# Table of Contents

## 1 - Introduction (UM-15)

- ModelSim graphic interface ........................................ UM-15
- Standards supported ............................................... UM-16
- Assumptions .......................................................... UM-16
- Sections in this document ......................................... UM-16
- Command reference .................................................. UM-18
- Text conventions ..................................................... UM-18
- What is an "HDL item" ............................................... UM-19
- Where to find our documentation ................................. UM-19
  - Download a free PDF reader with Search ................... UM-20
- Technical support and updates .................................... UM-20

## 2 - Projects and system initialization (UM-21)

- Introduction ......................................................... UM-22
  - What are projects? ................................................ UM-22
  - What are the benefits of projects? ............................. UM-22
  - How do projects differ in version 5.5? ......................... UM-23
- Getting started with projects .................................... UM-24
  - Step 1 — Create a new project ................................ UM-25
  - Step 2 — Add files to the project ............................... UM-27
  - Step 3 — Compile the files ..................................... UM-28
  - Step 4 — Simulate a design ..................................... UM-29
  - Other project operations ........................................ UM-29
- Customizing project settings ..................................... UM-30
  - Changing compile order ......................................... UM-30
  - Grouping files .................................................... UM-30
  - Setting compiler options ....................................... UM-31
- Accessing projects from the command line ..................... UM-32
- System initialization ............................................... UM-33
  - Files accessed during startup ................................ UM-33
  - Environment variables accessed during startup ............ UM-34
  - Initialization sequence ......................................... UM-35

## 3 - Design libraries (UM-37)

- Design library contents ........................................... UM-38
- Design unit information ........................................... UM-38
- Design library types ............................................... UM-38
- Working with design libraries ................................... UM-39
  - Creating a library ............................................... UM-39
# Table of Contents

### 4 - VHDL Simulation (UM-51)

- Compiling and simulating with the GUI ........................................... UM-52
- ModelSim variables ...................................................................... UM-52
- Compiling VHDL designs ................................................................. UM-53
  - Creating a design library .......................................................... UM-53
  - Invoking the VHDL compiler ...................................................... UM-53
  - Dependency checking ................................................................ UM-53
- Simulating VHDL designs .............................................................. UM-54
  - Invoking the simulator from the Main window ......................... UM-54
  - Invoking Code Coverage with Vsim ......................................... UM-54
- Using the TextIO package ............................................................... UM-55
  - Syntax for file declaration ........................................................ UM-55
  - Using STD_INPUT and STD_OUTPUT within ModelSim .......... UM-56
- TextIO implementation issues ......................................................... UM-57
  - Writing strings and aggregates .................................................. UM-57
  - Reading and writing hexadecimal numbers ............................... UM-58
  - Dangling pointers ................................................................... UM-58
  - The ENDLINE function .............................................................. UM-58
  - The ENDFILE function ............................................................... UM-58
  - Using alternative input/output files ........................................... UM-59
  - Providing stimulus ................................................................ UM-59
- Obtaining the VITAL specification and source code ......................... UM-60
- VITAL packages ........................................................................ UM-60
- ModelSim VITAL compliance ......................................................... UM-61
  - VITAL compliance checking ...................................................... UM-61
  - VITAL compliance warnings ..................................................... UM-61
- Compiling and Simulating with accelerated VITAL packages .......... UM-62
- Util package ............................................................................. UM-63
  - get_resolution() .................................................................. UM-63
  - init_signal_spy() .................................................................. UM-64
  - to_real() ............................................................................ UM-65
5 - Verilog Simulation (UM-67)

- ModelSim variables
- Compilation
  - Incremental compilation
  - Library usage
  - Verilog-XL compatible compiler options
  - Verilog-XL 'uselib compiler directive
- Simulation
  - Invoking the simulator
  - Simulation resolution limit
  - Event order issues
  - Verilog-XL compatible simulator options
- Compiling for faster performance
  - Compiling with -fast
  - Compiling mixed designs with -fast
  - Compiling gate-level designs with -fast
  - Referencing the optimized design
  - Enabling design object visibility with the +acc option
  - Using pre-compiled libraries
- Cell Libraries
- SDF timing annotation
- Delay modes
- System Tasks
  - IEEE Std 1364 system tasks
  - Verilog-XL compatible system tasks
  - $init_signal_spy
- Compiler Directives
  - IEEE Std 1364 compiler directives
  - Verilog-XL compatible compiler directives
- Verilog PLI/VPI
  - Registering PLI applications
  - Registering VPI applications
  - Compiling and linking PLI/VPI applications
  - The PLI callback reason argument
  - The sizetf callback function
  - PLI object handles
  - Third party PLI applications
  - Support for VHDL objects
  - IEEE Std 1364 ACC routines
  - IEEE Std 1364 TF routines
  - Verilog-XL compatible routines
  - 64-bit support in the PLI
  - PLI/VPI tracing
# 6 - Mixed VHDL and Verilog Designs (UM-121)

