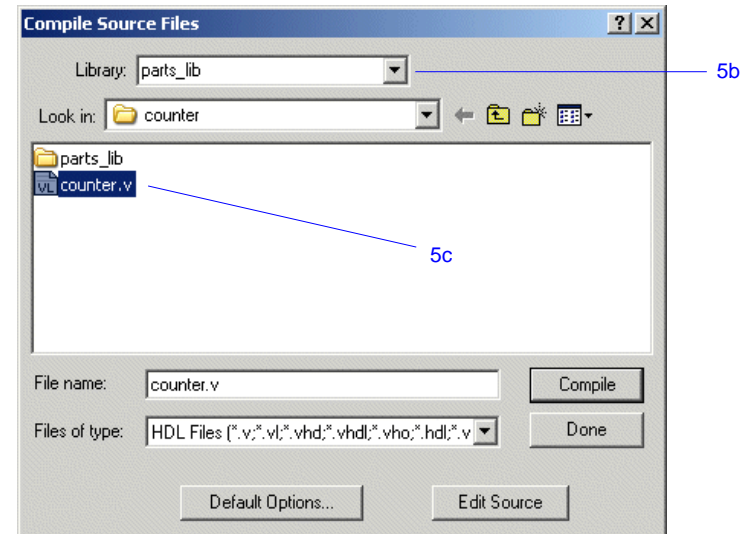

    b   Select the *parts_lib* library from the Library list (Figure 28).

    c   Double-click *counter.v*.

    d   Click Done.

You now have a resource library containing a compiled version of the *counter* design unit.

6   Change to the directory you created in step 2.

    a   Select **File > Change Directory** and change to the directory you created in step 2.

Creating the project

Now you will create a project that contains *tcounter.v*, the counter's testbench.

1   Create the project.

    a   Select **File > New > Project**.

    b   Type **counter** in the Project Name field.

    c   Click **OK**.

    d   If a dialog appears asking about which *modelsim.ini* file to use, click **Use Default Ini**.

2   Add the testbench to the project.

    a   Click **Add Existing File** in the Add items to the Project dialog.

    b   Click the Browse button and select *tcounter.v*.

    c   Click Open and then OK.

    d   Click Close to dismiss the Add items to the Project dialog.

       The *tcounter.v* file is listed in the Project tab of the Main window.

3   Compile the testbench.

    a   Right-click *tcounter.v* and select **Compile > Compile Selected**.

Linking to the resource library

To wrap up this part of the lesson, you will link to the *parts_lib* library you created earlier. But first, try simulating the testbench without the link and see what happens.

ModelSim responds differently for Verilog and VHDL in this situation.

Verilog

1 Simulate a Verilog design with a missing resource library.

    a   In the Library tab, click the '+' icon next to the *work* library and double-click *test_counter*.

        The Main window Transcript reports an error (Figure 29). When you see a message that contains text like "Error: (vsim-3033)", you can view more detail by using the **verror** command.

    b   Type **verror 3033** at the ModelSim> prompt.

        The expanded error message tells you that a design unit could not be found for instantiation. It also tells you that the original error message should list which libraries ModelSim searched. In this case, the original message says ModelSim searched only *work*.

VHDL

1 Simulate a VHDL design with a missing resource library.

    a   In the Library tab, click the '+' icon next to the *work* library and double-click *test_counter*.

        The Main window Transcript reports a warning(Figure 30). When you see a message that contains text like "Warning: (vsim-3473)", you can view more detail by using the **verror** command.

    b   Type **verror 3473** at the ModelSim> prompt.

        The expanded error message tells you that a component ('dut' in this case) has not been explicitly bound and no default binding can be found.

    c   Type **quit -sim** to quit the simulation.

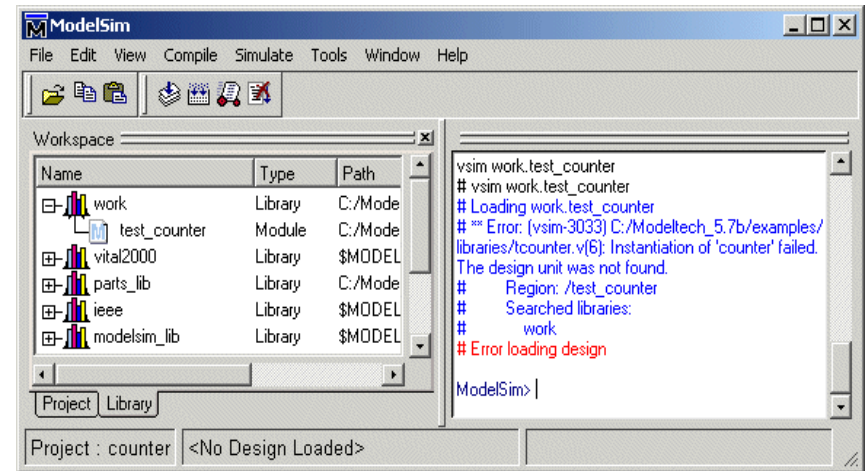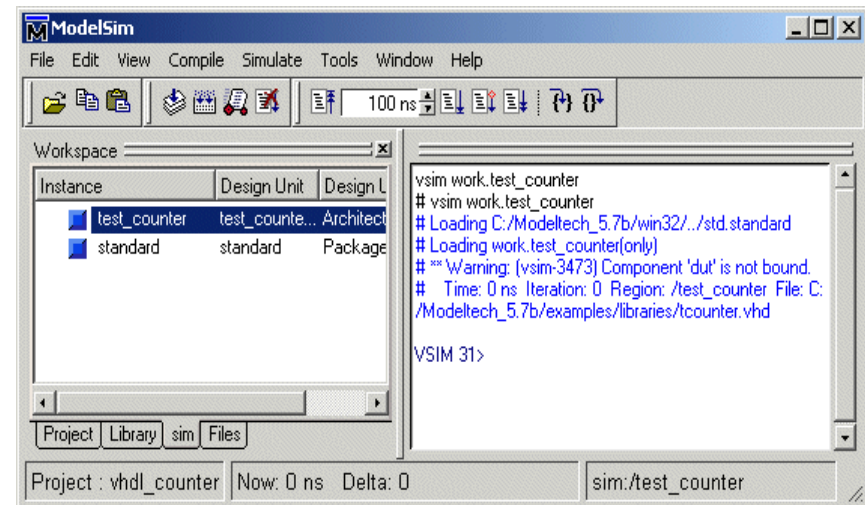**Figure 29: Verilog simulation error reported in the Main window**

Figure 30: VHDL simulation warning reported in Main window

The process for linking to a resource library differs between Verilog and VHDL. If you are using Verilog, follow the steps in "Linking in Verilog" (T-49). If you are using VHDL, follow the steps in "Linking in VHDL" (T-50) one page later.

Linking in Verilog

Linking in Verilog requires that you specify a "search library" when you invoke the simulator.

1   Specify a search library during simulation.

    a   Click the Simulate icon on the Main window toolbar.

    b   Click the '+' icon next to the *work* library and select *test_counter*.

    c   Click the Libraries tab.

    d   Click the Add button next to the Search Libraries field and browse to *parts_lib* in the first directory you created earlier in the lesson.

    e   Click Open.

       The dialog should have *parts_lib* listed in the Search Libraries field (Figure 31).

    f   Click OK.

       The design loads without errors.

Figure 31: Specifying a search library in the Simulate dialog
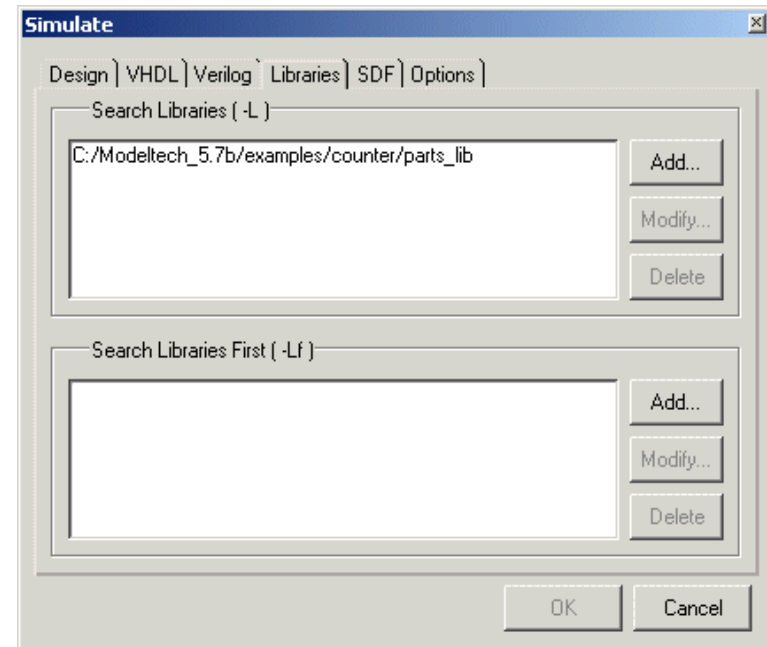
Linking in VHDL

To link to a resource library in VHDL, you have to create a logical mapping to the physical library and then add LIBRARY and USE statements to the source file.

1   Create a logical mapping to *parts_lib*.

   a   Select **File > New > Library**.

   b   In the Create a New Library dialog, select **a map to an existing library**.

   c   Type **parts_lib** in the Library Name field.

   d   Click Browse and browse to *parts_lib* in the first directory you created earlier in the lesson.

      The dialog should look similar to the one shown in Figure 32.

2   Add LIBRARY and USE statements to *tcounter.vhd*.

   a   Right-click *tcounter.vhd* in the Library tab and select **Edit**.

      This opens the file in the Source window.

   b   Add these two lines to the top of the file:

```
LIBRARY parts_lib;
USE parts_lib.ALL;
```

   c   Select **File > Save**.

3   Recompile and simulate.

   a   In the Project tab of the Main window, right-click *tcounter. vhd* and select **Compile > Compile Selected**.

   b   In the Library tab of the Main window, click the '+' icon next to the *work* library and double-click *test_counter*.

      The design loads without errors.

Figure 32: Mapping to the parts_lib library
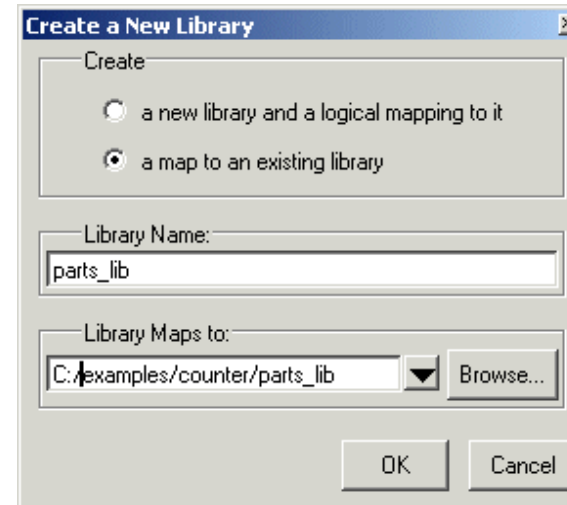
Permanently mapping resource libraries

If you reference particular resource libraries in every project or simulation, you may want to permanently map the libraries. Doing this requires that you edit the master *modelsim.ini* file in the installation directory. Though you won't actually practice it in this tutorial, here are the steps for editing the file:

1   Locate the *modelsim.ini* file in the ModelSim installation directory (*<install_dir>/modeltech/modelsim.ini*).

2   IMPORTANT - Make a backup copy of the file.

3   Change the file attributes of modelsim.ini so it is no longer "read-only."

4   Open the file and enter your library mappings in the [Library] section. For example:

```
parts_lib = C:/libraries/parts_lib
```

5   Save the file.

6   Change the file attributes so the file is "read-only" again.

Lesson wrap-up

This concludes this lesson. Before continuing we need to end the current simulation and close the project.

1   Select **Simulate > End Simulation**. Click Yes.

2   Select **File > Close > Project**. Click OK.

Lesson 5 - Simulating designs with SystemC

Topics

The following topics are covered in this lesson:

▶ **Note:** In the current release SystemC works only on Linux, HP, and Sun platforms. It is not yet available for AIX or Windows.

The functionality described in this tutorial requires a systemc license feature in your ModelSim license file. Please contact your Mentor Graphics sales representatives if you currently do not have such a feature.

Introduction

ModelSim treats SystemC as just another design language. With only a few exceptions in the current release, you can simulate and debug your SystemC designs the same way you do HDL designs.

Design files for this lesson

The sample design for this lesson is a ring buffer where the testbench and top-level chip are implemented in SystemC and the lower-level modules are written in HDL.

The pathnames to the files are as follows:

SystemC/Verilog – *<install_dir>/modeltech/examples/systemc/sc_vlog*

SystemC/VHDL – *<install_dir>/modeltech/examples/systemc/sc_vhdl*

This lesson uses the SystemC/Verilog version of the design in the examples. If you have a VHDL license, use the VHDL version instead. There is also a mixed version of the design, but the instructions here do not account for the slight differences in that version.

Related reading

ModelSim User's Manual – Chapter 7 - SystemC simulation (UM-187), *Chapter 8 - Mixed-language simulations* (UM-209), *Chapter 14 - C Debug* (UM-473)

ModelSim Command Reference – **sccom** command (CR-248)

Setting up the environment

SystemC is a licensed feature. You need the *systemc* license feature in your ModelSim license file to simulate SystemC designs. Please contact your Mentor Graphics sales representatives if you currently do not have such a feature.

The table below shows the supported operating systems for SystemC and the corresponding required versions of a C compiler.

Platform	Supported compiler versions
HP-UX 11.0 or later	aCC 3.45 with associated patches
RedHat Linux 7.3 or later	gcc 3.2
SunOS 5.6 or later	gcc 3.2

See *SystemC simulation* in the *ModelSim User's Manual* for further details.

Compiling and loading the example design

With designs that contain SystemC objects, you compile SystemC files using the
sccom compiler and HDL files using **vlog** or **vcom**. You also must link the created
C object files using **sccom -link**. In this exercise you will use a DO file to compile
and load the design.

1   Create a new directory and copy the tutorial files into it.

   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 files from
   <install_dir>/modeltech/examples/systemc/sc_vlog into the new directory.

   If you have a VHDL license, copy the files in *<install_dir>/modeltech/
   examples/systemc/sc_vhdl* instead.

2   Start ModelSim and change to the exercise directory.

   If you just finished the previous lesson, ModelSim should already be running.
   If not, start ModelSim.

   a   Type **vsim** at a UNIX shell prompt.

      If the Welcome to ModelSim dialog appears, click **Close**.

   b   Select **File > Change Directory** and change to the directory you created
      in step 1.

3   Execute the lesson DO file.

   a   Type **do run.do** at the ModelSim> prompt.

      The DO file does the following:

      •   Creates the working library

      •   Compiles links the C source files

      •   Compiles the HDL design files

      •   Opens all windows

      •   Loads the design

      Feel free to open the DO file and look at its contents. The DO file will
      take a little while to run.

Viewing SystemC items in the GUI

SystemC items are denoted in the ModelSim GUI with a green 'S' on the Library tab, a green 'C' on the Files tab, and a green diamond icon elsewhere.

1   Expand the *work* library in the Main window.

   a   Click the '+' icon next to the work library in the Library tab of the Main window.

        SystemC items have a green 'S' or 'C' next to their names (Figure 33).

2   Observe window linkages.

   a   Select a different instance in the sim tab of the Main window (Figure 34).

        The Source, Signals, and Process windows update to show the associated SystemC or HDL items.

3   Add items to the Wave window.

   a   Right-click *test_ringbuf* in the sim tab and select **Add > Add to Wave**.

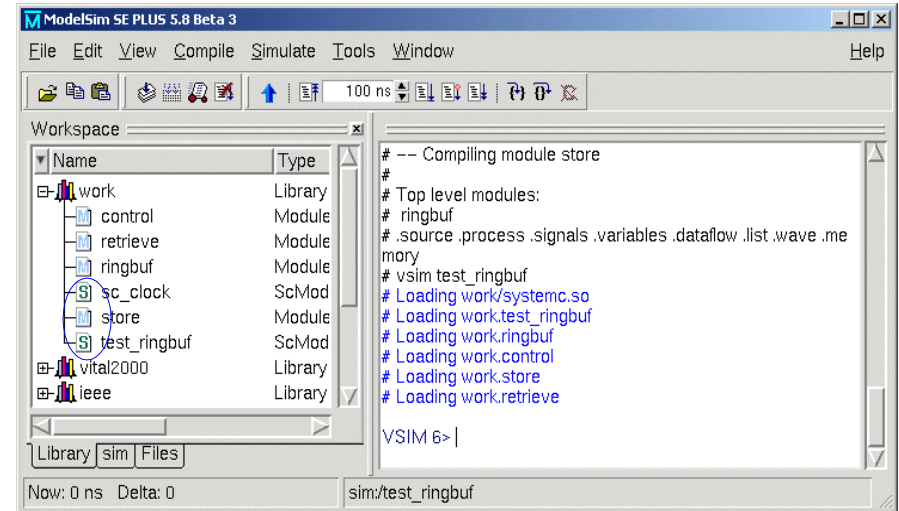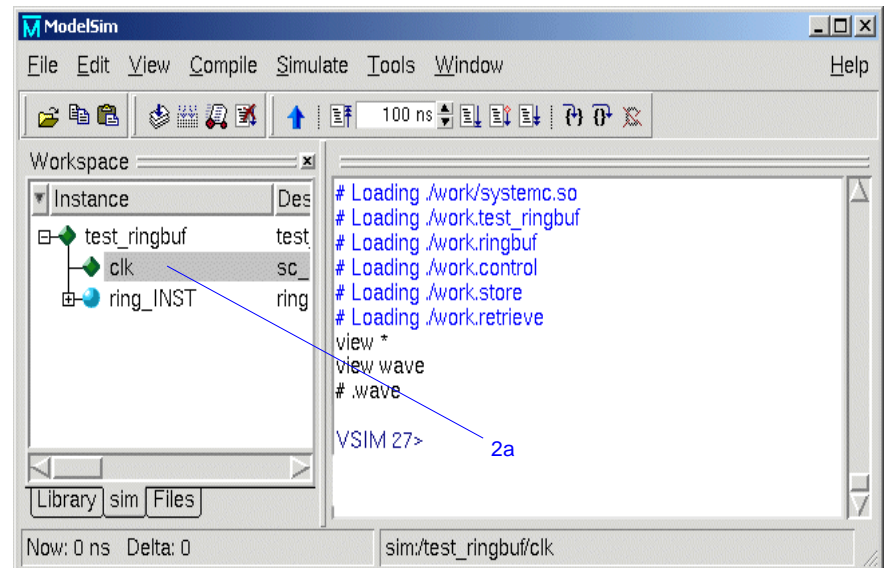**Figure 33: SystemC items in the work library**

Figure 34: SystemC items in the Main window sim tab

Setting breakpoints and stepping in the Source window

As with HDL files, you can set breakpoints and step through SystemC files in the Source window. In the case of SystemC, ModelSim uses C Debug, an interface to the open-source **gdb** debugger. Please see the C Debug chapter in the *ModelSim User's Manual* for complete details.

1 Set a breakpoint on line 142 of *test_ringbuf.h*.

   a If necessary, select *test_ringbuf* in the Main window workspace.

   b In the Source window, scroll to line 142 and click on or to the left of the line number.

     ModelSim recognizes that the file contains SystemC code, so it automatically launches C Debug. Once the debugger is running, ModelSim places a solid red diamond next to the line number (Figure 35).

2 Run and step through the code.

   a Type **run 500** at the VSIM> prompt.

     When the simulation hits the breakpoint, it stops running, highlights the line with an arrow in the Source window (Figure 36), and issues the following message in the Main window:

```
# C breakpoint c.1
# test_ringbuf::compare_data (this=0x842f658) at
test_ringbuf.h:142
```

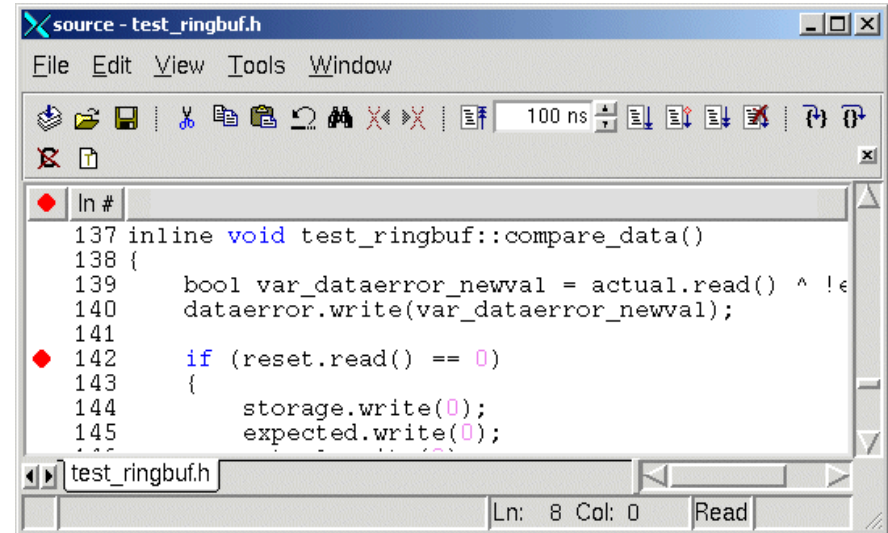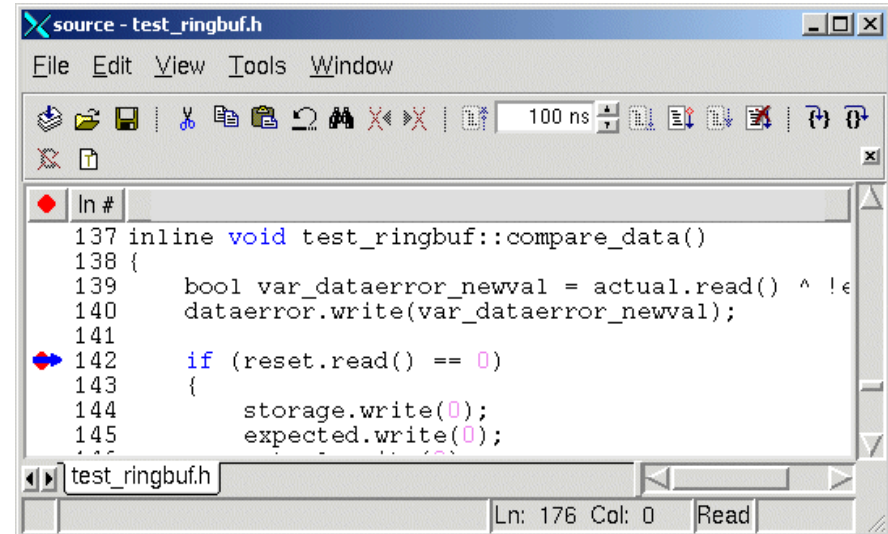**Figure 35: An active breakpoint in a SystemC file**

Figure 36: Simulation stopped at the breakpoint

b   Click the Step icon on the Source window toolbar.

This steps the simulation to the next statement. Because the next statement is a function call, ModelSim steps into the function, which is in a separate file (Figure 37).

c   Click the Continue Run icon on the Source window toolbar.

The breakpoint in *test_ringbuf.h* is hit again.

d   Right-click the red diamond on line 142 and select **Remove Breakpoint 142**.

e   Click the Continue Run button again.

The simulation runs for 500 ns and waves are drawn in the Wave window (Figure 38).

If you are using the VHDL version, you might see warnings in the Main window transcript. These warnings are related to VHDL value conversion routines and can be ignored.

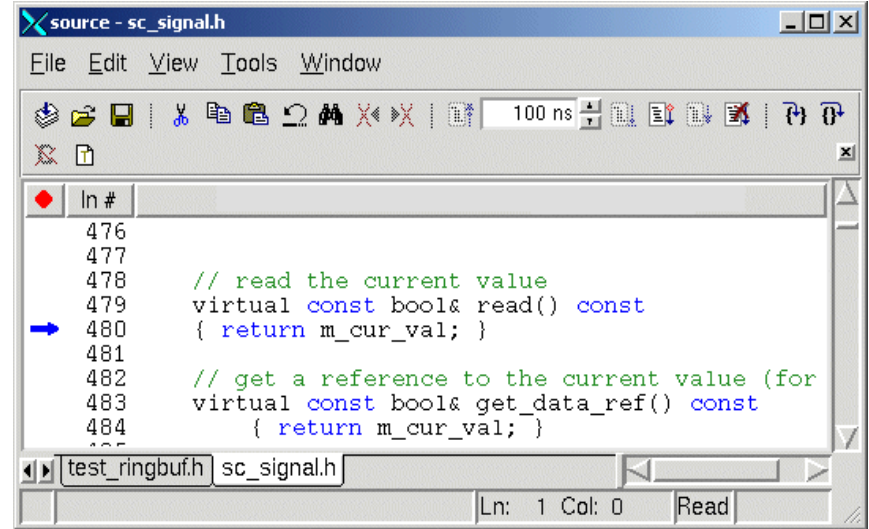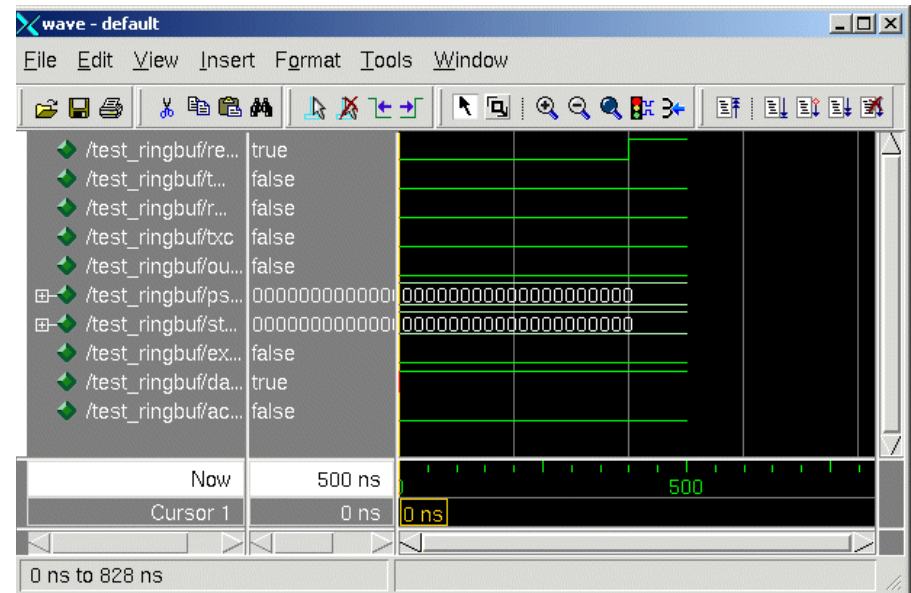**Figure 37: ModelSim steps into a function in a separate file**

Figure 38: SystemC primitive channels in the Wave window

Lesson Wrap-up

This concludes the lesson. Before continuing we need to quit the C debugger and end the current simulation.

1   Select **Tools > C Debug > Quit C Debug**.

2   Select **Simulate > End Simulation**. Click **Yes** when prompted to confirm that you wish to quit simulating.

Lesson 6 - Viewing simulations in the Wave window

Topics

The following topics are covered in this lesson:

Introduction

The Wave window allows you to view the results of your simulation as HDL waveforms and their values.

The Wave window is divided into a number of window panes (Figure 39). All window panes in the Wave window can be resized by clicking and dragging the bar between any two panes.

Related reading

ModelSim User's Manual – "Wave window" (UM-337), *Chapter 9 - WLF files (datasets) and virtuals* (UM-239)

Figure 39: The Wave window and its many panes
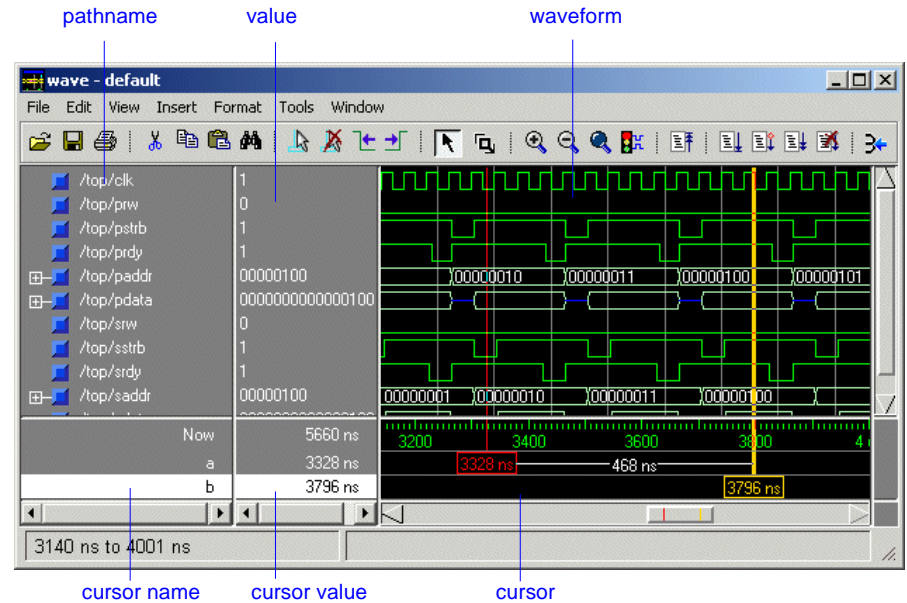
Loading a design

For the examples in this lesson, we have used the design simulated in *Chapter Lesson 2 - Basic simulation*.

1   If you just finished the previous lesson, ModelSim should already be running. If not, start ModelSim.

   a   Type **vsim** at a UNIX shell prompt or use the ModelSim icon in Windows.

     If the Welcome to ModelSim dialog appears, click **Close**.

2   Load the design.

   a   Select **File > Change Directory** and open the directory you created in Lesson 2.

     The *work* library should already exist.

   b   Click the '+' icon next to the *work* library and double-click *test_counter*.

     ModelSim loads the design and adds *sim* and *Files* tabs to the Workspace.

Adding items to the Wave window

ModelSim offers several methods for adding items to the Wave window. In this exercise, you'll try out different methods.

1   Add items from the Signals window.

   a   Select **View > Wave** to open the Wave window.

   b   Select **View > Signals** to open the Signals window.

   c   In the Signals window, select **Add > Wave > Signals in Design**.

       ModelSim adds three signals to the Wave window.

   d   In the Wave window, select **Edit > Select All** and then **Edit > Delete**.

       This deletes all items in the window.

2   Add items using drag-and-drop.

   You can drag an item to the Wave window from many other windows (e.g., Main, Signals, and Variables).

   a   Drag an instance from the *sim* tab of the Main window to the Wave.

       ModelSim adds the items for that instance to the Wave window.

   b   Drag a signal from the Signals window to the Wave window.

   c   In the Wave window, select **Edit > Select All** and then **Edit > Delete**.

3   Add items using a command.

   a   Type **add wave *** at the VSIM> prompt.

       ModelSim adds all items from the current region.

   b   Run the simulation for awhile so you can see waveforms.

Zooming the waveform display

Zooming lets you change the display range in the waveform pane. There are numerous methods for zooming the display.

1   Zoom the display using various techniques.

   a   Click the Zoom Mode icon on the Wave window toolbar.

   b   In the waveform pane, click and drag down and to the right.

      You should see blue vertical lines and numbers defining an area to zoom in (Figure 40).

   c   Select **View > Zoom > Zoom Last**.

      The waveform pane returns to the previous display range.

   d   Click the Zoom In 2x icon a few times.

   e   In the waveform pane, click and drag up and to the right.

      You should see a blue line and numbers defining an area to zoom out (Figure 41).

   f   Select **View > Zoom > Zoom Full**.

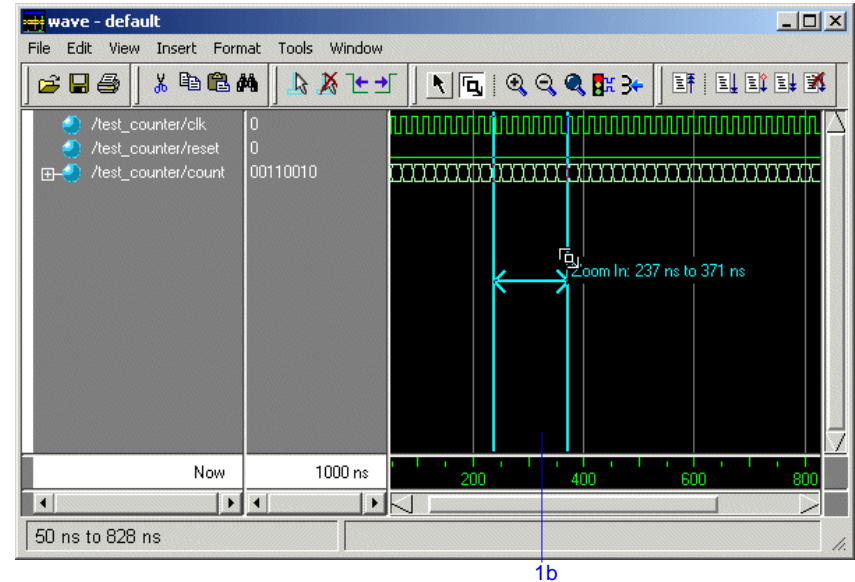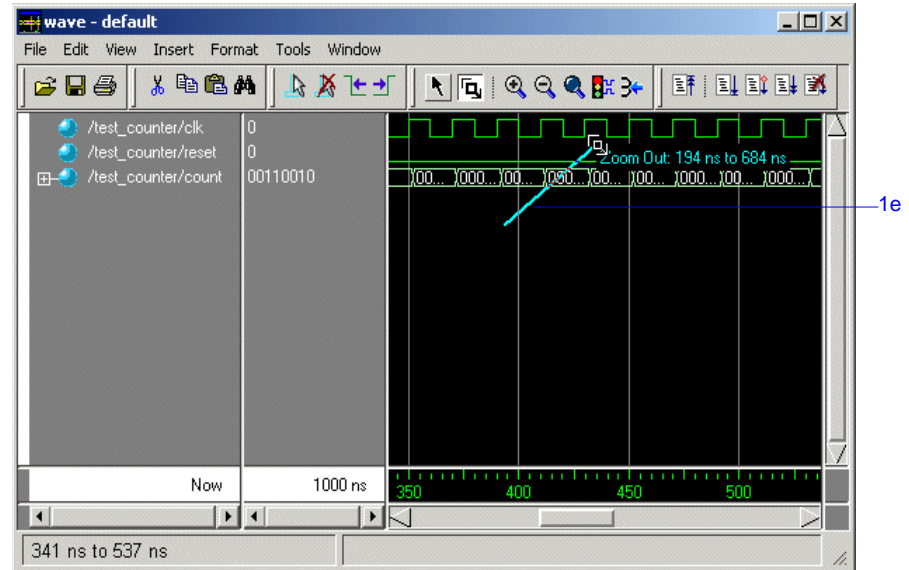**Figure 40: Zooming in with the mouse pointer**

Figure 41: Zooming out with the mouse pointer

Using cursors in the Wave window

Cursors mark simulation time in the Wave window. When ModelSim first draws the Wave window, it places one cursor at time zero. Clicking anywhere in the waveform pane brings that cursor to the mouse location.

You can also add additional cursors; name, lock, and delete cursors; use cursors to measure time interval; and use cursors to find transitions.

Working with a single cursor

1   Position the cursor by clicking and dragging.

   a   Click the Select Mode icon on the Wave window toolbar.

   b   Click anywhere in the waveform pane.

       A cursor is inserted at the time where you clicked (Figure 42).

   c   Drag the cursor and observe the value pane.

       The signal values change as you move the cursor. This is perhaps the easiest way to examine the value of a signal at a particular time.

   d   In the waveform pane, drag the cursor to the right of a transition with the mouse positioned over a waveform.

       The cursor "snaps" to the transition. Cursors "snap" to a waveform edge if you click or drag a cursor to within ten pixels of a waveform edge. You can set the snap distance in the Window Preferences dialog (select **Tools > Window Preferences**).

   e   In the cursor pane, drag the cursor to the right of a transition (Figure 42).

       The cursor doesn't snap to a transition if you drag in the cursor pane.

Figure 42: Working with a single cursor in the Wave window
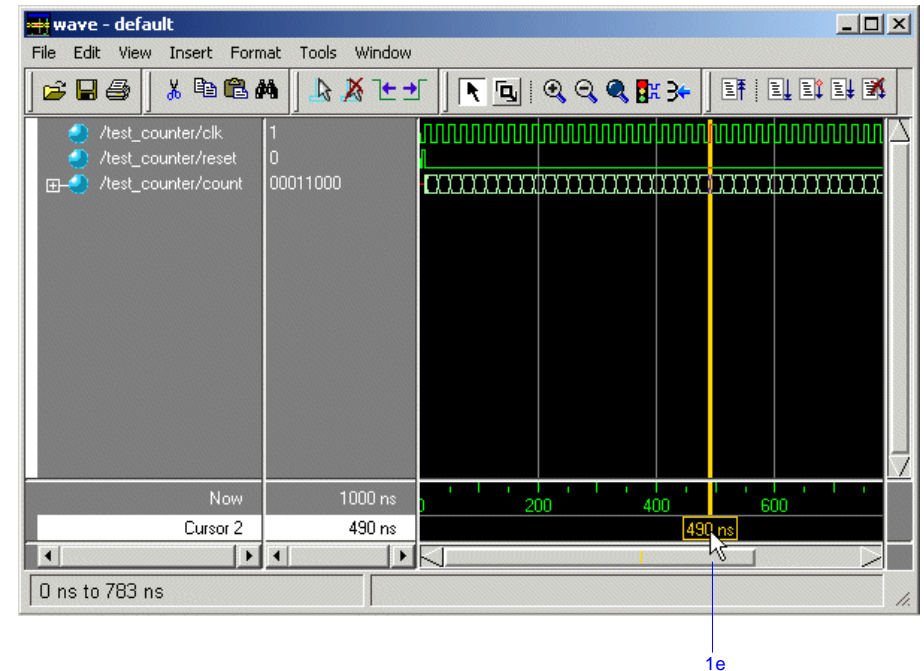
2   Rename the cursor.

    a   Right-click "Cursor 1" in the cursor name pane, and select and delete the text (Figure 43).

    b   Type **A** and press Enter.

       The cursor name changes to "A".

3   Jump the cursor to the next or previous transition.

    a   Click signal *count* in the pathname pane.

    a   Click the Find Next Transition icon on the Wave window toolbar.

       The cursor jumps to the next transition on the currently selected signal.

   b   Click the Find Previous Transition icon on the Wave window toolbar.

       The cursor jumps to the previous transition on the currently selected signal.

Working with multiple cursors

1   Add a second cursor.

    a   Click the Add Cursor icon on the Wave window toolbar.

    b   Right-click the name of the new cursor and delete the text.

    c   Type **B** and press Enter.

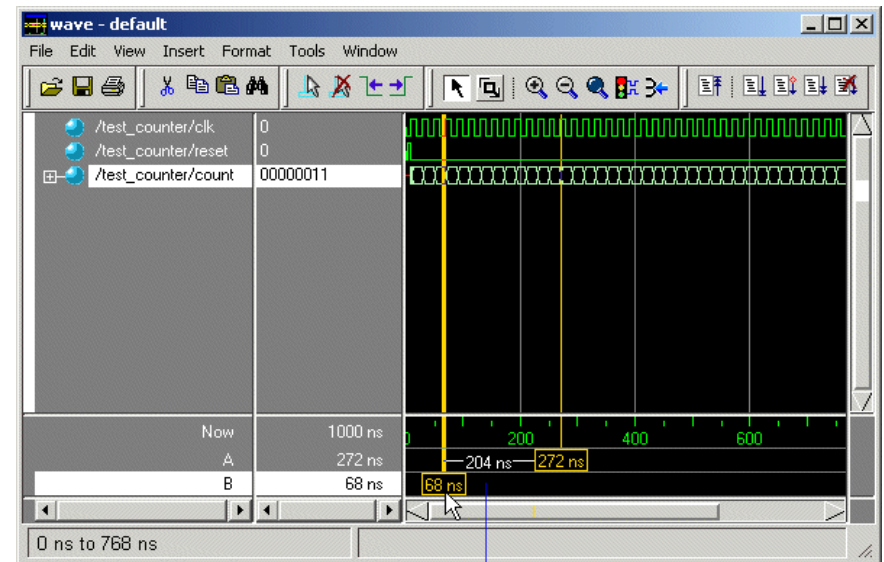   d   Drag cursor *B* and watch the interval measurement change dynamically (Figure 44).

Figure 43: Renaming a cursor
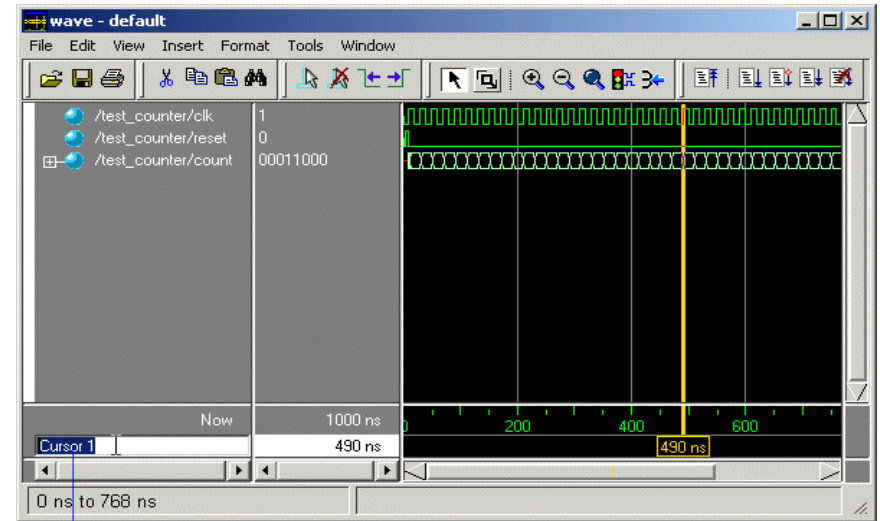
Figure 44: Interval measurement between two cursors

2 Lock cursor *B*.

    a With cursor *B* selected, select **Edit > Edit Cursor**.

    b Check **Lock cursor to specified time** and click OK (Figure 45).

       The cursor color changes to red and you can no longer drag the cursor (Figure 46).