- Separate compilers, common libraries ................................................. UM-122
- Mapping data types ........................................................................ UM-122
  - VHDL generics ............................................................................... UM-122
  - Verilog parameters ....................................................................... UM-123
  - VHDL and Verilog ports ................................................................ UM-123
  - Verilog states ............................................................................... UM-124
- VHDL instantiation of Verilog design units ....................................... UM-126
- Verilog instantiation criteria ............................................................... UM-126
  - Component declaration ................................................................. UM-126
- VHDL instantiation of VHDL design units .......................................... UM-129
- Verilog instantiation criteria ............................................................... UM-129
- SDF annotation ................................................................................ UM-129

## 7 - WLF files (datasets) and virtuals (UM-131)

- WLF files (datasets) ........................................................................ UM-132
  - Saving a simulation to a WLF file ............................................... UM-132
  - Opening datasets ......................................................................... UM-133
  - Viewing dataset structure ............................................................ UM-134
  - Managing datasets ....................................................................... UM-136
  - Using datasets with ModelSim commands ..................................... UM-136
  - Restricting the dataset prefix display .......................................... UM-137
- Virtual Objects (User-defined buses, and more) ................................ UM-138
  - Virtual signals ............................................................................ UM-138
  - Virtual functions .......................................................................... UM-139
  - Virtual regions ............................................................................ UM-140
  - Virtual types ................................................................................ UM-140
- Dataset, WLF file, and virtual commands ......................................... UM-141

## 8 - Graphic Interface (UM-143)

- Window overview ............................................................................. UM-144
- Common window features ................................................................. UM-145
  - Quick access toolbars ................................................................ UM-146
  - Drag and Drop ............................................................................... UM-146
  - Command history ......................................................................... UM-146
  - Automatic window updating ......................................................... UM-147
  - Finding names, searching for values, and locating cursors .......... UM-147
  - Sorting HDL items ........................................................................ UM-148
  - Multiple window copies ................................................................ UM-148
  - Saving window layout .................................................................. UM-148
  - Context menus ............................................................................. UM-148
  - Menu tear off .............................................................................. UM-148
  - Customizing menus and buttons .................................................. UM-148
<table>
<thead>
<tr>
<th>Window Type</th>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main window</td>
<td>Combining signals into a user-defined bus</td>
<td>UM-149</td>
</tr>
<tr>
<td></td>
<td>Tree window hierarchical view</td>
<td>UM-149</td>
</tr>
<tr>
<td></td>
<td>Main window</td>
<td>UM-151</td>
</tr>
<tr>
<td></td>
<td>Workspace</td>
<td>UM-152</td>
</tr>
<tr>
<td></td>
<td>Transcript</td>
<td>UM-153</td>
</tr>
<tr>
<td></td>
<td>The Main window menu bar</td>
<td>UM-154</td>
</tr>
<tr>
<td></td>
<td>The Main window toolbar</td>
<td>UM-160</td>
</tr>
<tr>
<td></td>
<td>The Main window status bar</td>
<td>UM-162</td>
</tr>
<tr>
<td></td>
<td>Mouse and keyboard shortcuts</td>
<td>UM-162</td>
</tr>
<tr>
<td></td>
<td>Link to active cursor in Wave window</td>
<td>UM-166</td>
</tr>
<tr>
<td></td>
<td>Dataflow window menu bar</td>
<td>UM-167</td>
</tr>
<tr>
<td></td>
<td>Tracing HDL items with the Dataflow window</td>
<td>UM-168</td>
</tr>
<tr>
<td></td>
<td>Saving the Dataflow window as a Postscript file</td>
<td>UM-169</td>
</tr>
<tr>
<td>List window</td>
<td>HDL items you can view</td>
<td>UM-170</td>
</tr>
<tr>
<td></td>
<td>The List window menu bar</td>
<td>UM-171</td>
</tr>
<tr>
<td></td>
<td>Setting List window display properties</td>
<td>UM-173</td>
</tr>
<tr>
<td></td>
<td>Adding HDL items to the List window</td>
<td>UM-175</td>
</tr>
<tr>
<td></td>
<td>Editing and formatting HDL items in the List window</td>
<td>UM-176</td>
</tr>
<tr>
<td></td>
<td>Examining simulation results with the List window</td>
<td>UM-179</td>
</tr>
<tr>
<td></td>
<td>Finding items by name in the List window</td>
<td>UM-180</td>
</tr>
<tr>
<td></td>
<td>Searching for item values in the List window</td>
<td>UM-180</td>
</tr>
<tr>
<td></td>
<td>Setting time markers in the List window</td>
<td>UM-182</td>
</tr>
<tr>
<td></td>
<td>List window keyboard shortcuts</td>
<td>UM-182</td>
</tr>
<tr>
<td></td>
<td>Saving List window data to a file</td>
<td>UM-183</td>
</tr>
<tr>
<td>Process window</td>
<td>The Process window menu bar</td>
<td>UM-184</td>
</tr>
<tr>
<td>Signals window</td>
<td>The Signals window menu bar</td>
<td>UM-187</td>
</tr>
<tr>
<td></td>
<td>Selecting HDL item types to view</td>
<td>UM-188</td>
</tr>
<tr>
<td></td>
<td>Forcing signal and net values</td>
<td>UM-189</td>
</tr>
<tr>
<td></td>
<td>Adding HDL items to the Wave and List windows or a WLF file</td>
<td>UM-190</td>
</tr>
<tr>
<td></td>
<td>Finding HDL items in the Signals window</td>
<td>UM-191</td>
</tr>
<tr>
<td></td>
<td>Setting signal breakpoints</td>
<td>UM-192</td>
</tr>
<tr>
<td></td>
<td>Defining clock signals</td>
<td>UM-194</td>
</tr>
<tr>
<td>Source window</td>
<td>The Source window menu bar</td>
<td>UM-195</td>
</tr>
<tr>
<td></td>
<td>The Source window toolbar</td>
<td>UM-196</td>
</tr>
<tr>
<td></td>
<td>Setting file-line breakpoints</td>
<td>UM-198</td>
</tr>
<tr>
<td></td>
<td>Editing the source file in the Source window</td>
<td>UM-199</td>
</tr>
<tr>
<td></td>
<td>Checking HDL item values and descriptions</td>
<td>UM-202</td>
</tr>
<tr>
<td></td>
<td>Finding and replacing in the Source window</td>
<td>UM-202</td>
</tr>
<tr>
<td></td>
<td>Setting tab stops in the Source window</td>
<td>UM-203</td>
</tr>
<tr>
<td>Structure window</td>
<td>The Structure window menu bar</td>
<td>UM-204</td>
</tr>
<tr>
<td></td>
<td>Finding items in the Structure window</td>
<td>UM-205</td>
</tr>
<tr>
<td>Variables window</td>
<td></td>
<td>UM-206</td>
</tr>
</tbody>
</table>
# Table of Contents