3 Delete cursor *B*.

    a With cursor *B* selected, select **Edit > Delete Cursor**.
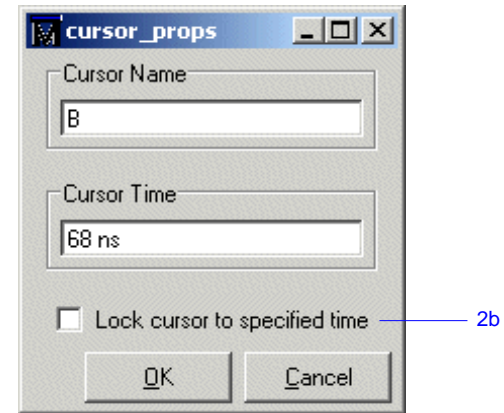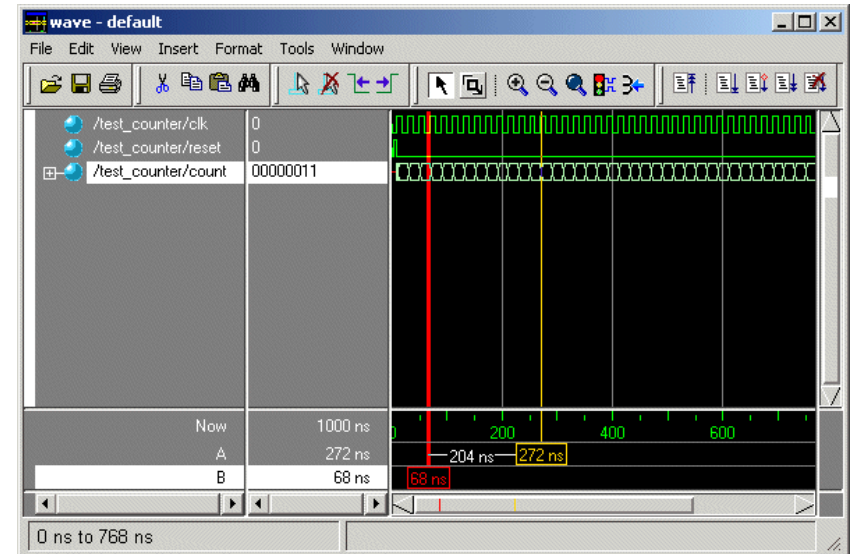
Figure 45: The cursor properties dialog

Figure 46: A locked cursor in the Wave window

Saving the window format

If you close the Wave window, any configurations you made to the window (e.g., signals added, cursors set, etc.) are discarded. However, you can use the Save Format command to capture the current Wave window display and signal preferences to a DO file. You open the DO file later to recreate the Wave window as it appeared when the file was created.

Format files are design-specific; use them only with the design you were simulating when they were created.

1   Save a format file.

   a   Select **File > Save > Format**.

   b   Leave the file name set to *wave.do* and click **Save**.

   c   Close the Wave window.

2   Load a format file.

   a   In the Main window, select **View > Wave**.

      All signals and cursor(s) that you had set are gone.

   b   In the Wave window, select **File > Open > Format**.

   c   Select *wave.do* and click **Open**.

      ModelSim restores the window to its previous state.

   d   Close the Wave window when you are finished by selecting **File > Close**.

Lesson wrap-up

This concludes this lesson. Before continuing we need to end the current
simulation.

1   Select **Simulate > End Simulation**. Click Yes.

Lesson 7 - Debugging with the Dataflow window

Topics

The following topics are covered in this lesson:

▶ **Note:** The functionality described in this tutorial requires a dataflow license feature in your ModelSim license file. Please contact your Mentor Graphics sales representative if you currently do not have such a feature.

Introduction

The Dataflow window allows you to explore the "physical" connectivity of your design; to trace events that propagate through the design; and to identify the cause of unexpected outputs. The window displays processes; signals, nets, and registers; and interconnect.

Design files for this lesson

The sample design for this lesson is a testbench that verifies a cache module and how it works with primary memory. A processor design unit provides read and write requests.

The pathnames to the files are as follows:

Verilog – *<install_dir>/modeltech/examples/dataflow/verilog*

VHDL – *<install_dir>/modeltech/examples/dataflow/vhdl*

This lesson uses the Verilog version in the examples. If you have a VHDL license, use the VHDL version instead. When necessary, we distinguish between the Verilog and VHDL versions of the design.

Related reading

ModelSim User's Manual – "Dataflow window" (UM-270)

Compiling and loading the example design

In this exercise you will use a DO file to compile and load the design.

1   Create a new directory and copy the tutorial files into it.

Start by creating a new directory for this exercise (in case other users will be working with these lessons). Create the directory and copy all files from *<install_dir>/examples/dataflow/verilog* to the new directory.

If you have a VHDL license, copy the files in *<install_dir>/examples/ dataflow/vhdl* instead.

2   Start ModelSim and change to the exercise directory.

If you just finished the previous lesson, ModelSim should already be running. If not, start ModelSim.

a   Type **vsim** at a UNIX shell prompt or use the ModelSim icon in Windows.

If the Welcome to ModelSim dialog appears, click **Close**.

b   Select **File > Change Directory** and change to the directory you created in step 1.

3   Execute the lesson DO file.

a   Type **do run.do** at the ModelSim> prompt.

The DO file does the following:

- Creates the working library
- Compiles the design files
- Opens the Dataflow, Signals, and Wave windows
- Adds signals to the Wave window
- Logs all signals in the design
- Runs the simulation

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

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 Add a signal to the Dataflow window.

    a   Make sure instance *p* is selected in the sim tab of the Main window.

    b   Drag signal *strb* from the Signals window to the Dataflow window (Figure 47).

2 Explore the design.

    a   Double-click the net highlighted in red.

        The view expands to display the processes that are connected to *strb* (Figure 48).

    b   Select signal *test* on process *#NAND#41* (labeled *line_62* in the VHDL version) and click the **Expand net to all drivers** icon.

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

    c   Select signal *oen* on process *#ALWAYS#146*(labeled *line_75* in the VHDL version), and click the **Expand net to all readers** icon.

Continue exploring if you wish. When you are done, click the **Erase All** icon.
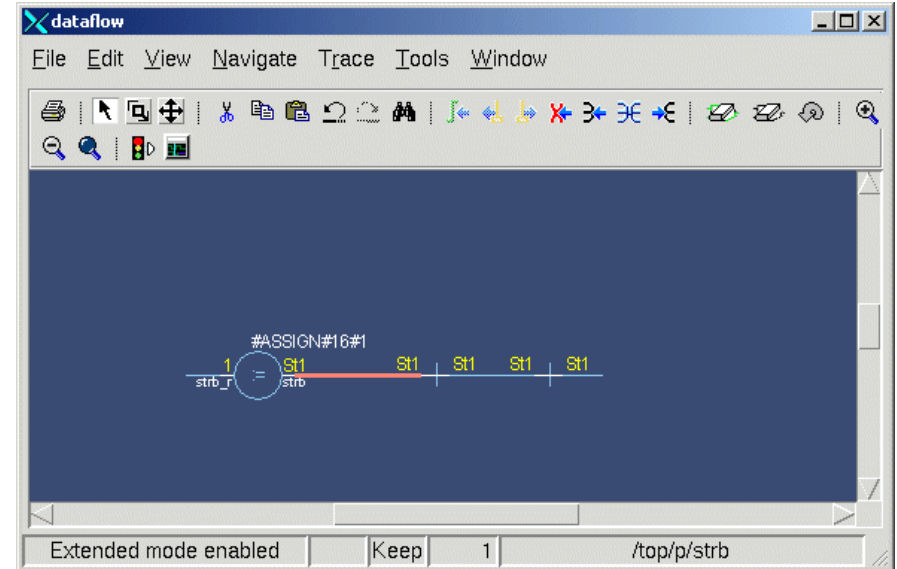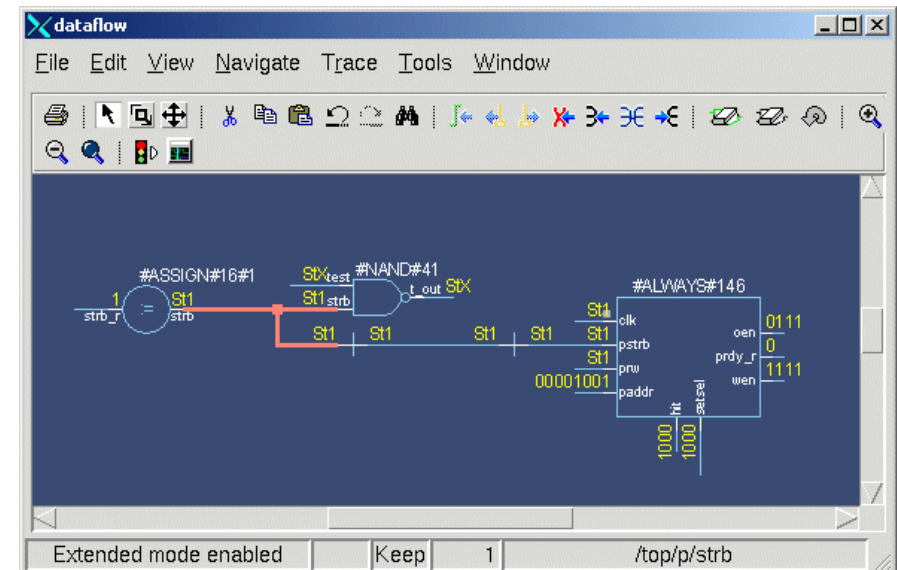
Figure 47: A signal in the Dataflow window

Figure 48: Expanding the view to display connected processes

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 which process or signal caused the
unexpected output.

1   Add an item to the Dataflow window.

    a   Make sure instance *p* is selected in the sim tab of the Main window.

    b   Drag signal *t_out* from the Signals window into the Dataflow window.

    c   Select **View > Show Wave** to open the wave viewer (Figure 49). You
       may need to increase the size of the Dataflow window and scroll the
       panes to see everything.

2   Trace the inputs of the nand gate.

    a   Select process *#NAND#41* (labeled *line_62* in the VHDL version) in the
       dataflow pane.

       All input and output signals of the process are displayed automatically in
       the wave viewer.

    b   In the wave view, scroll to time 2785 ns (the last transition of signal
       t_out) .

    c   Click on the last transition of signal *t_out* to set a cursor (Figure 50).
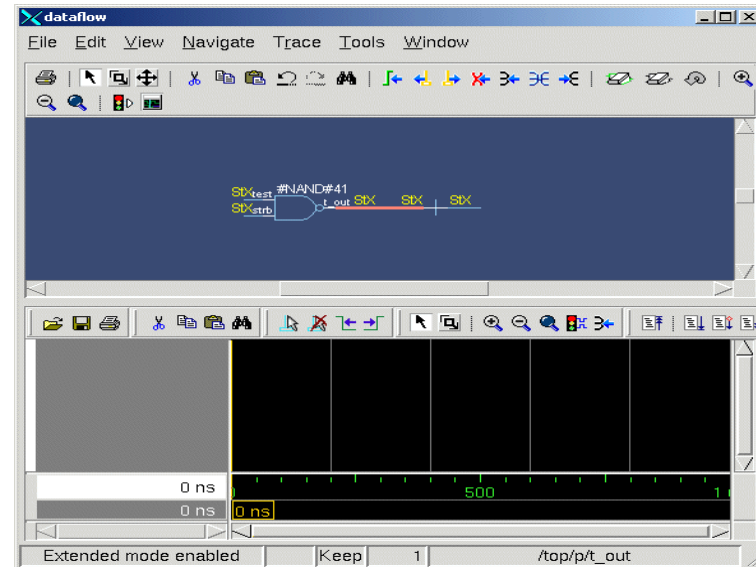
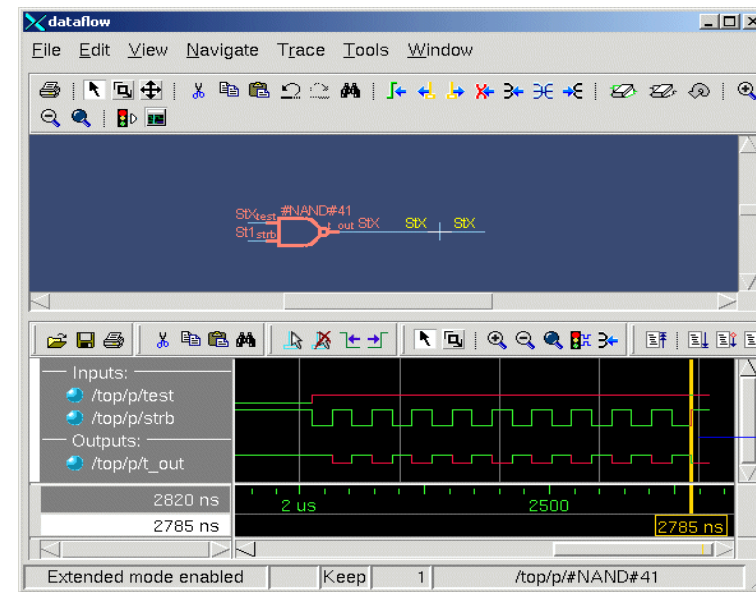**Figure 49: The embedded wave viewer pane**

Figure 50: Signals added to the wave viewer automatically

ModelSim SE Tutorial

d   Select **Trace > Trace next event** to trace the first contributing event.

ModelSim adds a cursor marking the last event, the transition of the strobe to 0 at 2745 ns, which caused the output of 0 on *t_out* (Figure 51).

e   Select **Trace > Trace next event** two more times.

f   Select **Trace > Trace event set**.

The dataflow pane sprouts to the preceding process and shows the input driver of signal *strb* (Figure 52). Notice too that the wave viewer now shows the input and output signals of the newly selected process.
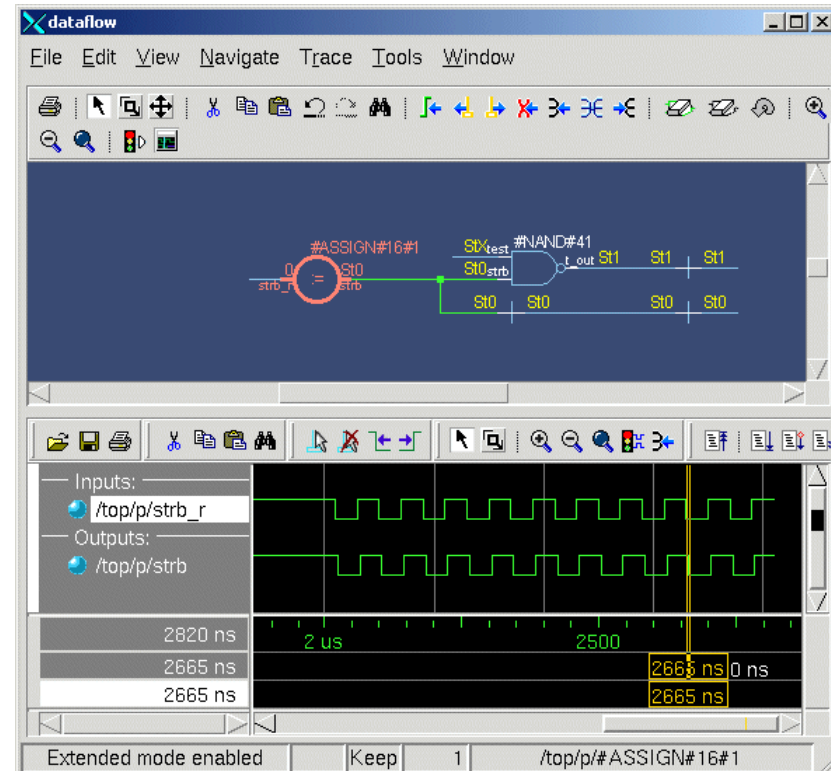
You can continue tracing events through the design in this manner: select **Trace next event** until you get to a transition of interest in the wave viewer, and then select **Trace event set** to update the dataflow pane.

3   Select **File > Close** to close the Dataflow window.

Figure 51: Cursor in wave viewer marking last event

Figure 52: Tracing the event set

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 are added automatically to the Wave window.

1   View *t_out* in the Wave and Dataflow windows.

    a   Scroll in the Wave window until you can see */top/p/t_out*.

        t_out goes to an unknown state at 2065 ns and continues transitioning between 1 and unknown for the rest of the run (Figure 53). The red color of the waveform indicates an unknown value.

    b   Double-click the last transition of signal *t_out* at 2785 ns.

        This automatically opens the Dataflow window and displays *t_out*, its associated process, and its waveform. You may need to increase the size of the Dataflow window and scroll the panes to see everything.

    c   Move the cursor in the Wave window.

        As previously mentioned the Wave and Dataflow windows are designed to work together. As you move the cursor in the Wave, the value of *t_out* changes in the Dataflow window.

    d   Move the cursor to a time when *t_out* is unknown (e.g., 2724 ns).

2   Trace the unknown.

    a   In the Dataflow window, make sure *t_out* is selected and then select **Trace > ChaseX**.

        The design expands to show the source of the unknown (Figure 54). In this case there is a HiZ (U in the VHDL version) on input signal *test_in* and a 0 on input signal_*rw* (*bar_rw* in the VHDL version), so output signal *test2* resolves to an unknown.

        Scroll to the bottom of the Wave window, and you will see that all of the signals contributing to the unknown value have been added.

3   Clear the Dataflow window before continuing.
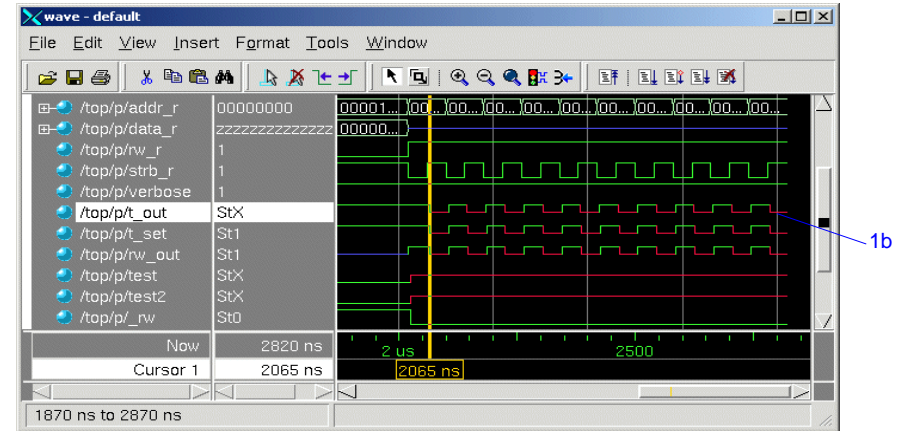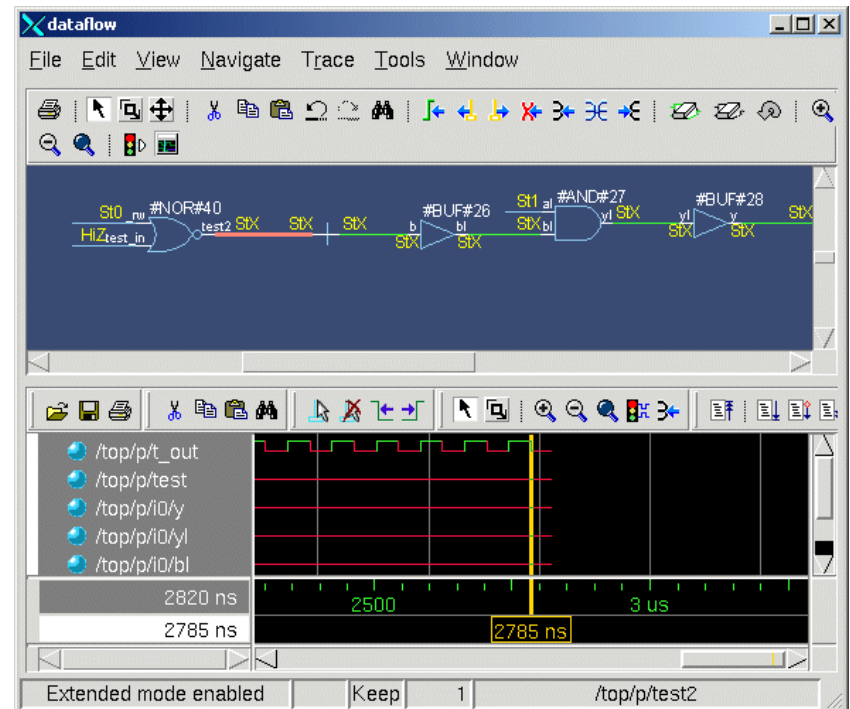
Figure 53: A signal with unknown values

Figure 54: ChaseX identifies the cause of the unknown on *t_out*

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   Change options to display hierarchy.

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

   b   Check **Show Hierarchy** and then click **OK** (Figure 55).

2   Add signal *t_out* to the Dataflow window.

   a   Type **add dataflow /top/p/t_out** at the VSIM> prompt (Figure 56).

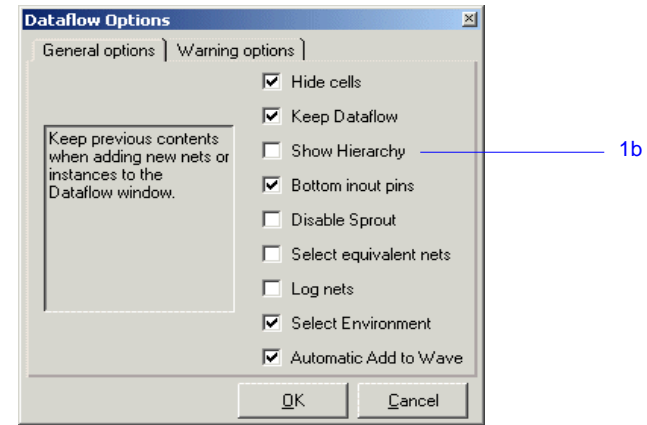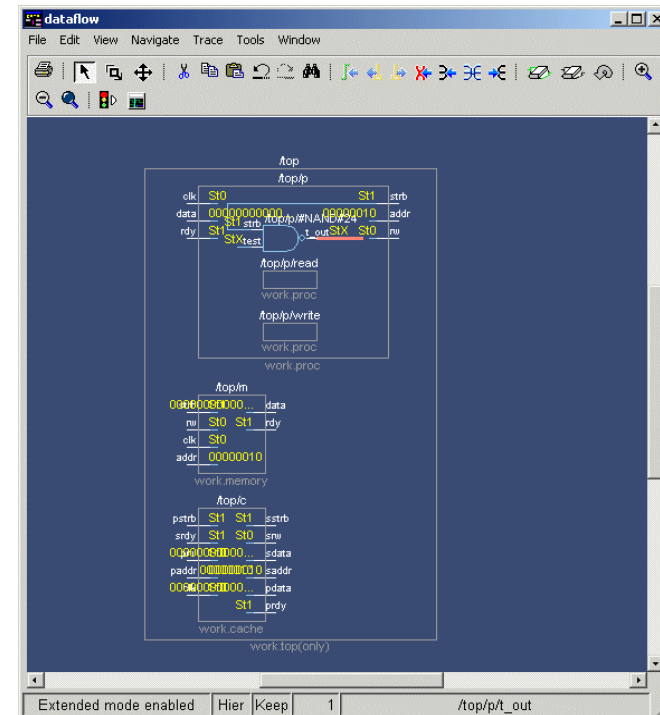**Figure 55: The Dataflow options dialog**

Figure 56: Dataflow window displaying with hierarchy

Lesson Wrap-up

This concludes this lesson. Before continuing we need to end the current simulation.

1   Type **quit -sim** at the VSIM> prompt.

Lesson 8 - Viewing and initializing memories

Topics

The following topics are covered in this lesson:

Introduction

In this lesson you will learn how to view and initialize memories in ModelSim.
ModelSim defines and lists as memories any of the following:

- reg, wire, and std_logic arrays

- Integer arrays

- Single dimensional arrays of VHDL enumerated types other than
  std_logic

▶ **Note:** This lesson uses the Verilog files *dp_syn_ram.v*, *ram_tb.v*, and
sp_syn_ram.v in the examples. If you are a VHDL user, use
dp_syn_ram.vhd, *ram_tb.vhd*, and *sp_syn_ram.vhd* instead.

Related reading

ModelSim User's Manual – "Memory window" (UM-302)

ModelSim Command Reference – **mem display** (CR-192)*,* **mem load** (CR-195),
mem save (CR-198) , **radix** (CR-235) commands

Compiling and loading the design

1   Create a new directory and copy the tutorial files into it.

   Start by creating a new directory for this exercise (in case other users will be working with these lessons). Create the directory and copy all files from *<install_dir>/examples/memory/verilog* to the new directory.

   If you have a VHDL license, copy the files in *<install_dir>/examples/ memory/vhdl* instead.

2   Start ModelSim and change to the exercise directory.

   If you just finished the previous lesson, ModelSim should already be running. If not, start ModelSim.

   a   Type **vsim** at a UNIX shell prompt or use the ModelSim icon in Windows.

      If the Welcome to ModelSim dialog appears, click **Close**.

   b   Select **File > Change Directory** and change to the directory you created in step 1.

3   Create the working library and compile the design.

   a   Type **vlib work** at the ModelSim> prompt.

   b   **Verilog**:
      Type **vlog sp_syn_ram.v dp_syn_ram.v ram_tb.v** at the ModelSim> prompt.

      VHDL:
      Type **vcom -93 sp_syn_ram.vhd dp_syn_ram.vhd ram_tb.vhd** at the ModelSim> prompt.

      Type **set NumericStdNoWarnings 1** at the ModelSim> prompt to suppress NumericStd warnings encountered during simulation.
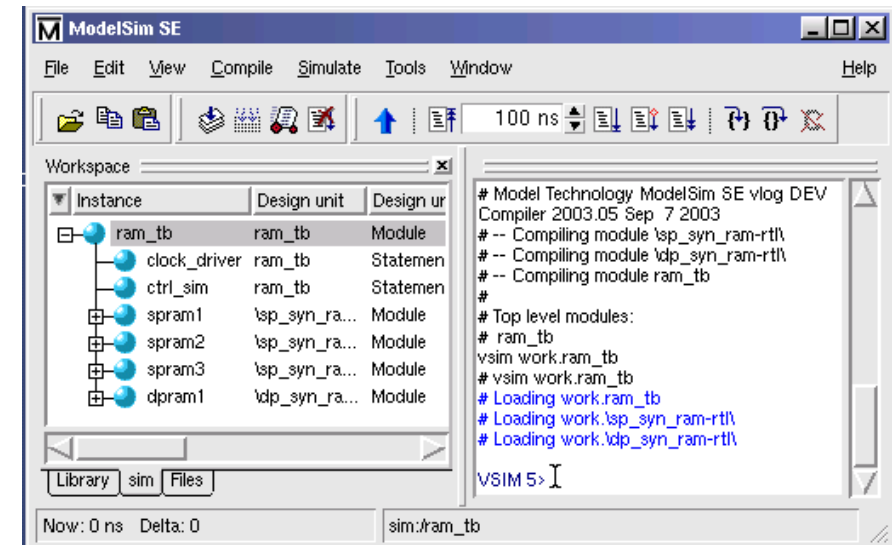
4   Load the design.

   a   On the Library tab of the Main window, click the "+" icon next to the
        work library.

   b   Double-click the *ram_tb* design unit (Figure 57). The design appears as
        shown in Figure 58.

Figure 57: Loading the memory testbench
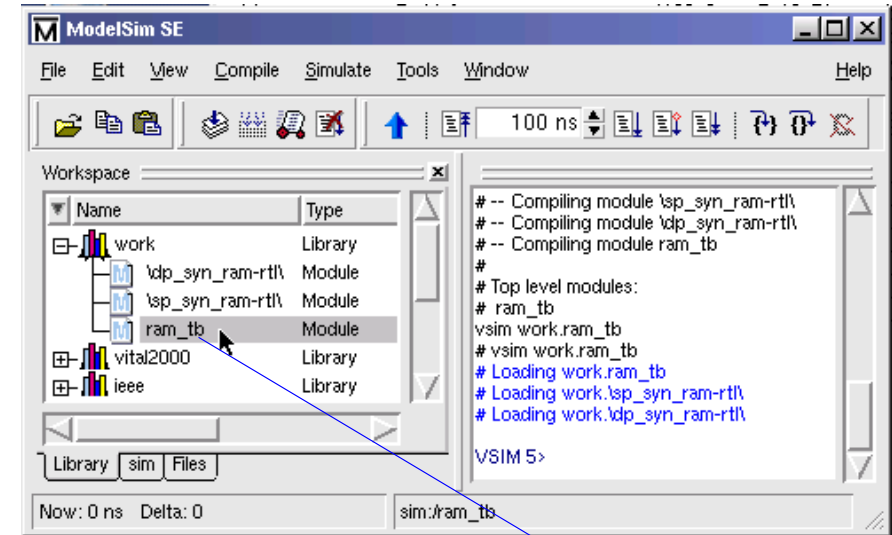
Figure 58: Loaded memory testbench

Viewing a memory

Memories can be viewed via the ModelSim GUI.

1   Open a Memory instance.

    a   Select **View > Memory** from the Main window menu bar to open the Memory window.

       The Memory List pane on the left side of the window lists the memories in the current design context (*ram_tb*).

    b   VHDL: The radix for enumerated types is Symbolic. To change the radix to binary for the purposes of this lesson, type the following command at the vsim prompt:
VSIM> **radix bin**

    c   Select the */ram_tb/spram1* instance in the Memory List to view its contents in the Address Data pane.

       The data are all **X** (**0** in VHDL) since you have not yet simulated the design (Figure 59).

    d   In the Main window "sim" tab (as shown in Figure 58 above), select instance *spram2*.

       The Memory List pane in the Memory window updates automatically to display *spram2* (Figure 60). However, the Address Data pane still shows the contents of instance *spram1*, because you have not yet opened the *spram2* memory instance.

       By default the Memory List updates dynamically to display memories from the current context (i.e., the current design unit). You can make the Memory List static by fixing it to a particular context. To do this you would select **File > Environment > Fix to current context**. For the purposes of this tutorial, you will leave the view dynamic.

Figure 59: Viewing the memory instance
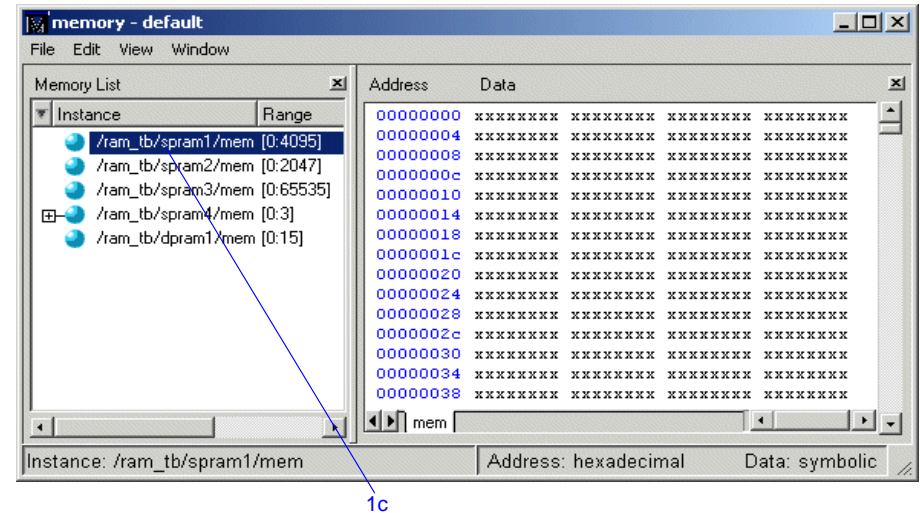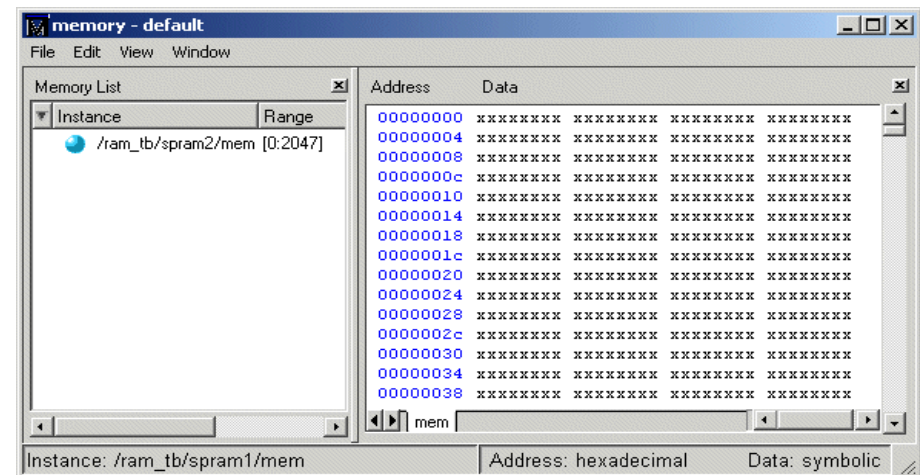

Figure 60: Memory List shows /ram_tb/spram2

2 Simulate the design.

    a   Click the **run -all** icon in the Main window.

        The Address Data pane updates to show values from instance *ram_tb/spram1* (Figure 61).

    VHDL:

    In the transcript window, you will see an assertion failure that is functioning to stop the simulation. The simulation itself has not failed.

You can open additional memory instances by selecting an instance in either the Main window, Signals window, or Structure window, and dragging and dropping it into the Memory window.

3 Open a second memory instance from the Main window.

    a   In the Main window, drag and drop *spram2* into the Address Data pane of the Memory window.

        The contents of *spram2* is displayed, and a new tab is added to the bottom of the Address Data pane (Figure 62). The Memory List now displays only the */ram_tb/spram2* instance.

    b   Reset the Memory List view to *ram_tb*. Click on the *ram_tb* module in the Main window "sim" tab.

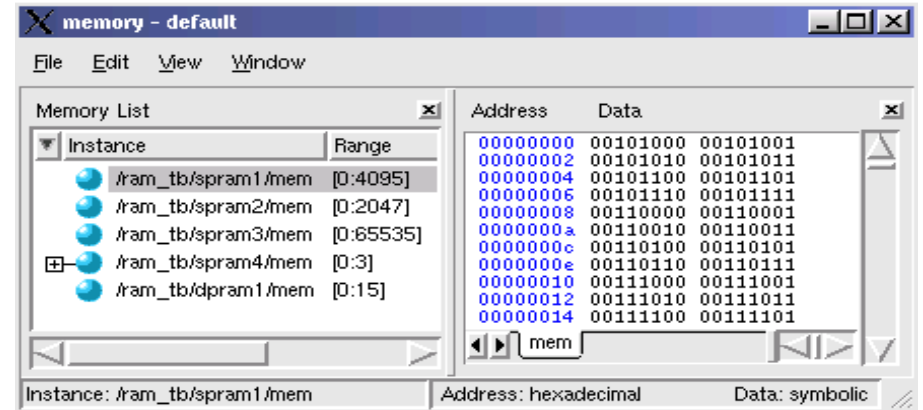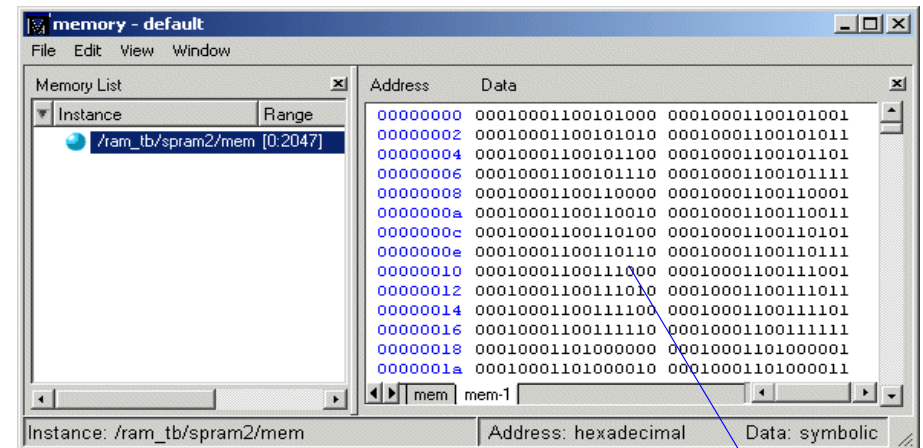**Figure 61: Memory display updates with simulation**

Figure 62: View of spram2 data dragged from Main window

4   Let's change the address radix and the number of words per line for the
     ram_tb/spram1 memory instance.

   a   Select the *mem* tab at the bottom of the Address Data pane to view the
        ram_tb/spram1 instance.

   b   Select **View > Display options** to bring up the dialog box (Figure 63).

   c   For the **Address Radix, s**elect **Decimal**.

   d   Select **Words per line** and type **1** in the field.

   e   Click OK.

   You can see the results of the settings in Figure 64.
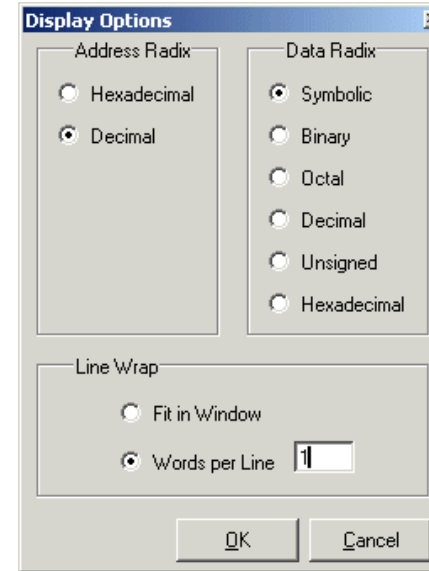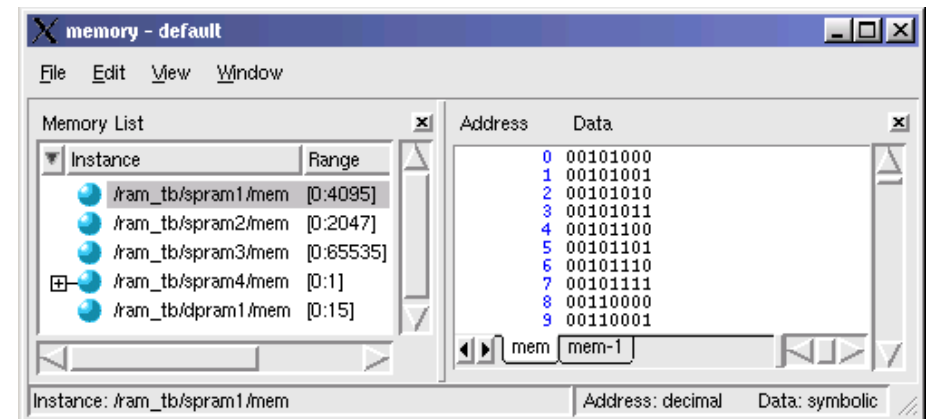
Figure 63: Display Options dialog box

Figure 64: Memory window: new address radix and line length

Navigating within the memory

You can navigate to specific memory address locations, or to locations containing particular data patterns. First, you will go to a specific address.

1 Use Goto to find a specific address.

   a   If necessary, click on the */ram_tb/spram1* in the Memory list pane.

   b   Select **Edit > Goto** from the Memory menu.

   The Goto dialog box opens in the data pane (Figure 65).

   c   Type **12** in the dialog box.

   d   Click OK.

   The requested address appears in the top line of the Address Data pane in the Memory window (Figure 65).

2 Edit the address location directly.

   To quickly move to a particular address, do the following:

   a   Double click any address in the Address Data pane of the Memory window (Figure 66)

   b   Enter any desired address.

   c   Press <Enter> on your keyboard.

   The Address Data pane scrolls to that address.

Figure 65: The Goto dialog box

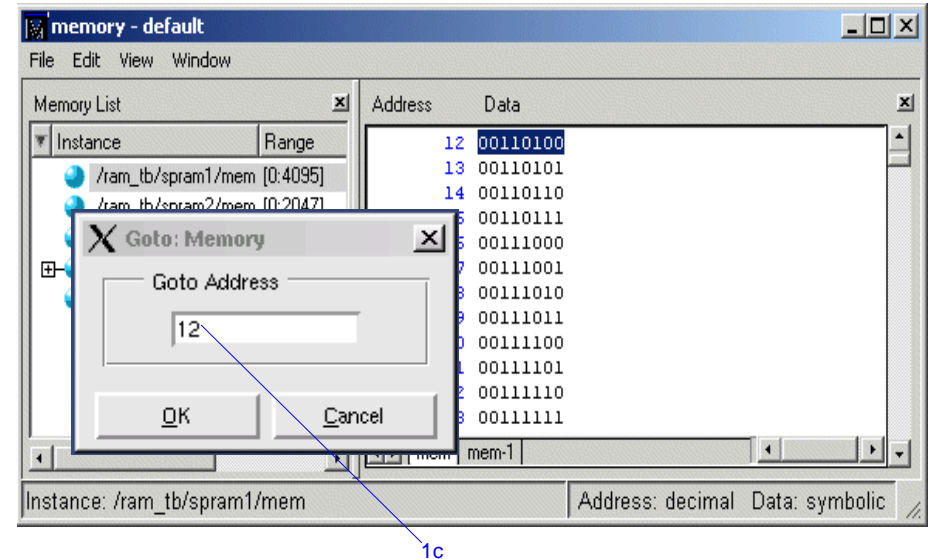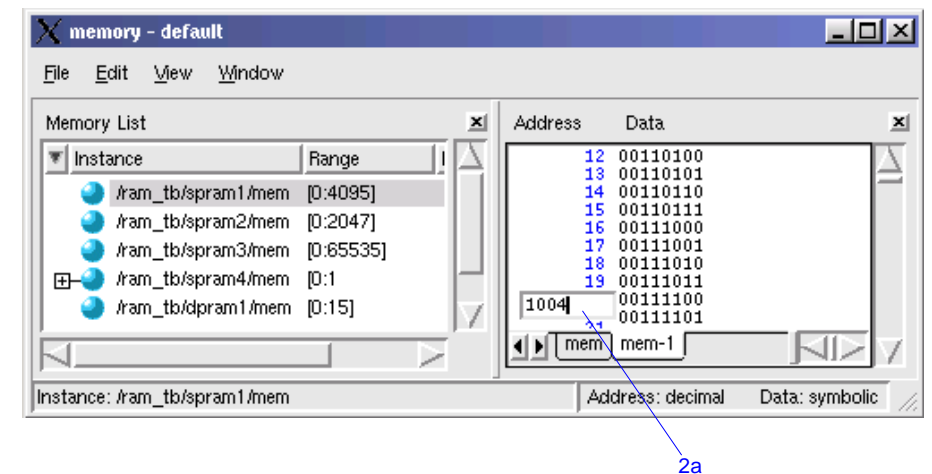**Figure 66: Edit the address directly**

3   Let's search for a particular data entry now.

    a   Select **Edit > Data Search** from the Memory window menu bar.