The Variables window menu bar .................................................. UM-208

Wave window ................................................................. UM-210
- Pathname pane .............................................................. UM-210
- Values pane ................................................................. UM-211
- Waveform pane ............................................................. UM-211
- Cursor panes ............................................................... UM-212
- HDL items you can view ..................................................... UM-212
- Adding HDL items in the Wave window ................................. UM-213
- The Wave window menu bar ............................................... UM-214
- The Wave window toolbar ................................................ UM-218
- Using Dividers ............................................................. UM-221
- Splitting Wave window panes ............................................. UM-222
- Combining items in the Wave window ................................ UM-223
- Editing and formatting HDL items in the Wave window .......... UM-224
- Setting Wave window display properties ............................ UM-229
- Sorting a group of HDL items ............................................. UM-230
- Setting signal breakpoints ............................................... UM-230
- Finding items by name or value in the Wave window ............ UM-231
- Searching for item values in the Wave window ..................... UM-231
- Using time cursors in the Wave window ............................... UM-233
- Finding a cursor ................................................................ UM-234
- Making cursor measurements ............................................. UM-234
- Zooming - changing the waveform display range ................. UM-234
- Saving zoom range and scroll position with bookmarks ......... UM-236
- Wave window mouse and keyboard shortcuts ....................... UM-238
- Printing and saving waveforms ......................................... UM-239

Compiling with the graphic interface ..................................... UM-244
- Locating source errors during compilation ........................ UM-245
- Setting default compile options ........................................ UM-246

Simulating with the graphic interface .................................... UM-250
- Design selection tab ....................................................... UM-251
- VHDL settings tab ......................................................... UM-253
- Verilog settings tab ....................................................... UM-255
- Libraries settings tab ..................................................... UM-256
- SDF settings tab .......................................................... UM-257
- SDF options .................................................................. UM-258
- Setting default simulation options .................................... UM-259

ModelSim tools ................................................................. UM-263
- The Button Adder .......................................................... UM-263
- The Macro Helper .......................................................... UM-264
- The Tcl Debugger ........................................................... UM-265
- The GUI Expression Builder ............................................ UM-269

Graphic interface commands ................................................. UM-271

Customizing the interface ..................................................... UM-273

## 9 - Performance Analyzer (UM-275)

Introducing Performance Analysis ........................................... UM-276
10 - Code Coverage (UM-285)

Enabling Code Coverage ............................... UM-286
The coverage_summary window ......................... UM-286
  Summary information ................................. UM-287
  Misses tab ........................................ UM-287
  Exclusions tab ..................................... UM-287
  The coverage_summary window menu bar ............... UM-288
The coverage_source window ............................. UM-290
  Excluding lines and files .......................... UM-291
Merging coverage report files ......................... UM-292
Exclusion filter files ................................ UM-293
  Syntax ........................................... UM-293
  Arguments ....................................... UM-293
  Example ......................................... UM-293
  Default filter file ............................... UM-293
Code Coverage preference variables .................... UM-294
Code Coverage commands ............................... UM-294

11 - Waveform Comparison (UM-295)

Introduction .......................................... UM-296
  Two Modes of Comparison ............................ UM-297
  Comparing Hierarchical and Flattened Designs .... UM-298
Graphical Interface to Waveform Comparison .......... UM-299
  Opening Dataset Comparison ....................... UM-299
  Adding Signals, Regions and/or Clocks ............. UM-301
  Setting Compare Options .......................... UM-308
  Wave window display ................................ UM-310
  Printing compare differences ...................... UM-315
  List window display ............................... UM-316
Waveform Comparison preference variables ............ UM-317
Waveform Comparison commands ........................ UM-317
# 12 - Standard Delay Format (SDF) Timing Annotation (UM-319)

- Specifying SDF files for simulation .......................................................... UM-320
- Instance specification ................................................................. UM-320
- SDF specification with the GUI ........................................................ UM-321
- Errors and warnings ........................................................................ UM-321
- VHDL VITAL SDF ........................................................................ UM-322
- SDF to VHDL generic matching ........................................................ UM-322
- Resolving errors ................................................................................ UM-323
- Verilog SDF ....................................................................................... UM-324
- The $sdf_annotate system task ......................................................... UM-324
- SDF to Verilog construct matching .................................................. UM-325
- Optional edge specifications ............................................................. UM-327
- Optional conditions ........................................................................ UM-328
- Rounded timing values ..................................................................... UM-329
- SDF for Mixed VHDL and Verilog Designs ........................................ UM-330
- Interconnect delays .......................................................................... UM-330
- Troubleshooting ................................................................................ UM-331
- Specifying the wrong instance ......................................................... UM-331
- Mistaking a component or module name for an instance label .......... UM-332
- Forgetting to specify the instance .................................................... UM-332

# 13 - Value Change Dump (VCD) Files (UM-333)

- ModelSim VCD commands and VCD tasks ........................................ UM-334
- Resimulating a design from a VCD file .............................................. UM-336
  - Example 1 — Verilog counter ........................................................ UM-336
  - Example 2 — VHDL adder ............................................................. UM-336
  - Example 3 — Mixed-HDL design ................................................ UM-337
- A VCD file from source to output ..................................................... UM-338
  - VHDL source code .................................................................... UM-338
  - VCD simulator commands ........................................................ UM-338
  - VCD output ................................................................................ UM-339
- Capturing port driver data ............................................................... UM-341
  - Supported TSSI states ............................................................... UM-341
  - Strength values ........................................................................ UM-342
  - Port identifier code ................................................................ UM-342
  - Example VCD output from vcd dumpports ................................ UM-343

# 14 - Logic Modeling SmartModels (UM-345)

- VHDL SmartModel interface ............................................................ UM-346
- Creating foreign architectures with sm_entity ................................ UM-347
- Vector ports ................................................................................ UM-349
- Command channel ........................................................................ UM-350
- SmartModel Windows .................................................................. UM-351
- Memory arrays ............................................................................. UM-352
Verilog SmartModel interface ................................................................. UM-353
Linking the LMTV interface to the simulator ........................................ UM-353

15 - Logic Modeling Hardware Models (UM-355)

VHDL Hardware Model interface ........................................................ UM-356
Creating foreign architectures with hm_entity ..................................... UM-357
Vector ports .......................................................................................... UM-359
Hardware model commands .................................................................... UM-360

16 - Tcl and macros (UM-361)

Tcl features within ModelSim ............................................................... UM-362
Tcl References ..................................................................................... UM-362
Tcl tutorial ............................................................................................ UM-362
Tcl commands ...................................................................................... UM-363
Tcl command syntax ............................................................................ UM-364
  if command syntax ............................................................................ UM-366
  set command syntax ......................................................................... UM-367
  Command substitution .......................................................................... UM-367
  Command separator ............................................................................. UM-368
  Multiple-line commands ...................................................................... UM-368
  Evaluation order ................................................................................ UM-368
  Tcl relational expression evaluation ................................................ UM-368
  Variable substitution .......................................................................... UM-369
  System commands ............................................................................... UM-369
List processing ...................................................................................... UM-370
ModelSim Tcl commands ........................................................................ UM-370
ModelSim Tcl time commands ................................................................ UM-371
  Conversions ....................................................................................... UM-371
  Relations ............................................................................................ UM-371
  Arithmetic ........................................................................................... UM-372
Tcl examples ......................................................................................... UM-373
  Example 1 .......................................................................................... UM-373
  Example 2 .......................................................................................... UM-374
Macros (DO files) .................................................................................. UM-377
  Creating DO files .............................................................................. UM-377
  Using Parameters with DO files ....................................................... UM-377
  Useful commands for handling breakpoints and errors ....................... UM-378
  Error action in DO files ..................................................................... UM-378