        The Data Search in Memory dialog box opens in the data pane (Figure 67).

    b   Type **11111010** in the **Search for:** field and click **Search Next**.

        The Address Data pane scrolls to the first occurrence of that address (Figure 68). Click Search Next a few more times to search through the list.

    c   Click Close to close the dialog box.

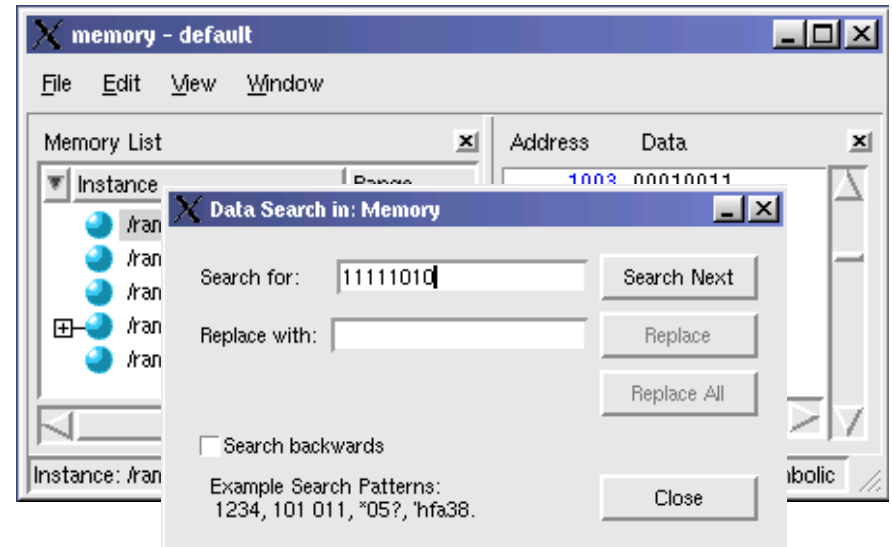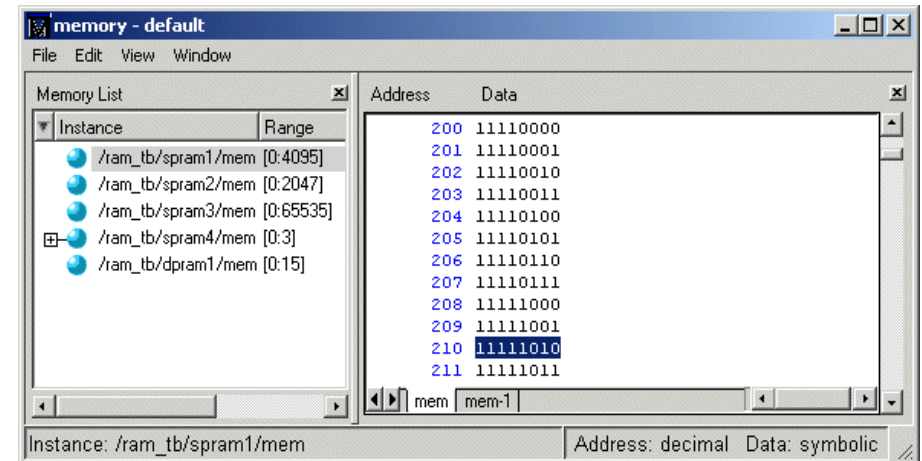**Figure 67: Find: searching for data value**

Figure 68: Data value found

Saving memory contents to a file

Figure 69: Save Memory dialog box

You can save memory contents to a file that can be loaded at some later point in simulation.

1   Save a memory pattern from the *ram_tb/spram1* instance to a file.

   a   Select **File > Save** in the Memory window to bring up the Save Memory dialog box (Figure 69).

       Note that **MTI** is the default File Format.

   b   For the Address Radix, select **Decimal**.

   c   For the Data Radix, select **Binary**.

   d   Type **data_mem.mem** into the Filename field.

   e   Click OK.

   You can view the saved file in any editor.

Memory pattern files can be saved as relocatable files, simply by leaving out the address information. Relocatable memory files can be loaded anywhere in a memory because no addresses are specified.

2   Save a relocatable memory pattern file.

    a   Click on the *ram_tb/spram2* instance in the Memory List pane.

    b   Select **View > Display Options** to set the Address Radix of this file to Decimal. Click OK

    c   Select **File > Save** to bring up the Save Memory dialog box (Figure 70).

    d   Specify start of address as **0** and end address as **250.**

    e   For Address Radix select Decimal, and for Data Radix select Binary.

    f   Click **No addresses** to create a memory pattern that you can use to relocate somewhere else in the memory, or in another memory.

    g   Enter the file name as **reloc.mem.**

    h   Click OK

You will use this file for initialization in the next section.

Figure 70: Saving a relocatable memory file

Initializing a memory

In ModelSim, it is possible to initialize a memory using one of three methods: from a saved memory file, from a fill pattern, or from both.

First, let's initialize a memory from a file only. You will use one you saved previously, *data_mem.mem*.

1   View instance *ram_tb/spram3*.

    a   Click on the *ram_tb/spram3* instance in the Memory window.

        Scan the contents so you can identify changes once the initialization is complete.

    b   Select **View > Display Options** to bring up the Display Options dialog.

    c   Change the Address Radix to Decimal and click OK.

2   Initialize *spram3* from a file.

    a   Select **File > Load** to bring up the Load Memory dialog box (Figure 71).

        The default Load Type is File Only.

    b   Select *data_mem.mem* file in the Files list.

    c   Click OK.

The addresses in instance *ram_tb/spram3* are updated with the data from *data_mem.mem* (Figure 72).

Figure 71: Load Memory dialog box

Figure 72: Initialized memory from file and fill pattern

In this next step, you will experiment with loading from both a file and a fill pattern. You will initialize *spram3* with the 250 addresses of data you saved previously into the relocatable file *reloc.mem*. You will also initialize 50 additional address entries with a fill pattern.

3   Load the *ram_tb/spram3* instance with a relocatable memory pattern (*reloc.mem*) and a fill pattern.

  a   Select **File > Load** to bring up the Load Memory dialog box (Figure 73).

  b   For Load Type, select **Both File and Data**.

  c   For File Load, select *reloc.mem* from the Files list.

  d   Select **Addresses** and enter **0** as the start address and **300** as the end address.

      This means that you will be loading the file from 0 to 300. However, the *reloc.mem* file contains only 251 addresses of data. Addresses 251 to 300 will be loaded with the fill data you specify next.

  e   For Fill Type, select **Increment**.

  f   In the Fill Data field, set the seed value of **0** for the incrementing data.

  g   Click OK.

  h   View the data near address 250 by double-clicking on any address in the Address column and entering **250**.

You can see the specified range of addresses overwritten with the new data. Also, you can see the incrementing data beginning at address 251 (Figure 74).

Now, before you leave this section, go ahead and clear the instances already being viewed.

4   Right click in the Address Data pane and select **Close All**.
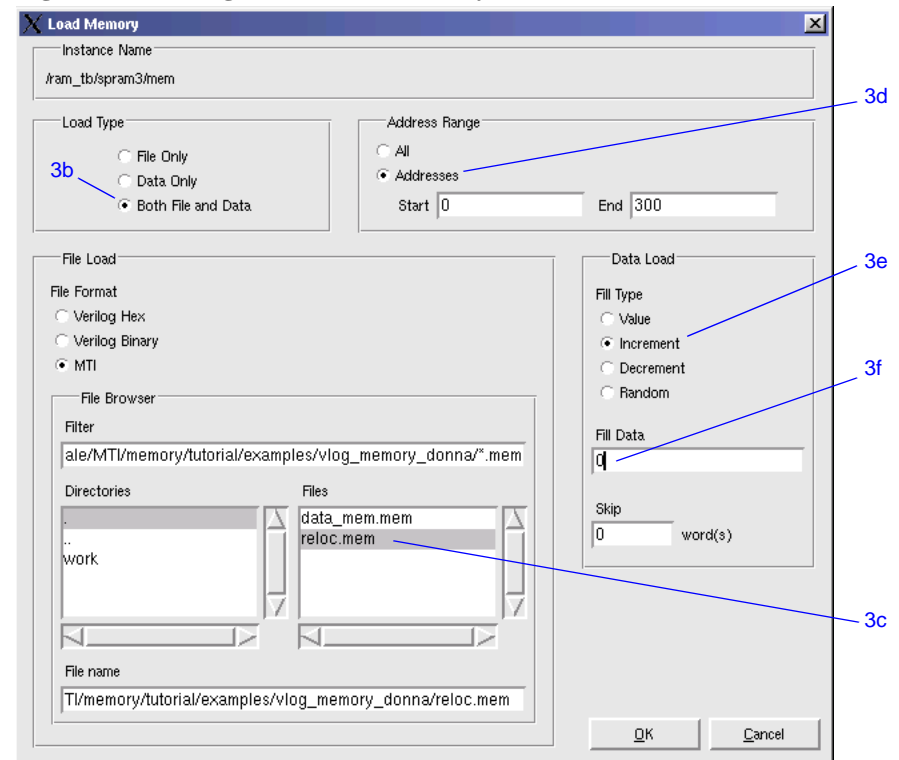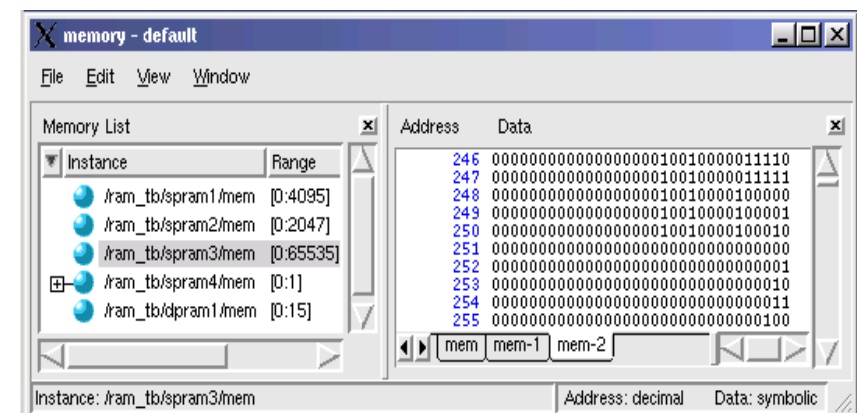
Figure 73: Loading a relocatable memory file

Figure 74: Overwritten values in memory file

Interactive debugging commands

The memory window can be used interactively for a variety of debugging purposes. The features described in this section are useful for this purpose.

1 Open a memory instance and change its display characteristics.

   a Click on the *ram_tb/dpram1* instance in the Memory window.

   b Select **View > Display Options** to bring up the dialog box.

   c Change the Data Radix to **Hexadecimal**.

   d Select **Words per line** and enter **2**.

   e Click OK.

2 Initialize a range of memory addresses from a fill pattern.

   a Select **Edit > Change** from the Memory window menu bar (Figure 76).

   b Click the **Addresses** radio button and enter the start address as **0x00000006** and the end address as **0x00000009**. The "0x" hex notation is optional.

   c Select **Random** as the **Fill Type**.

   d Enter **0** as the **Fill Data**, setting the seed for the Random pattern.

   e Click OK.

   The data in the specified range are replaced with a generated random fill pattern (Figure 77).
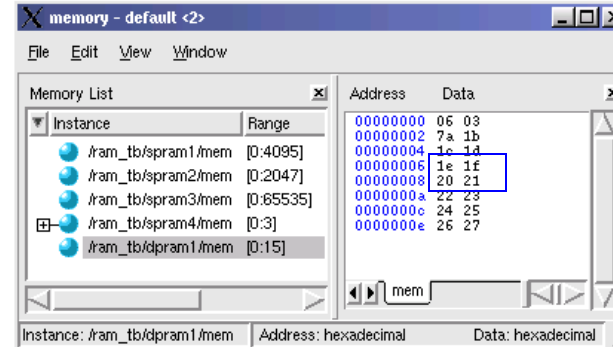
Figure 75: Original memory contents

Figure 76: Changing memory contents for a range of addresses
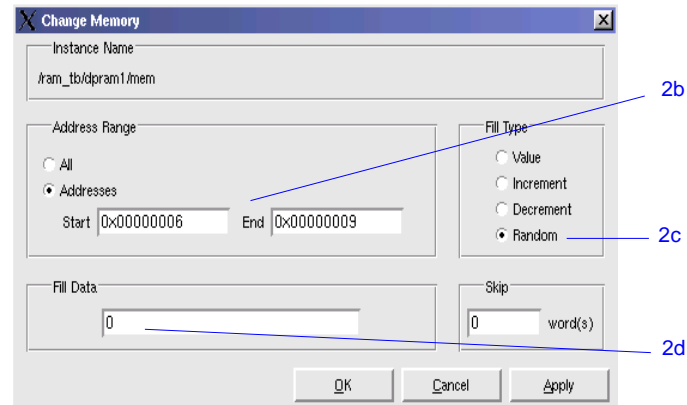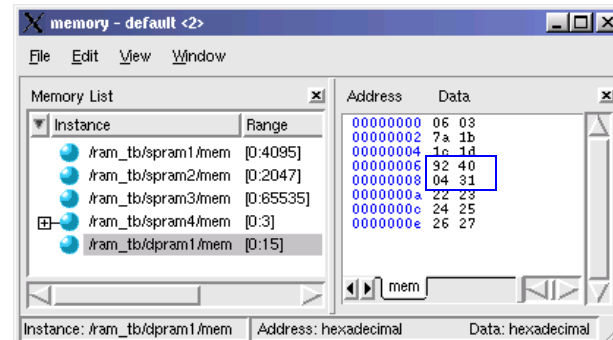

Figure 77: Random contents of a range of addresses

3   Change contents by highlighting.

You can also change data by highlighting them in the Address Data pane.

a   Highlight the data for the addresses **0x0000000c:0x0000000e**, as shown in Figure 78.

b   Right click the highlighted data and select **Change**.

This brings up the Change dialog box (Figure 79). Note that the Addresses field is already populated with the range you highlighted.

c   Select **Value** as the Fill Type.

d   Enter the data values into the Fill Data field as follow: **34 35 36**

e   Click OK.

The data in the address locations change to the values you entered (Figure 80).

4   Edit data in place.

To edit only one value at a time, do the following:

a   Double click any value in the Data column.

b   Enter the desired value.

c   Double-click on another value to quickly save the previously edited value and begin editing a new value.

d   When you are finished editing all values, press the <Enter> key on your keyboard to exit the editing mode.

If you needed to cancel the edit function, press the <Esc> key on your keyboard.
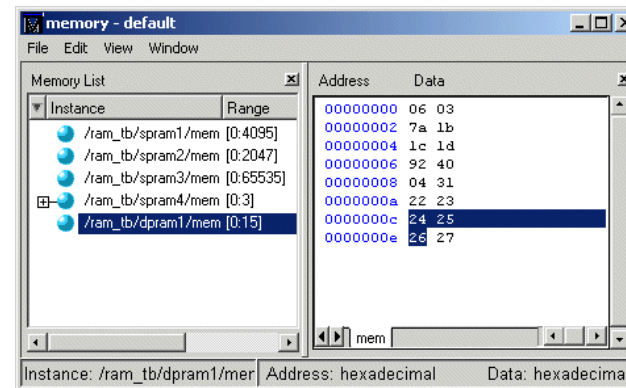
Figure 78: Changing contents by highlighting

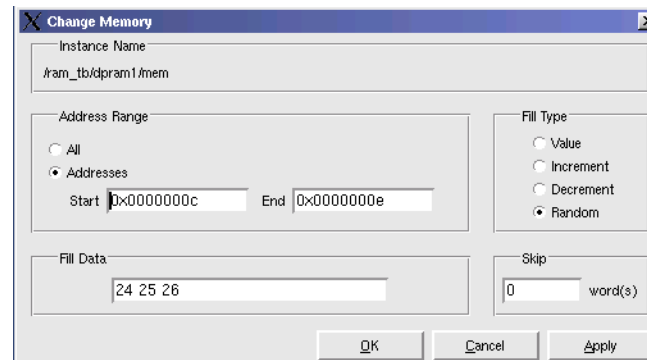

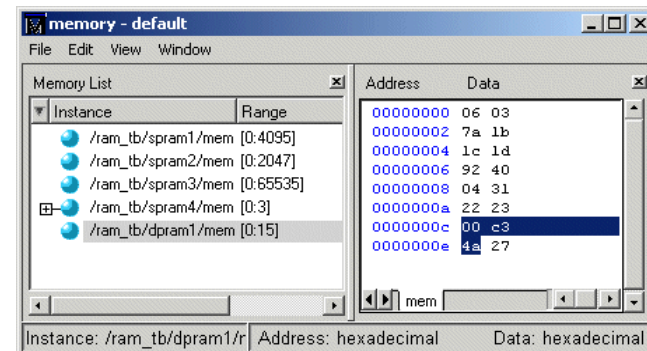**Figure 79: Entering data to change**

Figure 80: Changed contents for specified addresses

Lesson Wrap-up

This concludes this lesson. Before continuing we need to end the current simulation.

1   Select **Simulate > End Simulation**. Click Yes.

Lesson 9 - Simulating with Performance Analyzer

Topics

The following topics are covered in this lesson:

▶ **Note:** The functionality described in this tutorial requires ModelSim SE and a profile
license feature in your ModelSim license file. Please contact your Mentor Graphics sales
representatives for more information.

Introduction

The 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 the Performance Analyzer.

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

Design files for this lesson

The sample design for this lesson consists of a finite state machine which controls a behavioral memory. The testbench *test_sm* provides stimulus.

The ModelSim installation comes with Verilog and VHDL versions of this design. The files are located in the following directories:

Verilog – *<install_dir>/modeltech/examples/profiler/verilog*

VHDL – *<install_dir>/modeltech/examples/profiler/vhdl*

This lesson uses the Verilog version for the examples. If you have a VHDL license, use the VHDL version instead.

Related reading

ModelSim User's Manual – Chapter 11 - Performance Analyzer (UM-407),
Chapter 21 - Tcl and macros (DO files) (UM-591)

Compiling and loading the example design

In this exercise you will use a DO file to compile and load the design.

1   Create a new directory and copy the tutorial files into it.

   Start by creating a new directory for this exercise (in case other users will be
   working with these lessons). Create the directory and copy all files from
   <install_dir>/examples/profiler/verilog to the new directory.

   If you have a VHDL license, copy the files in *<install_dir>/examples/*
   profiler/vhdl instead.

2   Start ModelSim and change to the exercise directory.

   If you just finished the previous lesson, ModelSim should already be running.
   If not, start ModelSim.

   a   Type **vsim** at a UNIX shell prompt or use the ModelSim icon in
      Windows.

      If the Welcome to ModelSim dialog appears, click **Close**.

   b   Select **File > Change Directory** and change to the directory you created
      in step 1.

3   Execute the lesson DO file.

   a   Type **do run.do** at the ModelSim> prompt.

      The DO file does the following:

      •   Creates the working library

      •   Compiles the design files

      •   Loads the *test_sm* design unit

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

Running the simulation

Throughout this lesson you will run the simulation via a DO file. DO files are macros you create that automatically run several ModelSim commands. The DO file in this lesson uses the **seconds** Tcl command to time each simulation run. Feel free to open the DO file and look at its contents.

1   Enable the Performance Analyzer.

    a   Select **Tools > Profile > Profile On**.

       This must be done prior to running the simulation. ModelSim is now ready to collect data on the run.

2   Run the simulation via the DO file.

    a   Type **do profile_run.do** at the VSIM> prompt.

       The status bar at the bottom of the Main window reports the number of Profile Samples taken as the simulation runs.

       Make note of the run time reported in the Transcript (Figure 81). You will use it later to compare how much you have increased simulation speed by tweaking the design. (Your times may differ from those shown here due to differing system configurations.)

Figure 81: Note the run time reported in the Transcript
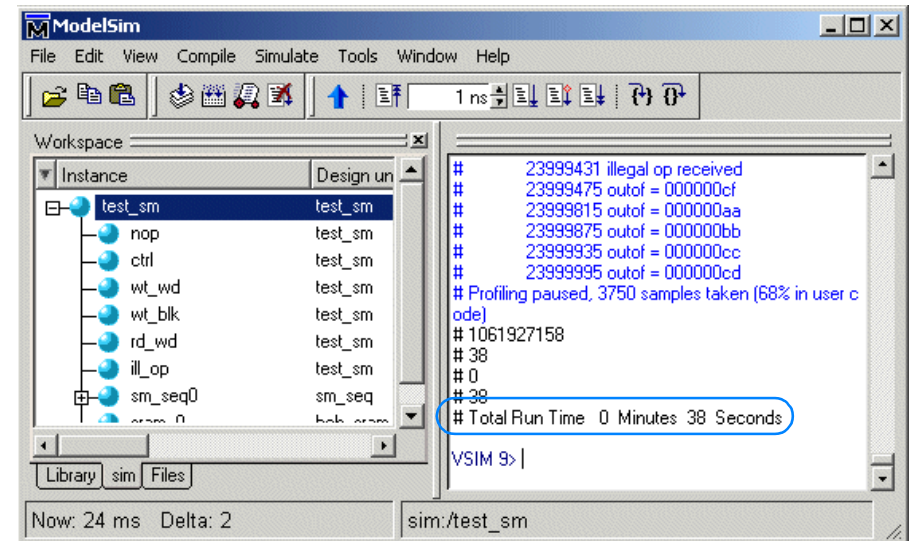
3   Display the performance data.

   a   Select **Tools > Profile > View hierarchical profile**.

      This displays the Hierarchical Profile window (Figure 82).

      The results differ between the Verilog and VHDL versions of the design. In Verilog, line 96 (*test_sm.v:96*) is taking the majority of simulation time. In VHDL, lines 192 and 84 are taking the majority of the time.

      ▶ **Note:** Your results may look slightly different as a result of the computer you're using and different system calls that occur during the simulation. Also, the line number reported in the Hierarchical Profile may be one or two lines off the actual source file. This happens due to how the stacktrace is decoded on different platforms.

   b   In the Hierarchical Profile window, click on one of the lines that is taking a lot of time.

      The Source window opens with that line number highlighted in the source code (Figure 83).

Figure 82: The Hierarchical Profile window

Figure 83: The Source window showing a line from the profile data

Using the data to improve performance

The information provided by the Performance Analyzer can be used to speed up the simulation. In this example, the repeated printing of data values to the screen is a significant burden to simulation. A more efficient approach would be to print only fail messages when they occur and a single pass message at the end of a data block or the entire simulation run.

1   Edit the source code to remove the repeated screen printing.

   a   Uncheck **Edit > read only** (Source window) to make the file editable.

   b   "Comment out" the repeated screen printing.

     In Verilog change lines 96-97 so they look like this:

```
//always @ (outof)  // any change of outof
//   $display ($time,,"outof = %h",outof);
```

     In VHDL change lines 189-192 so they look like this:

```
--   write(msg_line,NOW,field=>10);
--   write(msg_line,msg1);
--   hwrite(msg_line,rd_data);
--   writeline(OUTPUT,msg_line);
```

2   Save the file and re-compile.

   a   Select **File > Save** (Source window).

   b   Recompile the file.

     For Verilog – Type **vlog test_sm.v** at the VSIM> prompt.

     For VHDL – Type **vcom test_sm.vhd** at the VSIM> prompt.

3   Re-start and re-run the design.

   a   Type **restart -f** at the VSIM prompt.

   b   Type **do profile_run.do** at the VSIM> prompt.

     The simulation time is reduced by almost 50% (Figure 84).

Figure 84: Simulation time reduced by almost 50%
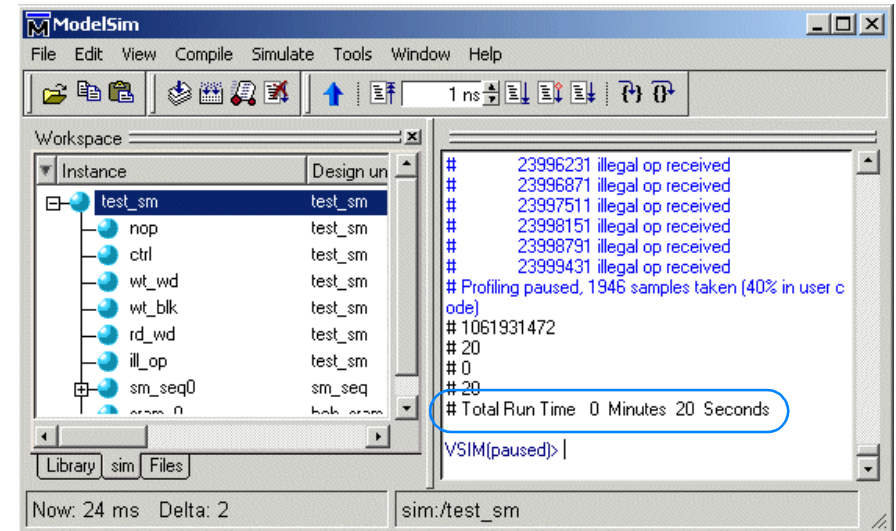
4   Look at the performance data again.

a   Select **Tools > Profile > View hierarchical profile**.

The problem with repeated screen printing has been removed (Figure 85).

Figure 85: Source edit removes the performance bottleneck

Filtering and saving the data

As a last step, you will filter out lines that take less than 2% of the simulation time and then save the report data to a text file.

1 Filter lines that take less than 2% of the simulation time.

a Change the **Under %** field to 2 (Figure 86).

b Click the **Update Data** button.

ModelSim filters the list to show only those lines that take 2% or more of the simulation time.

2 Save the report.

a Click the save icon in the Hierarchical Profile window.

b Type **hier.rpt** in the **File name** field and then click **Save**.

If you want, open the report file with an editor to see what is saved. You can also output this report from the command line using the **profile report** command. See the *ModelSim Command Reference* for details.

Figure 86: The filtered profile data
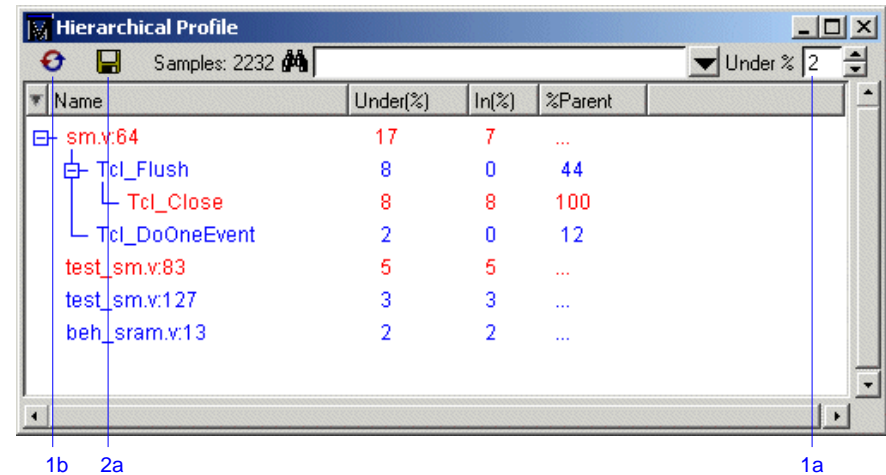
Lesson wrap-up

This concludes this lesson. Before continuing we need to end the current simulation.

1   Select **Simulate > End Simulation**. Click Yes.

Lesson 10 - Simulating with Code Coverage

Topics

The following topics are covered in this lesson:

▶ **Note:** The functionality described in this tutorial requires a coverage license feature in your ModelSim license file. Please contact your Mentor Graphics sales representative if you currently do not have such a feature.

Introduction

ModelSim Code Coverage gives you graphical and report file feedback on which executable statements, branches, conditions, and expressions in your source code have been executed. It also measures bits of logic that have been toggled during execution.

Design files for this lesson

The sample design for this lesson consists of a finite state machine which controls a behavioral memory. The testbench *test_sm* provides stimulus.

The ModelSim installation comes with Verilog and VHDL versions of this design. The files are located in the following directories:

Verilog – *<install_dir>/modeltech/examples/coverage/verilog*

VHDL – *<install_dir>/modeltech/examples/coverage/vhdl*

This lesson uses the Verilog version in the examples. If you have a VHDL license, use the VHDL version instead. When necessary, we distinguish between the Verilog and VHDL versions of the design.

Related reading

ModelSim User's Manual – Chapter 12 - Code Coverage (UM-419)

Compiling and loading the design

Enabling Code Coverage is a two step process–first, you compile the files and identify which coverage statistics you want; second, you load the design and tell ModelSim to produce those statistics.

1   Create a new directory and copy the tutorial files into it.

    Start by creating a new directory for this exercise (in case other users will be working with these lessons). Create the directory and copy all files from *<install_dir>/modeltech/examples/coverage/verilog* to the new directory.

    If you have a VHDL license, copy the files in *<install_dir>/modeltech/ examples/coverage/vhdl* instead.

2   Start ModelSim and change to the exercise directory.

    If you just finished the previous lesson, ModelSim should already be running. If not, start ModelSim.

    a   Type **vsim** at a UNIX shell prompt or use the ModelSim icon in Windows.

        If the Welcome to ModelSim dialog appears, click **Close**.

    b   Select **File > Change Directory** and change to the directory you created in step 1.

3   Create the working library.

    a   Type **vlib work** at the ModelSim> prompt.

4   Compile the design files.

    a   For Verilog – Type **vlog -cover bct sm.v sm_seq.v beh_sram.v test_sm.v** at the ModelSim> prompt.

        For VHDL – Type **vcom -cover bct sm.vhd sm_seq.vhd sm_sram.vhd test_sm.vhd** at the ModelSim> prompt.

        The **-cover bct** argument instructs ModelSim that you want branch, condition, and toggle coverage statistics. See the *Code Coverage* chapter in the *ModelSim User's Manual* for more information on the available coverage types.

5   Load the design and run.

a   Type **vsim -coverage test_sm** at the ModelSim> prompt.

b   Type **run 1 ms** at the VSIM> prompt.

When you load a design with Code Coverage enabled, ModelSim adds several columns to the Files and sim tabs in the Workspace (Figure 87). ModelSim also displays four Code Coverage panes in the Main window (Figure 88):

• **Missed Coverage**

   Displays the selected file's un-executed statements, branches, conditions, and expressions and signals that have not toggled.

• **Current Exclusions**

   Lists all files and lines that are excluded from coverage statistics (see "Excluding lines and files from coverage statistics" (T-116) for more information).

• **Instance Coverage**

   Displays statement, branch, condition, expression and toggle coverage statistics for each instance in a flat, non-hierarchical view.

• **Details**

   Shows details of missed coverage such as truth tables or toggle details.

These panes can be re-sized, rearranged, and "undocked" to make the data more easily viewable. To resize a pane, click-and-drag on the top or bottom border. To move a pane, click-and-drag on the double-line to the right of the pane name. Dragging a pane out of the Main window will "undock" that pane. To redock the pane, drag the pane back into the window by the "double bar" at the top of the pane window.

We will look at these panes more closely in the next exercise. For complete details on each pane, see the Code Coverage chapter in the *ModelSim User's Manual*.

Figure 87: Coverage columns in the Main window Workspace

Figure 88: Main window with coverage panes displayed
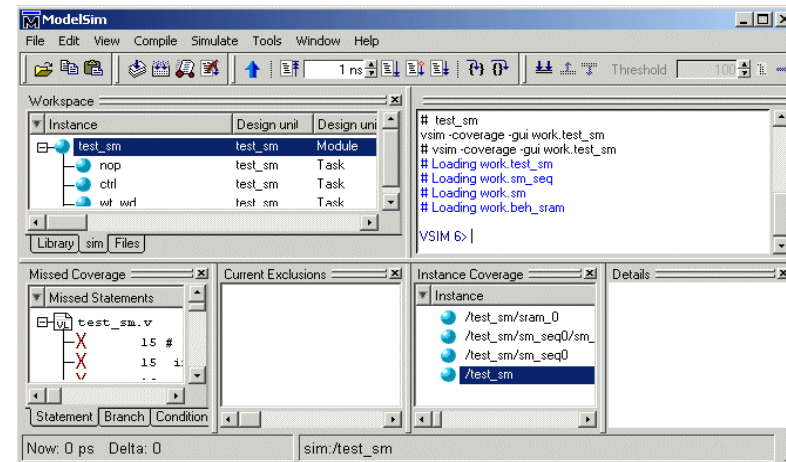
Viewing statistics in the Main window

Let's take a look at the data in these various panes.

1   View statistics in the Workspace pane.

a   Select the Files tab in the Workspace and scroll to the right.

Each file in the design shows summary statistics for statements, branches, conditions, and expressions.

b   Select the sim tab in the Workspace and scroll to the right.

Coverage statistics are shown for each item in the design.

c   Click the right-mouse button on any column name and select an item from the list (Figure 89).

Whichever column you selected is hidden. To redisplay the column, right-click again and select that column name. The status of which columns are displayed or hidden is persistent between invocations of ModelSim.

2   View statistics in the Missed Coverage pane.

a   Select different files from the Files tab of the Workspace.

The Missed Coverage pane updates to show statistics for the selected file (Figure 90).

b   Select any entry in the Statement tab to display that line in the Source window.

Figure 89: Right click a column heading to hide or show columns
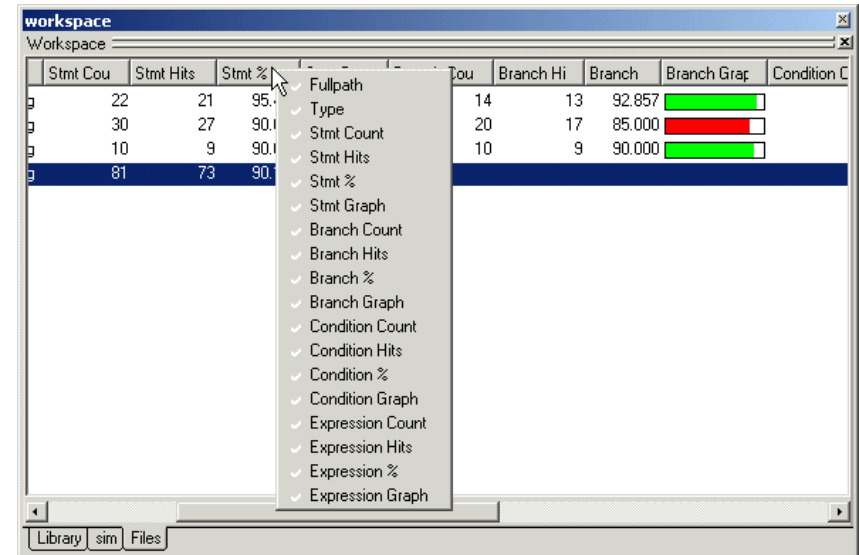
Figure 90: Statement statistics in the Missed Coverage pane
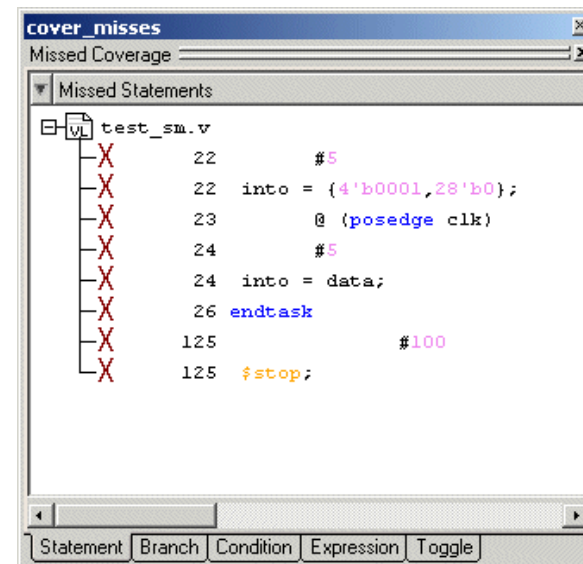
c   Select the Toggle tab in the Missed Coverage pane.

   If the Toggle tab isn't visible, you can do one of two things: 1) widen the
   pane by clicking-and-dragging on the pane border; 2) if your mouse has
   a middle button, click-and-drag the tabs with the middle mouse button.

d   Select any item in the Toggle tab to see details in the Details pane (Figure
   91).

3   View instance coverage statistics.

   The Instance Coverage pane displays coverage statistics for each instance in
   a flat, non-hierarchical view (Figure 92). Select any instance in the Instance
   Coverage pane to see its source code displayed in the Source window.

Figure 91: Details pane showing toggle coverage statistics
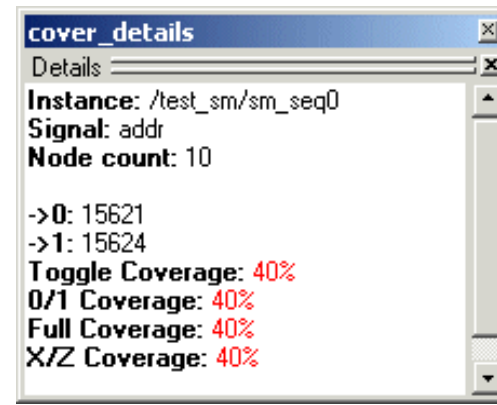
Figure 92: The Instance Coverage pane

Viewing statistics in the Source window

In the previous section you saw that the Source window and the Main window coverage panes are linked. You can select items in the Main window panes to view the underlying source code in the Source window. Furthermore, the Source window contains statistics of its own.

1   View coverage statistics for *test_sm* in the Source window.

    a   Make sure *test_sm* is selected in the sim tab of the Workspace.

   b   In the Statement tab of the Missed Coverage pane, expand *test_sm* and select any line (Figure 93).

       The Source window pops up with the line you selected highlighted in yellow (Figure 94).

    c   Switch to the Source window.

       The table below describes the various icons.

Icon	Description
green checkmark	indicates a statement that has been executed
red X	indicates that a statement in that line has not been executed (zero hits)
green E	indicates a line that has been excluded from code coverage statistics
red X_T or X_F	indicates that a true or false branch (respectively) of a conditional statement has not been executed

Lines that contain unexecuted statements and branches are highlighted in pink.

Figure 93: Selecting a line in the Missed Coverage pane
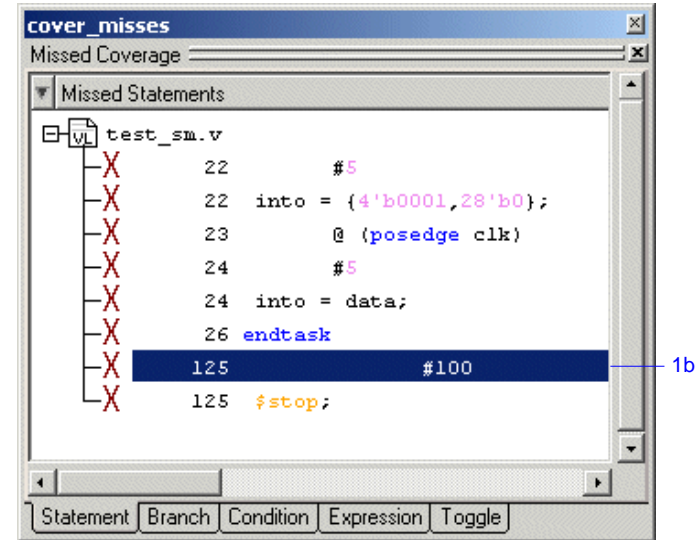
Figure 94: Coverage statistics in the Source window
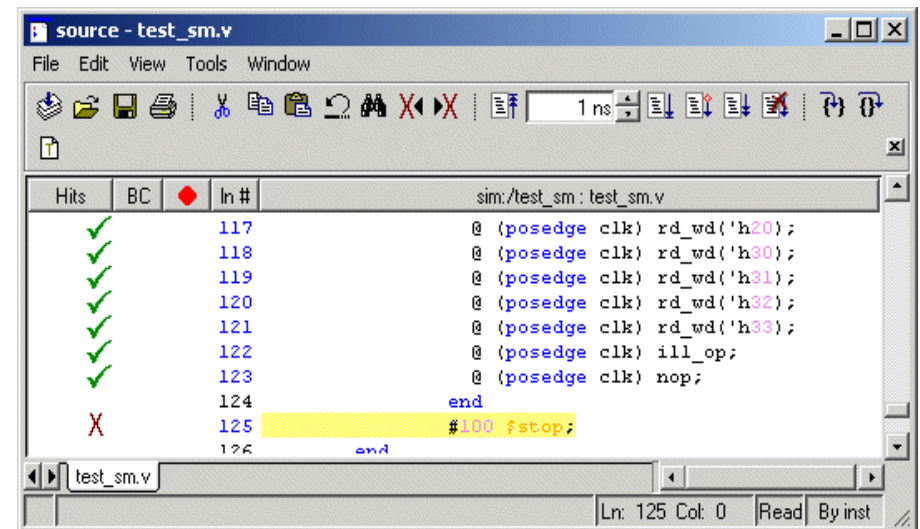
d Hover your mouse pointer over a line of code with a green checkmark.

The icons change to numbers that indicate how many times the statements and branches in that line were executed (Figure 95). In this case line 122 was executed 1562 times.

e Select **View > Show coverage numbers**.

The icons are replaced by execution counts on every line (Figure 96). An ellipsis (...) is displayed whenever there are multiple statements on the line. Hover the mouse pointer over a statement to see the count for that statement.

f Select **View > Show coverage numbers** again to return to icon display.