A - ModelSim Variables (UM-381)

Variable settings report ......................................................................... UM-382
Personal preferences ............................................................................. UM-382
Returning to the original ModelSim defaults ........................................ UM-382
Environment variables ................................................ UM-383
ModelSim Environment Variables ................................. UM-383
Creating environment variables in Windows .................. UM-384
Referencing environment variables within ModelSim ...... UM-385
Removing temp files (VSOUT) .................................... UM-385

Preference variables located in INI files ....................... UM-386
[Library] library path variables ................................. UM-386
[vcom] VHDL compiler control variables ...................... UM-386
[vlog] Verilog compiler control variables ................... UM-388
[vsim] simulator control variables ............................. UM-388
[lmc] Logic Modeling variables ................................. UM-392
Setting variables in INI files ................................... UM-392
Reading variable values from the INI file .................... UM-392
Commonly used INI variables ................................... UM-393

Preference variables located in TCL files ..................... UM-396
User-defined variables ................................. UM-396
More preferences ....................................... UM-396

Variable precedence ........................................ UM-397

Simulator state variables ................................ UM-398
Referencing simulator state variables ....................... UM-398
Special considerations for $now ............................... UM-399

B - ModelSim Shortcuts (UM-401)

Wave window mouse and keyboard shortcuts ................. UM-401
List window keyboard shortcuts ............................. UM-402
Command shortcuts .................................. UM-403
Command history shortcuts ............................. UM-403
Mouse and keyboard shortcuts in Main and Source windows UM-403
Right mouse button .................................. UM-405

C - Using the FLEXlm License Manager (UM-407)

Starting the license server daemon ..................... UM-408
Locating the license file ............................... UM-408
Controlling the license file search ..................... UM-408
Manual start ...................................... UM-409
Automatic start at boot time ............................. UM-409
What to do if another application uses FLEXlm ........ UM-409

Format of the license file ................................ UM-410
Format of the daemon options file ..................... UM-410

License administration tools ............................. UM-412
lmstat ....................................... UM-412
lmdown ....................................... UM-412
lmremove ..................................... UM-412
lmreread ...................................... UM-413
D - Tips and Techniques (UM-415)

- How to use checkpoint/restore
- The difference between checkpoint/restore and restarting
- Using macros with restart and checkpoint/restore
- Running command-line and batch-mode simulations
  - Command-line mode
  - Batch mode
- Source code security and -nodebug
- Saving and viewing waveforms
- Setting up libraries for group use
- Maintaining 32-bit and 64-bit modules in the same library
- Bus contention checking
- Bus float checking
- Design stability checking
- Toggle checking
- Detecting infinite zero-delay loops
- Referencing source files with location maps
  - Using location mapping
  - Pathname syntax
  - How location mapping works
  - Mapping with Tcl variables

E - What’s new in ModelSim (UM-437)

- New features
- Command and variable changes
- Documentation changes
- GUI changes in version 5.5

ModelSim SE User’s Manual
License Agreement (UM-453)

Index (UM-323)
1 - Introduction

Chapter contents

ModelSim graphic interface . . . . . . . . . . . . . . . . . . . . . . . . . UM-15
Standards supported . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-16
Assumptions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-16
Sections in this document . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-16
Command reference . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-18
Text conventions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-18
What is an "HDL item" . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-19
Where to find our documentation . . . . . . . . . . . . . . . . . . . . . . . . . UM-19
Technical support and updates . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-20

This documentation was written for ModelSim SE version 5.5d for UNIX and Microsoft Windows 95/98/Me/NT/2000 (see note below for exception). If the ModelSim software you are using is a later release, check the README file that accompanied the software. Any supplemental information will be there.

Although this document covers both VHDL and Verilog simulation, you will find it a useful reference for single HDL design work.

ModelSim graphic interface

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

- SPARCstation with OpenWindows, OSF/Motif, or CDE
- IBM RISC System/6000 with OSF/Motif
- Hewlett-Packard HP 9000 Series 700 with HP VUE, OSF/Motif, or CDE
- Linux (Red Hat v. 6 / 7) with KDE or GNOME
- Microsoft Windows 95/98/Me/NT/2000

Because ModelSim’s graphic interface is based on Tcl/TK, you also have the tools to build your own simulation environment. Preference variables and configuration commands, "Preference variables located in INI files" (UM-386), and "Graphic interface commands" (UM-271) give you control over the use and placement of windows, menus, menu options and buttons. See "Tcl and macros" (UM-361) for more information on Tcl.