Figure 95: Coverage numbers shown by hovering the mouse pointer
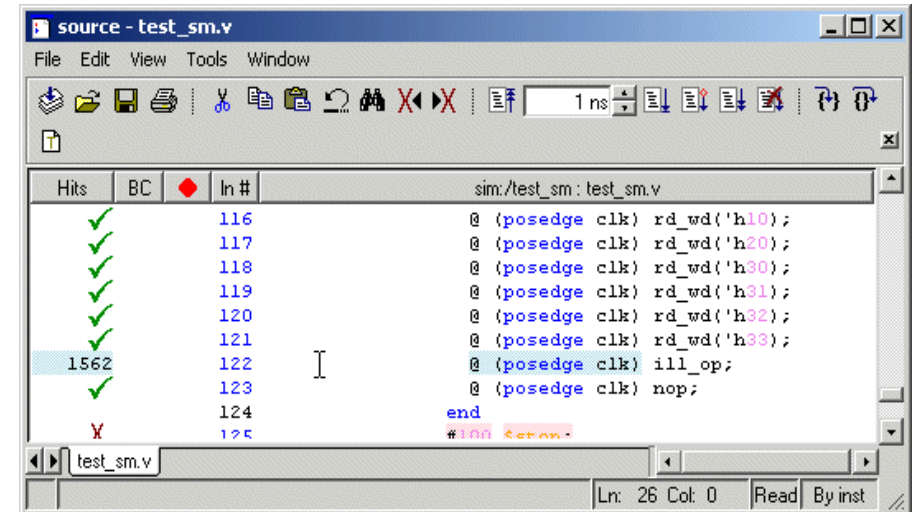
Figure 96: Coverage numbers on every line
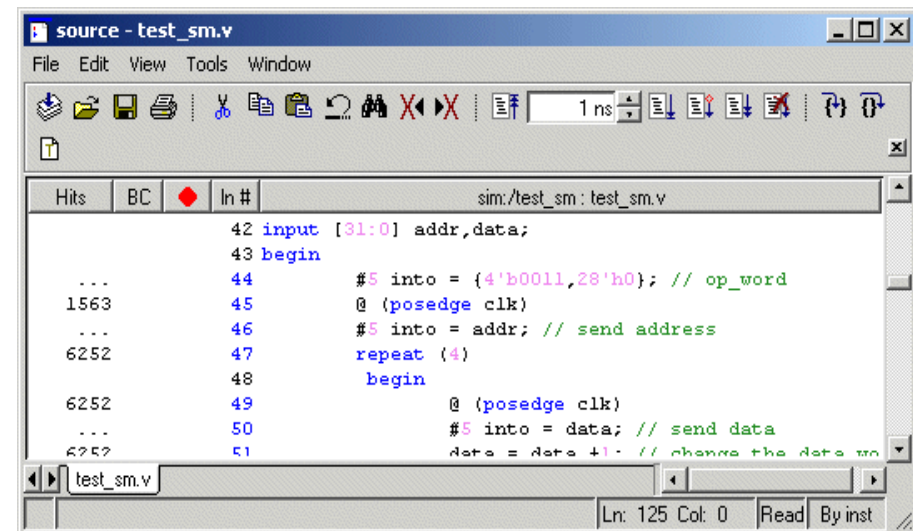
Viewing toggle statistics in the Signals window

Toggle coverage counts each time a logic node transitions from one state to another. In this example you enabled two-state toggle coverage (0 -> 1 and 1-> 0) with the **-cover t** argument. Alternatively, you can enable six-state toggle coverage using the **-cover x** argument. See the *ModelSim User's Manual* for more details.

1   View toggle data in the Signals window.

    a   Select *test_sm* in the sim tab of the Main window.

    b   Select **View > Signals** to open the Signals window.

    c   Scroll to the right and you will see the various toggle coverage columns (Figure 97).

        The blank columns show data when you have extended toggle coverage enabled.

Figure 97: Toggle coverage columns in the Source window

Excluding lines and files from coverage statistics

ModelSim allows you to exclude lines and files from code coverage statistics. You can set exclusions with the GUI, with a text file called an "exclusion filter file", or with "pragmas" in your source code. Pragmas are statements that instruct ModelSim to not collect statistics for the bracketed code. See the *ModelSim User's Manual* for more details on exclusion filter files and pragmas.

1   Exclude a line via the Source window.

   a   In the *Hits* column of the Source window, right-click a line and select **Exclude Coverage Line #** (Figure 98).

        Several things happen once you exclude the line:

        • The icon in the Source window changes to a green E.

        • The line is removed from the Statement tab of the Missed Coverage pane.

        • A new entry appears in the Current Exclusions pane.

        • The statistics are updated to reflect the exclusion.

2   Exclude a line via the Missed Coverage pane.

   a   Right click a line in the Missed Coverage pane and select **Exclude Selection**. (You can also exclude the selection for the current instance only by selecting Exclude Selection For Instance <inst_name>.)

3   Exclude an entire file.

   a   In the Files tab of the Workspace, locate *sm.v* (or *sm.vhd* if you are using the VHDL example).

   b   Right-click the file name and select **Coverage > Exclude Selected File** (Figure 99).

        The file is added to the Current Exclusions pane.

4   Cancel the exclusion of *sm.v*.

   a   Right-click *sm.v* in the Current Exclusions pane and select **Cancel Selected Exclusions**.

Figure 98: Excluding a line in the Source window
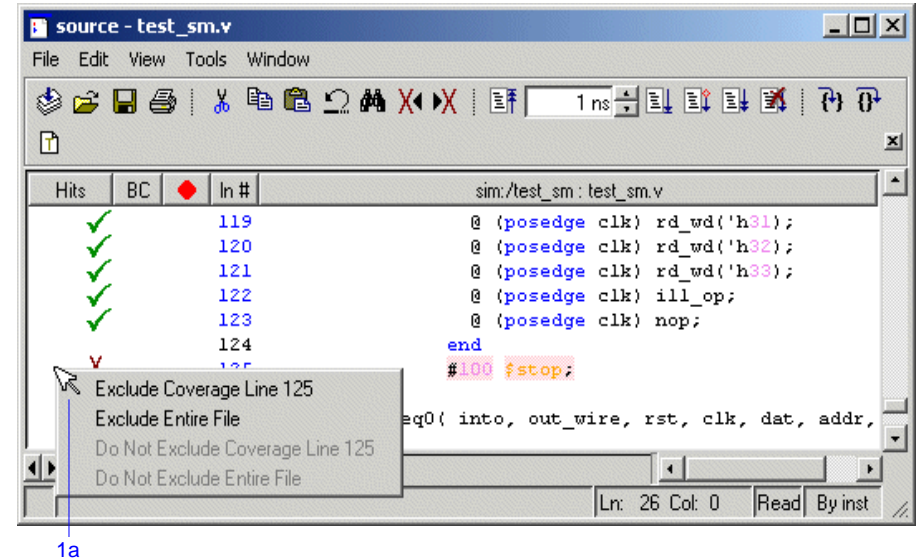
Figure 99: Excluding an entire file via the GUI
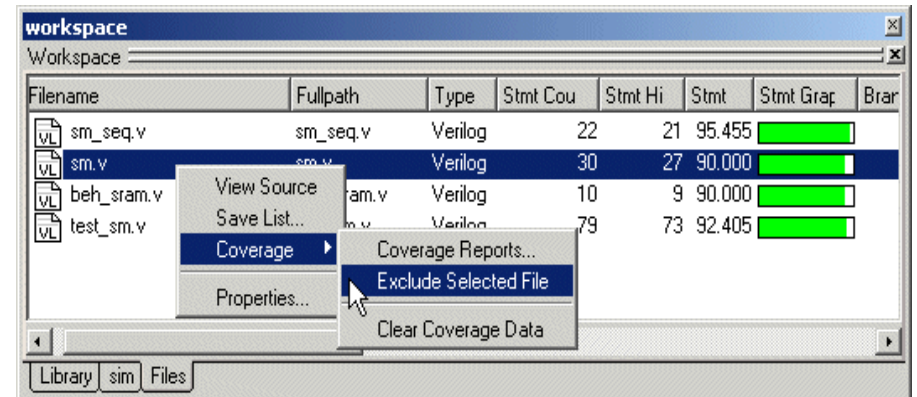
Creating Code Coverage reports

You can create reports on the coverage statistics using either the menus or by entering commands in the Transcript pane. The reports are output to a text file regardless of which method you use.

To create coverage reports via the menus, do one of the following:

- select **Tools > Coverage > Reports** from the Main window menu

- right-click any item in the sim or Files tab of the Main window Workspace and select **Coverage > Coverage Reports** from the context menu (and submenu)

- right-click any item in the Instance Coverage pane and select **Coverage reports** from the context menu

1   Create a report on all instances.

   a   Select **Tools > Coverage > Reports** from the Main window toolbar.

      This opens the Coverage Report dialog (Figure 100).

   b   Make sure **Report on all instances** and **No Filtering** are selected and then click OK.

      ModelSim creates a file *report.txt* in the current directory and displays the report in Notepad (Figure 101).

   c   Close Notepad when you are done looking at the report.

Figure 100: The Coverage Report dialog
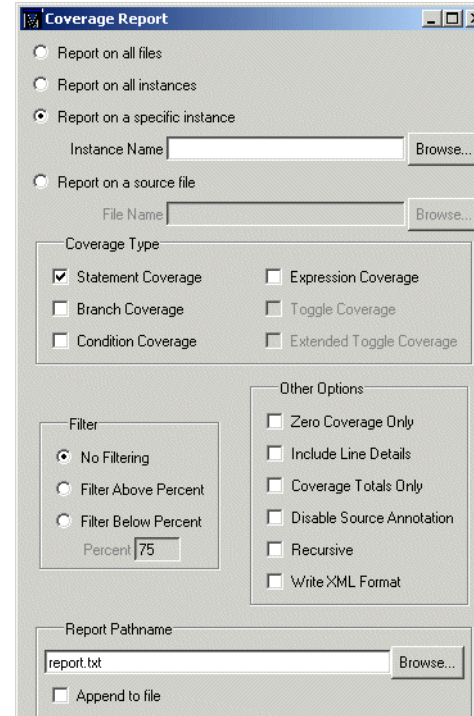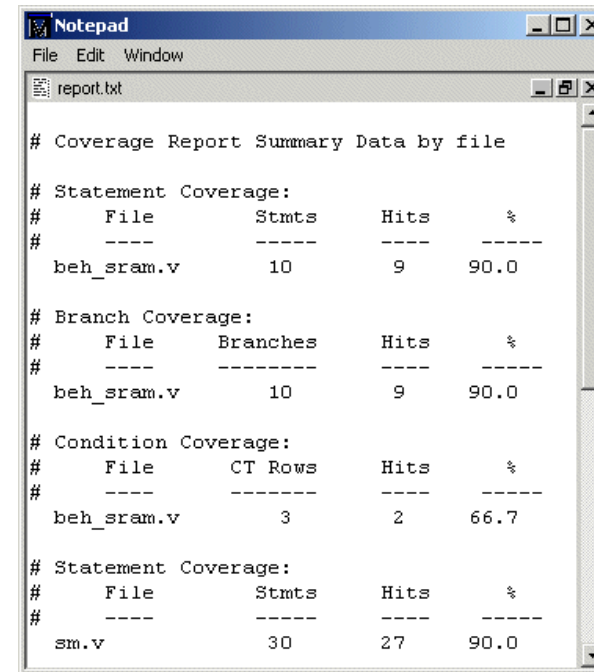
Figure 101: A coverage report in Notepad
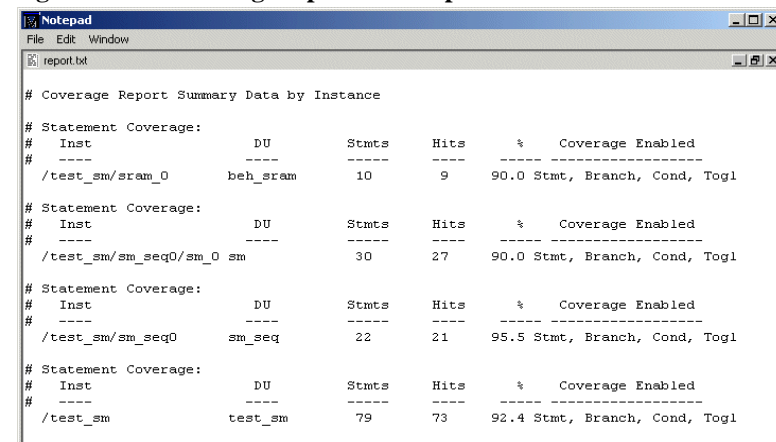
2 Create a summary report on all design files from the Transcript pane.

    a   Type **coverage report -file cover.txt** at the VSIM> prompt.

    b   Type **notepad cover.txt** at the VSIM> prompt to view the report (Figure 102).

    c   Close Notepad when you are done reviewing the report.

Figure 102: A summary report of all files

Lesson wrap-up

This concludes this lesson. Before continuing we need to end the current simulation.

1   Type **quit -sim** at the VSIM> prompt.

Lesson 11 - Waveform Compare

Topics

The following topics are covered in this lesson:

▶ **Note:** The functionality described in this tutorial requires a compare license feature in your ModelSim license file. Please contact your Mentor Graphics sales representative if you currently do not have such a feature.

Introduction

Waveform Compare computes timing differences between test signals and reference signals. 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

In this exercise you will run and save a simulation, edit one of the source files, run the simulation again, and finally compare the two runs.

Design files for this lesson

The sample design for this lesson consists of a finite state machine which controls a behavioral memory. The testbench *test_sm* provides stimulus.

The ModelSim installation comes with Verilog and VHDL versions of this design. The files are located in the following directories:

Verilog – *<install_dir>/modeltech/examples/compare/verilog*

VHDL – *<install_dir>/modeltech/examples/compare/vhdl*

This lesson uses the Verilog version in the examples. If you have a VHDL license, use the VHDL version instead. When necessary, we distinguish between the Verilog and VHDL versions of the design.

Related reading

ModelSim User's Manual – Chapter 13 - Waveform Compare (UM-455),
Chapter 9 - WLF files (datasets) and virtuals (UM-239)

Creating the reference dataset

The reference dataset is the *.wlf* file that the test dataset will be compared against. It can be a saved dataset, the current simulation dataset, or any part of the current simulation dataset.

In this exercise you will use a DO file to create the reference dataset.

1   Create a new directory and copy the tutorial files into it.

Start by creating a new directory for this exercise (in case other users will be working with these lessons). Create the directory and copy all files from *<install_dir>/modeltech/examples/dataflow/verilog* to the new directory.

If you have a VHDL license, copy the files in *<install_dir>/modeltech/ examples/dataflow/vhdl* instead.

2   Start ModelSim and change to the exercise directory.

If you just finished the previous lesson, ModelSim should already be running. If not, start ModelSim.

a   Type **vsim** at a UNIX shell prompt or use the ModelSim icon in Windows.

If the Welcome to ModelSim dialog appears, click **Close**.

b   Select **File > Change Directory** and change to the directory you created in step 1.

3   Execute the lesson DO file.

a   Type **do gold_sim.do** at the ModelSim> prompt.

The 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 look at its contents.

Creating the test dataset

The test dataset is the *.wlf* file that will be compared against the reference dataset. Like the reference dataset, the test dataset can be a saved dataset, the current simulation dataset, or any part of the current simulation dataset.

To simplify matters, you will create the test dataset from the simulation you just ran. However, you will edit the testbench to create differences between the two runs.

Verilog

1   Edit the testbench.

   a   Open *test_sm.v* in the Source window.

   b   Select **Edit > read only** to make the file writable.

   c   Scroll to line 113, which looks like this:

   ```
   @ (posedge clk) wt_wd('h10,'haa);
   ```

   d   Change the data pattern 'aa' to 'ab':

   ```
   @ (posedge clk) wt_wd('h10,'hab);
   ```

   e   Select **File > Save** to save the file.

2   Compile the revised file and rerun the simulation.

   a   Type **do sec_sim.do** at the ModelSim> prompt.

   The DO file does the following:

   • Re-compiles the testbench

   • Adds waves to the Wave window

   • Runs the simulation

VHDL

1 Edit the testbench.

a Open *test_sm.vhd* in the Source window.

b Select **Edit > read only** to make the file writable.

c Scroll to line 142, which looks like this:

```
wt_wd ( 16#10#, 16#aa#, clk, into );
```

d and change the data pattern 'aa' to 'ab'. Line 142 should look like this:

```
wt_wd ( 16#10#, 16#ab#, clk, into );
```

e Select **File > Save** to save the file.

2 Compile the revised file and rerun the simulation.

a Type **do sec_sim.do** at the ModelSim> prompt.

The DO file does the following:

• Re-compiles the testbench

• Adds waves to the Wave window

• Runs the simulation

Comparing the simulation runs

ModelSim includes a Comparison Wizard that walks you through the process. You can also configure the comparison manually with menu or command line commands.

1   Create a comparison using the Comparison Wizard.

   a   Select **Tools > Waveform Compare > Comparison Wizard**.

   b   Click the **Browse** button and select *gold.wlf* as the reference dataset (Figure 103).

      Recall that *gold.wlf* is from the first simulation run.

   c   Leaving the test dataset set to **Use Current Simulation**, click **Next**.

   d   Select **Compare All Signals** in the second dialog and click **Next** (Figure 104).

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

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

Figure 103: First dialog of the Comparison Wizard
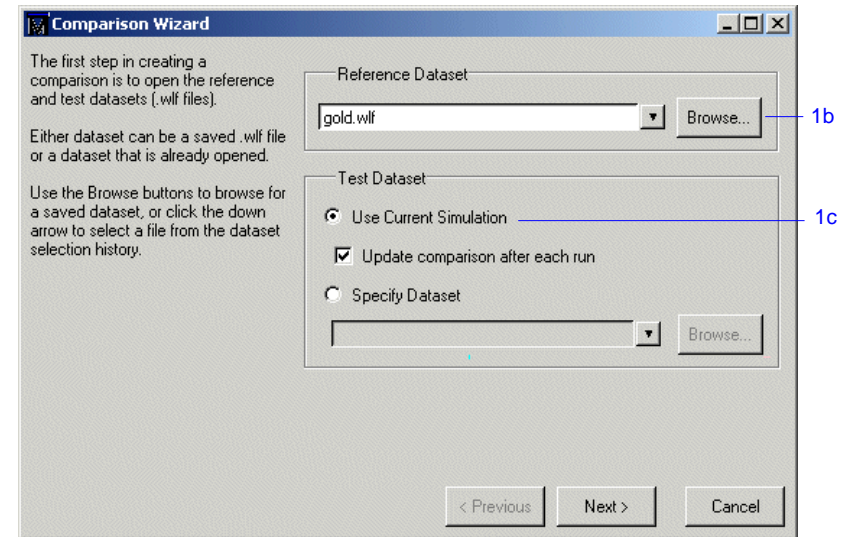
Figure 104: Second dialog of the Comparison Wizard
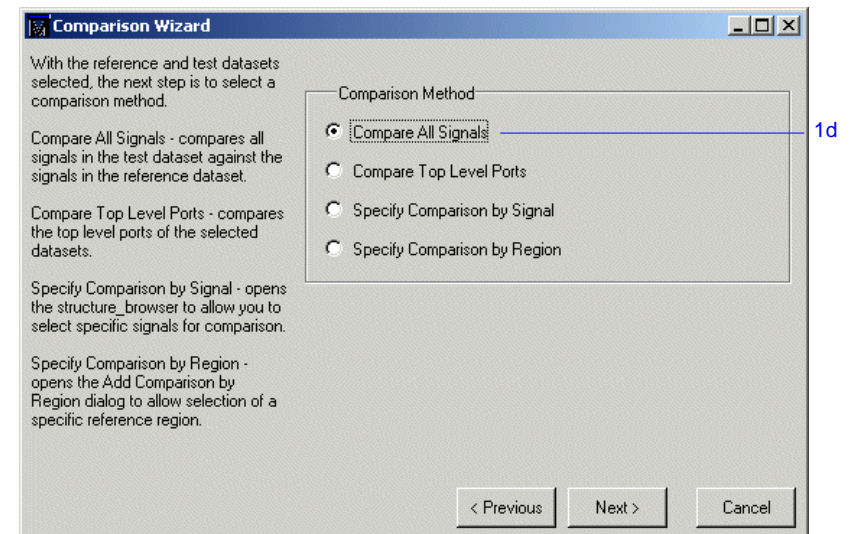
Viewing comparison data

Comparison data displays in three places within the ModelSim GUI: the Main window, the Wave window, and the List window.

Viewing comparison data in the Main window

The Compare tab in the Main window shows the region that was compared, and the Transcript shows the number of differences found between the reference and test datasets (Figure 105).

Viewing comparison data in the Wave window

In the signals pane of the Wave window, a timing differences is denoted by a red X (Figure 106). Red areas in the waveform pane show the location of the timing differences, as do the red lines in the scrollbars. Annotated differences are highlighted in blue.

The Wave window includes six compare icons that let you quickly jump between differences (Figure 107). 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 compare icons cycle through differences on all signals. To view differences for just the selected signal, use <tab> and <shift - tab>.