For an in-depth look at ModelSim’s graphic interface, see Chapter 8 - Graphic Interface.
Standards supported

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

ModelSim Verilog is based on IEEE Std 1364-1995 and a partial implementation of 1364-2001, Standard Hardware Description Language Based on the Verilog Hardware Description Language. The Open Verilog International Verilog LRM version 2.0 is also applicable to a large extent. Both PLI (Programming Language Interface) and VCD (Value Change Dump) are supported for ModelSim PE and SE users.

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

Assumptions

We assume that you are familiar with the use of your operating system. You should also be familiar with the window management functions of your graphic interface: either OpenWindows, OSF/Motif, CDE, HP VUE, KDE, GNOME, or Microsoft Windows 95/98/Me/NT/2000.

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

Finally, we make the assumption that you have worked the appropriate lessons in the ModelSim Tutorial or the Quick Start and are therefore familiar with the basic functionality of ModelSim. The ModelSim Tutorial and Quick Start are both available from the ModelSim Help menu. The ModelSim Tutorial is also available from the Support page of our web site: www.model.com.

For installation instructions please refer to the Start Here for ModelSim guide that was shipped with the ModelSim CD. Start Here may also be downloaded from our website: www.model.com.

Sections in this document

In addition to this introduction, you will find the following major sections in this document:

2 - Projects and system initialization (UM-21)
This chapter provides a definition of a ModelSim "project" and discusses the use of a new file extension for project files.

3 - Design libraries (UM-37)
To simulate an HDL design using ModelSim, you need to know how to create, compile, maintain, and delete design libraries as described in this chapter.
4 - VHDL Simulation (UM-51)
This chapter is an overview of compilation and simulation for VHDL within the ModelSim environment.

5 - Verilog Simulation (UM-67)
This chapter is an overview of compilation and simulation for Verilog within the ModelSim environment.

6 - Mixed VHDL and Verilog Designs (UM-121)
ModelSim/Plus single-kernel simulation (SKS) allows you to simulate designs that are written in VHDL and/or Verilog. This chapter outlines data mapping and the criteria established to instantiate design units between HDLs.

7 - WLF files (datasets) and virtuals (UM-131)
This chapter describes datasets and virtuals - both methods for viewing and organizing simulation data in ModelSim.

8 - Graphic Interface (UM-143)
This chapter describes the graphic interface available while operating ModelSim. ModelSim’s graphic interface is designed to provide consistency throughout all operating system environments.

9 - Performance Analyzer (UM-275)
This chapter describes how the ModelSim Performance Analyzer is used to easily identify areas in your simulation where performance can be improved.

10 - Code Coverage (UM-285)
This chapter describes the Code Coverage feature. Code Coverage gives you graphical and report file feedback on how the source code is being executed.

11 - Waveform Comparison (UM-295)
This chapter describes Waveform Comparison, a feature that lets you compare simulations.

12 - Standard Delay Format (SDF) Timing Annotation (UM-319)
This chapter discusses ModelSim’s implementation of SDF (Standard Delay Format) timing annotation. Included are sections on VITAL SDF and Verilog SDF, plus troubleshooting.

13 - Value Change Dump (VCD) Files (UM-333)
This chapter explains Model Technology’s Verilog VCD implementation for ModelSim. The VCD usage is extended to include VHDL designs.

14 - Logic Modeling SmartModels (UM-345)
This chapter describes the use of the SmartModel Library and SmartModel Windows with ModelSim.

15 - Logic Modeling Hardware Models (UM-355)
This chapter describes the use the Logic Modeling Hardware Modeler with ModelSim.
16 - Tcl and macros (UM-361)
This chapter provides an overview of Tcl (tool command language) as used with ModelSim.

A - ModelSim Variables (UM-381)
This appendix describes environment, system, and preference variables used in ModelSim.

B - ModelSim Shortcuts (UM-401)
This appendix describes ModelSim keyboard and mouse shortcuts.

C - Using the FLEXlm License Manager (UM-407)
This appendix covers Model Technology’s application of FLEXlm for ModelSim licensing.

D - Tips and Techniques (UM-415)
This appendix contains an extended collection of ModelSim usage examples taken from our manuals, and tech support solutions.

E - What’s new in ModelSim (UM-437)
This appendix lists new features and changes in the various versions of ModelSim.

Command reference

The complete command reference for all ModelSim commands is located in the ModelSim Command Reference. Command Reference cross reference page numbers are prefixed with "CR" (e.g., "ModelSim Commands" (CR-29)).