Figure 105: The Compare tab in the Main window Workspace
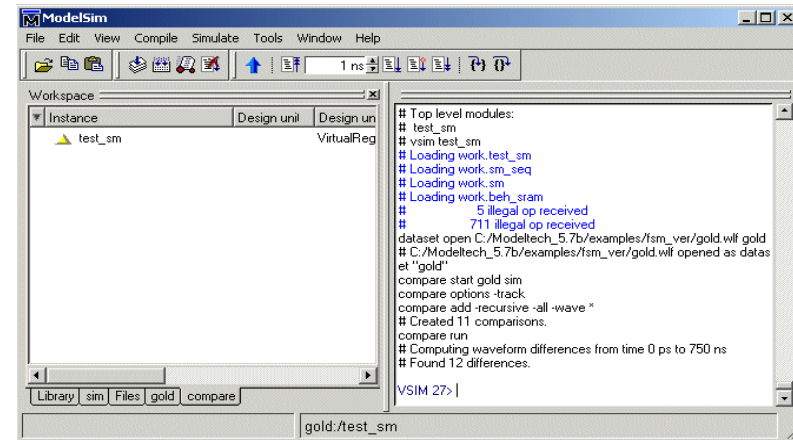
Figure 106: Comparison items in the Wave window
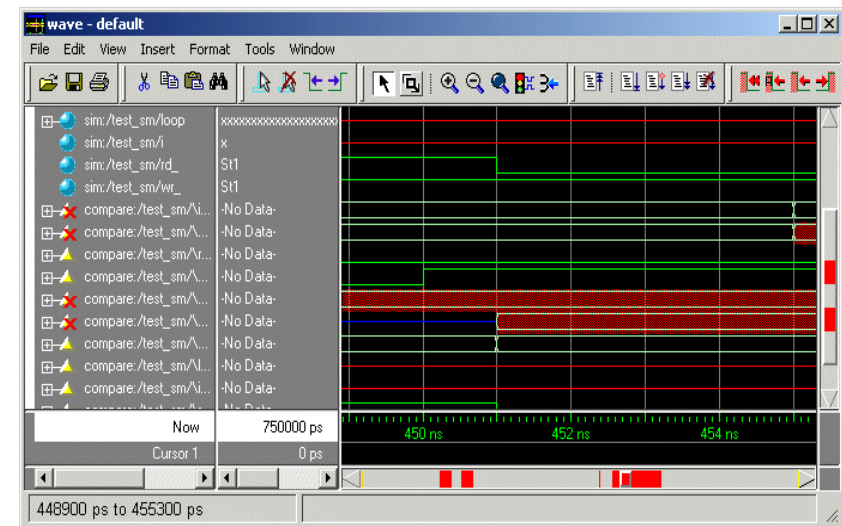
Figure 107: The compare icons

Viewing comparison data in the List window

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

1   Add comparison data to the List window.

   a   Select **View > List** from the Main window menu bar.

   b   Drag the *test_sm* comparison object from the compare tab of the Main window to the List window.

   c   Scroll down the window.

       Differences are noted with yellow highlighting (Figure 108).

Figure 108: Compare differences in the List window

Saving and reloading comparison data

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

To save comparison data so it can be reloaded into ModelSim, you must save two files. First, you save the computed differences to one file; next, you save the comparison configuration rules to a separate file. When you reload the data, you must have the reference dataset open.

1   Save the comparison data to a text file.

   a   Select **Tools > Waveform Compare > Differences > Write Report**.

   b   Click **Save**.

      This saved *compare.txt* to the current directory (Figure 109).

   c   Type **notepad compare.txt** at the VSIM> prompt.

   d   Close Notepad when you are done reviewing the report.

2   Save the comparison data.

   a   Select **Tools > Waveform Compare > Differences > Save**.

   b   Click **Save**.

      This saved *compare.dif* to the current directory.

   c   Select **Tools > Waveform Compare > Rules > Save**.

   d   Click Save.

      This saved *compare.rul* to the current directory.

   e   Select **Tools > Waveform Compare > End Comparison**.

3   Reload the comparison data.

   a   Select **File > Open > Dataset** and open *gold.wlf*.

   b   Select **Tools > Waveform Compare > Reload**.

      Since you saved the data using default file names, the dialog should already have the correct files specified (Figure 110).

   c   Click **OK**.

      The comparison reloads.

Figure 109: Coverage data saved to a text file

Figure 110: Reloading saved comparison data

Lesson wrap-up

This concludes this lesson. Before continuing we need to end the current simulation and close the *gold.wlf* dataset.

1   Type **quit -sim** at the VSIM> prompt.

2   Type **dataset close gold** at the ModelSim> prompt.

Lesson 12 - Automating ModelSim

Topics

The following topics are covered in this lesson:

Introduction

Aside from executing a couple of pre-existing DO files, the previous lessons focused on using ModelSim in interactive mode: executing single commands, one after another, via the GUI menus or Main window command line. In situations where you have repetitive tasks to complete, you can increase your productivity with DO files.

DO files are scripts that allow you to execute many commands at once. The scripts can be as simple as a series of ModelSim commands with associated arguments, or they can be full-blown Tcl programs with variables, conditional execution, and so forth. You can execute DO files from within the GUI or you can run them from the system command prompt without ever invoking the GUI.

▲ **Important:** This lesson assumes that you have added the *<install_dir>/modeltech/<platform>* directory to your PATH. If you did not, you will need to specify full paths to the tools (i.e., vlib, vmap, vlog, vcom, and vsim) that are used in the lesson.

Related reading

ModelSim User's Manual – Chapter 21 - Tcl and macros (DO files) (UM-239)

Practical Programming in Tcl and Tk, Brent B. Welch, Copyright 1997

Creating a simple DO file

Creating DO files is as simple as typing the commands in a text file. Alternatively, you can save the Main window transcript as a DO file. In this exercise, you will use the transcript to create a DO file that adds signals to the Wave window, provides stimulus to those signals, and then advances the simulation.

1   Load the *test_counter* design unit.

    a   If necessary, start ModelSim.

    b   Change to the directory you created in *Chapter Lesson 2 - Basic simulation*.

    c   In the Library tab of the Main window, double-click the *test_counter* design unit to load it.

    d   Select **File > Transcript > Clear Transcript** to clear the transcript.

2   Enter commands to add signals to the Wave window, force signals, and run the simulation.

    a   Enter the following commands, one at a time, at the VSIM> prompt in the Main window:

```
add wave count
add wave clk
add wave reset
force -freeze clk 0 0, 1 {50 ns} -r 100
force reset 1
run 100
force reset 0
run 300
force reset 1
run 400
force reset 0
run 200
```

3   Save the transcript.

    a   Select **File > Transcript > Save Transcript As**.

    b   Type **sim.do** in the **File name:** field and save it to the current directory.

4 View the DO file.

a Type **edit sim.do** at the VSIM prompt.

The DO file opens in either the Source window (Figure 111) or the editor you specified with the EDITOR environment variable. Make sure the file includes only those commands shown in step 2 above. Delete any extraneous commands and save the file.

5 Load the simulation anew and use the DO file.

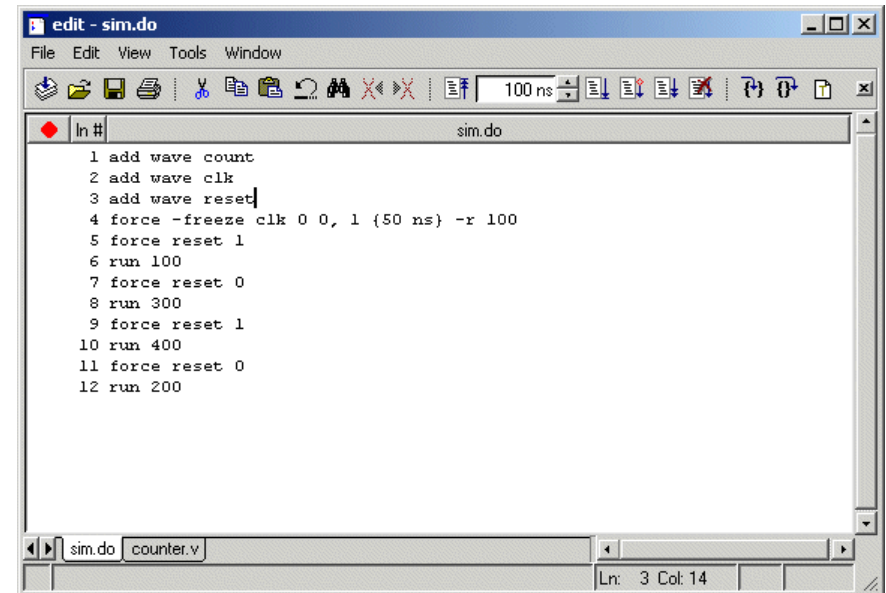a Type **quit -sim** at the VSIM> prompt.

b Type **vsim test_counter** at the VSIM> prompt.

c Type **do sim.do** at the VSIM> prompt.

ModelSim executes the saved commands and draws the waves in the Wave window.

6 When you are done with this exercise, select **File > Quit** to quit ModelSim.

Figure 111: The DO file opened in the Source window

Running ModelSim with a startup DO file

You can configure ModelSim to execute a particular DO file every time you start the tool. You modify the local ModelSim initialization file (*modelsim.ini*) to point at the file, and from that point forward, the DO file is executed whenever you invoke ModelSim from that directory.

In this exercise, you'll create a startup DO file that opens the Signals, Source, and Wave windows and changes their window titles to the name of your design. You'll be working from the system command prompt in the directory you created in *Chapter Lesson 2 - Basic simulation*.

1   Create the DO file.

    a   Open a text editor.

    b   Type the following lines into a new file:

```
view wave -title "$entity"
view si -title "$entity"
view so -title "$entity"
```

       The pre-defined variable *$entity* will read in the name of the top-level design unit you simulate.

    c   Save the file with the name *startup.do* and place it in the directory you created in *Chapter Lesson 2 - Basic simulation*.

2   Edit the *modelsim.ini* file.

    a   With a text editor, open the *modelsim.ini* file in the current directory. This should be the same directory to which you saved the *startup.do* file.

    b   Find the following line in the file:

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

    c   Remove the semicolon (;) to un-comment the line and then save and close the file.

3 Execute ModelSim from the system prompt.

a Open a DOS/ UNIX prompt and change to the directory you've been using in the last two steps.

b Type **vsim -title "counter" test_counter** at the command prompt.

ModelSim opens, loads the *test_counter* design unit, and then displays the Signals, Source, and Wave windows with the title "test_counter". Notice that we had to use the **-title** argument to **vsim** in order to change the title of the Main window.

4 Select **File > Quit** to close ModelSim.

5 Edit the *modelsim.ini* to remove the startup DO file.

a With a text editor, open the *modelsim.ini* file in the current directory.

b Find the following line in the file:

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

c Add a semicolon (;) and space to the beginning of the line and then save and close the file.

Running ModelSim in command-line mode

We use the term "command-line mode" to refer to simulations that are run from a DOS/ UNIX prompt without invoking the GUI. Several ModelSim commands (e.g., vsim, vlib, vlog, etc.) are actually stand-alone executables that can be invoked at the system command prompt. Additionally, you can create a DO file that contains other ModelSim commands and specify that file when you invoke the simulator.

1   Create a new directory and copy the tutorial files into it.

    Start by creating a new directory for this exercise. Create the directory and copy these files into it:

    •   *\<install_dir>\modeltech\examples\counter.v*

    •   *\<install_dir>\modeltech\examples\stim.do*

    We have used the Verilog file *counter.v* in this example. If you have a VHDL license, use *counter.vhd* instead.

2   Create a new design library and compile the source file.

    Again, enter these commands at a DOS/ UNIX prompt in the new directory you created in step 1.

    a   Type **vlib work** at the DOS/ UNIX prompt.

    b   For Verilog, type **vlog counter.v** at the DOS/ UNIX prompt. For VHDL, type **vcom counter.vhd**.

3   Create a DO file.

   a   Open a text editor.

   b   Type the following lines into a new file:

```
# list all signals in decimal format
add list -decimal *

# read in stimulus
do stim.do

# output results
write list counter.lst

# quit the simulation
quit -f
```

   c   Save the file with the name *sim.do* and place it in the current directory.

4   Run the batch-mode simulation.

   a   Type **vsim -c -do sim.do counter -wlf counter.wlf** at the DOS/ UNIX prompt.

       The **-c** argument instructs ModelSim not to invoke the GUI. The **-wlf** argument saves the simulation results in a WLF file. This allows you to view the simulation results in the GUI for debugging purposes.

5   View the list output.

   a   Open *counter.lst* and view the simulation results.

```
     ns        /counter/count
      delta        /counter/clk
                   /counter/reset
       0   +0            x z *
       1   +0            0 z *
      50   +0            0 * *
     100   +0            0 0 *
     100   +1            0 0 0
     150   +0            0 * 0
     151   +0            1 * 0
     200   +0            1 0 0
     250   +0            1 * 0
      .
      .
      .
```

This is the output produced by the Verilog version of the design. It may appear slightly different if you used the VHDL version.

6   View the results in the GUI.

Since you saved the simulation results in *counter.wlf*, you can view them in the GUI by invoking VSIM with the **-view** argument.

a   Type **vsim -view counter.wlf** at the DOS/ UNIX prompt.

The GUI opens and a dataset tab named "counter" is displayed in the Workspace (Figure 112).

b   Right-click the *counter* instance and select **Add > Add to Wave**.

The waveforms display in the Wave window.

7   When you finish viewing the results, select **File > Quit** to close ModelSim.

Figure 112: A dataset in the Main window Workspace
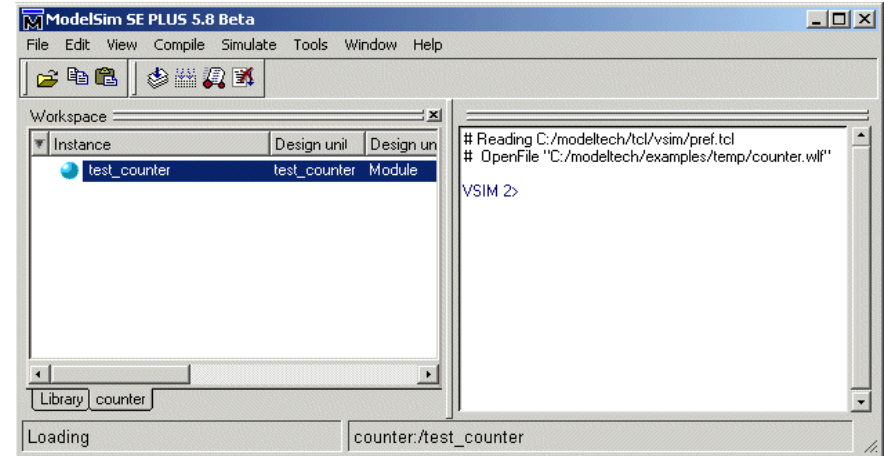
Using Tcl with ModelSim

The DO files used in previous exercises contained only ModelSim commands. However, DO files are really just Tcl scripts. This means you can include a whole variety of Tcl constructs such as procedures, conditional operators, math and trig functions, regular expressions, and so forth.

In this exercise you'll create a simple Tcl script that tests for certain values on a signal and then adds bookmarks that zoom the Wave window when that value exists. Bookmarks allow you to save a particular zoom range and scroll position in the Wave window. The Tcl script also creates buttons in the Main window that call these bookmarks.

1   Create the script.

    a   In a text editor, open a new file and enter the following lines:

```
proc add_wave_zoom {stime num} {
 echo "Bookmarking wave $num"
 bookmark add wave "bk$num"  "[expr $stime - 50] [expr $stime +
100]" 0
 add button "$num" [list bookmark goto wave bk$num]
}
```

These commands do the following:

- Create a new procedure called "add_wave_zoom" that has two arguments, *stime* and *num*.

- Create a bookmark with a zoom range from the current simulation time minus 50 time units to the current simulation time plus 100 time units.

- Add a button to the Main window that calls the bookmark.

    b   Now add these lines to the bottom of the script:

```
add wave -r /*
when {clk'event and clk="1"} {
   echo "Count is [exa count]"
   if {[exa count]== "00100111"} {
     add_wave_zoom $now 1
   } elseif {[exa count]== "01000111"} {
     add_wave_zoom $now 2
   }
}
```

These commands do the following:

- Add all signals to the Wave window.

- Use a when statement to identify when *clk* transitions to 1.

- Examine the value of *count* at those transitions and add a bookmark if it is a certain value.

c   Save the script with the name "*add_bkmrk.do*."

Save it into the directory you created in *Chapter Lesson 2 - Basic simulation*.

2   Load the *test_counter* design unit.

a   Start ModelSim.

b   Select **File > Change Directory** and change to the directory you saved the DO file to in step 1c above.

c   In the Library tab of the Main window, expand the *work* library and double-click the *test_counter* design unit.

3   Execute the DO file and run the design.

a   Type **do add_bkmrk.do** at the VSIM> prompt.

b   Type **run 1500 ns** at the VSIM> prompt.

The simulation runs and the DO file creates two bookmarks. It also creates buttons (labeled "1" and "2") on the Main window toolbar that jump to the bookmarks (Figure 113).

c   Click the buttons and watch the Wave window zoom on and scroll to the time when *count* is the value specified in the DO file.

Figure 113: Buttons added to the Main window toolbar
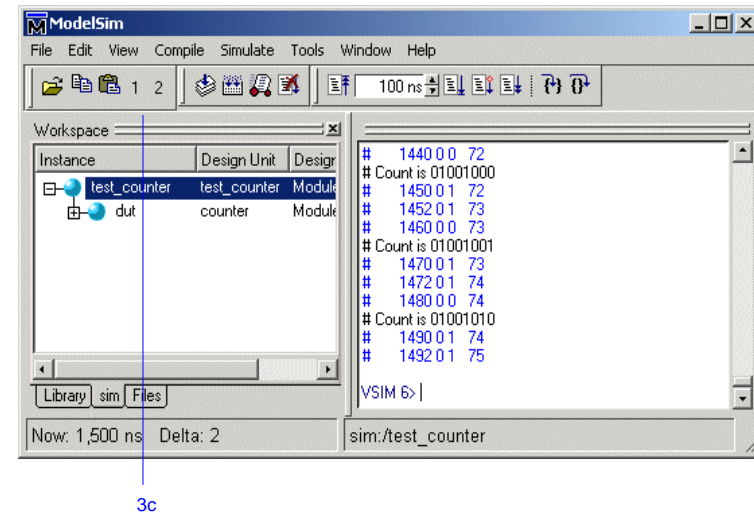
Lesson Wrap-up

This concludes this lesson.

1   Select **File > Quit** to close ModelSim.

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 acquiring the license, and Mentor Graphics Corporation and Mentor Graphics (Ireland) Limited, acting directly or through their subsidiaries or authorized distributors (collectively "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 five 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, documentation and design data ("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 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' standard policies and programs, 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) support services provided, including eligibility to receive telephone support, updates, modifications and revisions. Current standard policies and programs are available upon request.

2. **ESD SOFTWARE.** If you purchased a license to use embedded software development ("ESD") Software, Mentor Graphics 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.** 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. 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. You agree that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceives or made 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 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 applicable and mandatory 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, whether by operation of law or otherwise ("attempted transfer") without Mentor Graphics' prior written consent and payment of Mentor Graphics then-current applicable transfer charges. Any attempted transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may. at Mentor graphics' option, result in the immediate termination of the Agreement and licenses granted under this Agreement. The provisions of this section 4 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 LICENSED TO YOU FOR A LIMITED TERM OR LICENSED 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, IMPLIED, OR STATUTORY, 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 AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

6. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, 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.

8. **INDEMNIFICATION.** 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 YOUR USEOF SOFTWARE AS DESCRIBED IN SECTION 7.

9. **INFRINGEMENT.**

9.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against you alleging that Software infringes a patent or copyright or misappropriates a trade secret in the United States, Canada, Japan, or member state of the European Patent Office. Mentor Graphics will pay any costs and damages finally awarded against you that are attributable to the infringement action. You understand and agree that as conditions to Mentor Graphics' obligations under this section you must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to defend or settle the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

9.2. If an infringement claim is made, Mentor Graphics may, at its option and expense: (a) replace or modify Software so that it becomes noninfringing; (b) procure for you the right to continue using Software; or (c) require the return of Software and refund to you any license fee paid, less a reasonable allowance for use.

9.3. Mentor Graphics has no liability to you if 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 make, use or sell; (f) any Beta Code contained in Software; (g) any Software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by you that is deemed willful. In the case of (h) you shall reimburse Mentor Graphics for its attorney fees and other costs related to the action upon a final judgment.

9.4. THIS SECTION 9 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 OR TRADE SECRET MISAPPROPRIATION BY ANY SOFTWARE LICENSED UNDER THIS AGREEMENT.

10. **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.

11. **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 any Software or direct product of Software in any manner without first obtaining all necessary approval from appropriate local and United States government agencies.

12. **RESTRICTED RIGHTS NOTICE.** Software was 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 SW Boeckman Road, Wilsonville, Oregon 97070-7777 USA.

13. **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.

14. **AUDIT RIGHTS.** With reasonable prior notice, Mentor Graphics shall have the right to audit during your normal business hours all records and accounts as may contain information regarding your compliance with the terms of this Agreement. Mentor Graphics shall keep in confidence all information gained as a result of any audit. Mentor Graphics shall only use or disclose such information as necessary to enforce its rights under this Agreement.

15. **CONTROLLING LAW AND JURISDICTION.** THIS AGREEMENT SHALL BE GOVERNED BY AND CONSTRUED UNDER THE LAWS OF OREGON, USA, IF YOU ARE LOCATED IN NORTH OR SOUTH AMERICA, AND THE LAWS OF IRELAND IF YOU ARE LOCATED OUTSIDE OF NORTH AND SOUTH AMERICA. All disputes arising out of or in

relation to this Agreement shall be submitted to the exclusive jurisdiction of Dublin, Ireland when the laws of Ireland apply, or Wilsonville, Oregon when the laws of Oregon apply. This section shall not restrict Mentor Graphics' right to bring an action against you in the jurisdiction where your place of business is located.

16. **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.

17. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding 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) either referenced in the purchase order or otherwise governing this subject matter. This Agreement may only be modified in writing by authorized representatives of the parties. 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. 020826, Part Number 214231

Index