Text conventions

Text conventions used in this manual include:

<table>
<thead>
<tr>
<th>Text</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>italic text</td>
<td>provides emphasis and sets off filenames, path names, and design unit names</td>
</tr>
<tr>
<td>bold text</td>
<td>indicates commands, command options, menu choices, package and library logical names, as well as variables and dialog box selection</td>
</tr>
<tr>
<td>monospace type</td>
<td>monospace type is used for program and command examples</td>
</tr>
<tr>
<td>The right angle (&gt;)</td>
<td>is used to connect menu choices when traversing menus as in: File &gt; Save</td>
</tr>
<tr>
<td>path separators</td>
<td>examples will show either UNIX or Windows path separators - use separators appropriate for your operating system when trying the examples</td>
</tr>
<tr>
<td>UPPER CASE</td>
<td>denotes file types used by ModelSim (e.g., DO, WLF, INI, MPF, PDF, etc.)</td>
</tr>
</tbody>
</table>
What is an "HDL item"

Because ModelSim works with both VHDL and Verilog, “HDL” refers to either VHDL or Verilog when a specific language reference is not needed. Depending on the context, “HDL item” can refer to any of the following:

<table>
<thead>
<tr>
<th>VHDL</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>block statement, component instantiation, constant, generate statement, generic, package, signal, or variable</td>
<td>function, module instantiation, named fork, named begin, net, task, or register variable</td>
</tr>
</tbody>
</table>

Where to find our documentation

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

<table>
<thead>
<tr>
<th>Document</th>
<th>Format</th>
<th>How to get it</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Here for ModelSim SE (installation &amp; support reference)</td>
<td>paper</td>
<td>shipped with ModelSim</td>
</tr>
<tr>
<td></td>
<td>PDF</td>
<td>select Main window &gt; Help &gt; SE Documentation; also available from the Support page of our web site: <a href="http://www.model.com">www.model.com</a></td>
</tr>
<tr>
<td>ModelSim SE Quick Guide (command and feature quick-reference)</td>
<td>paper</td>
<td>shipped with ModelSim</td>
</tr>
<tr>
<td></td>
<td>PDF</td>
<td>select Main window &gt; Help &gt; SE Documentation; also available from the Support page of our web site: <a href="http://www.model.com">www.model.com</a></td>
</tr>
<tr>
<td>ModelSim SE Tutorial</td>
<td>PDF, HTML</td>
<td>select Main window &gt; Help &gt; SE Documentation; also available from the Support page of our web site: <a href="http://www.model.com">www.model.com</a></td>
</tr>
<tr>
<td>ModelSim SE User’s Manual</td>
<td>PDF, HTML</td>
<td>select Main window &gt; Help &gt; SE Documentation</td>
</tr>
<tr>
<td>ModelSim SE Command Reference</td>
<td>PDF, HTML</td>
<td>select Main window &gt; Help &gt; SE Documentation</td>
</tr>
<tr>
<td>ModelSim Foreign Language Interface Reference</td>
<td>PDF, HTML</td>
<td>select Main window &gt; Help &gt; SE Documentation</td>
</tr>
<tr>
<td>ModelSim Command Help</td>
<td>ASCII</td>
<td>type help [command name] at the prompt in the Main window</td>
</tr>
<tr>
<td>Tcl Man Pages (Tcl manual)</td>
<td>HTML</td>
<td>select Main window &gt; Help &gt; Tcl Man Pages, or find contents.htm in \modeltech\docs\tcl_help_html</td>
</tr>
</tbody>
</table>
Download a free PDF reader with Search

Model Technology’s PDF documentation requires an Adobe Acrobat Reader for viewing. The Reader may be installed from the ModelSim CD. It is also available without cost from Adobe at http://www.adobe.com. Be sure to download the Acrobat Reader with Search to take advantage of the index file supplied with our documentation; the index makes searching for key words much faster.

Technical support and updates

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

Support

www.model.com/support/default.asp

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

Updates

www.model.com/products/release.asp

Latest version email

Place your name on our list for email notification of news and updates. model.com/support/register_news_list.asp
# 2 - Projects and system initialization

This chapter discusses ModelSim projects. Projects simplify the process of compiling and simulating a design and are a great tool for getting started with ModelSim. This chapter also includes a section on ModelSim initialization.

## Chapter contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>UM-22</td>
</tr>
<tr>
<td>What are projects?</td>
<td>UM-22</td>
</tr>
<tr>
<td>What are the benefits of projects?</td>
<td>UM-22</td>
</tr>
<tr>
<td>How do projects differ in version 5.5?</td>
<td>UM-23</td>
</tr>
<tr>
<td>Getting started with projects</td>
<td>UM-24</td>
</tr>
<tr>
<td>Step 1 — Create a new project</td>
<td>UM-25</td>
</tr>
<tr>
<td>Step 2 — Add files to the project</td>
<td>UM-27</td>
</tr>
<tr>
<td>Step 3 — Compile the files</td>
<td>UM-27</td>
</tr>
<tr>
<td>Step 4 — Simulate a design</td>
<td>UM-27</td>
</tr>
<tr>
<td>Other project operations</td>
<td>UM-29</td>
</tr>
<tr>
<td>Customizing project settings</td>
<td>UM-30</td>
</tr>
<tr>
<td>Changing compile order</td>
<td>UM-30</td>
</tr>
<tr>
<td>Setting compiler options</td>
<td>UM-31</td>
</tr>
<tr>
<td>Accessing projects from the command line</td>
<td>UM-32</td>
</tr>
<tr>
<td>System initialization</td>
<td>UM-33</td>
</tr>
<tr>
<td>Files accessed during startup</td>
<td>UM-33</td>
</tr>
<tr>
<td>Environment variables accessed during startup</td>
<td>UM-34</td>
</tr>
<tr>
<td>Initialization sequence</td>
<td>UM-35</td>
</tr>
</tbody>
</table>

This chapter discusses ModelSim projects. Projects simplify the process of compiling and simulating a design and are a great tool for getting started with ModelSim. This chapter also includes a section on ModelSim initialization.


Introduction

What are projects?

Projects are collection entities for HDL designs under specification or test. At a minimum projects have a root directory, a work library, and "metadata" which are stored in a .mpf file located in a project's root directory. The metadata include compiler switch settings, compile order, and file mappings. Projects 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

What are the benefits of projects?

Projects offer benefits to both new and advanced users. Projects

- simplify interaction with ModelSim; you don’t need to understand the intricacies of compiler switches and library mappings
- eliminate the need to remember a conceptual model of the design; the compile order is maintained for you in the project
- remove the necessity to re-establish compiler switches and settings at each session; these are stored in the project metadata as are mappings to HDL source files
- allow users to share libraries without copying files to a local directory; you can establish references to source files that are stored remotely or locally
- allow you to change individual parameters across multiple files; in previous versions you could only set parameters one file at a time
- enable "what-if" analysis; you can copy a project, manipulate the settings, and rerun it to observe the new results
- reload .ini variable settings every time the project is opened; in previous versions you had to quit ModelSim and restart the program to read in a new .ini file
How do projects differ in version 5.5?

Projects have improved a great deal from earlier versions. Some of the key differences include:

- A new interface eliminates the need to write custom scripts.
- You don’t have to copy files into a specific directory; you can establish references to files in any location.
- You don’t have to specify compiler switches; the automatic defaults will work for many designs. However, if you do want to customize the settings, you do it through a dialog box rather than by writing a script.
- All metadata (compiler settings, compile order, file mappings) are stored in the project .mpf file.

⚠️ **Important:** Due to the significant changes, projects created in versions prior to 5.5 cannot be converted automatically. If you created a project in an earlier version, you will need to recreate it in version 5.5. With the new interface even the most complex project should take less than 15 minutes to recreate. Follow the instructions in the ensuing pages to recreate your project.
Getting started with projects

This section describes the four basic steps to working with a project. For a discussion of more advanced project features, see "Customizing project settings" (UM-30).

Step 1 — Create a new project (UM-25)
This creates a .mpf file and a working library.

Step 2 — Add files to the project (UM-27)
Projects can reference or include HDL source files and any other files you want to associate with the project. You can copy files into the project directory or simply create mappings to files in other locations.

Step 3 — Compile the files (UM-28)
This checks syntax and semantics and creates the pseudo machine code ModelSim uses for simulation.

Step 4 — Simulate a design (UM-29)
This specifies the design unit you want to simulate and opens a structure tab in the workspace.
Step 1 — Create a new project

1. Select **Create a Project** from the Welcome to ModelSim screen that opens the first time you start ModelSim. If this screen is not available, you can enable it by selecting **Help > Enable Welcome** (Main window).

![Create a Project dialog box](image)

You can also use the **File > New > Project** (Main window) command to create a new project.

2. Clicking the **Create a Project** button opens the Create Project dialog box.

![Create Project dialog box](image)
3 Specify a **Project Name** and **Project Location**. The location is where the project .mpf file and any copied source files will be stored. You can leave the Default Library Name set to "work," or specify a different name if desired. The name that is specified will be used to create a working library subdirectory within the Project Location.

After selecting OK, you will see a blank Project tab in the workspace area of the Main window. You can hide or show the workspace at any time using the **View > Hide/Show Workspace** command.

The name of the current project is shown at the bottom left corner of the Main window.
Step 2 — Add files to the project

Your right mouse button (2nd button in Windows; 3rd button in UNIX) gives quick access to project commands. When you right-click in the workspace, a context menu appears. The menu that appears depends on where you click in the workspace.

1 Right click in a blank area on the Project tab and select Add file to Project. This opens the Add file to Project dialog. You can also select Project > Add file to Project from the menu bar.

2 Specify one or more files you want to add to the project. (The files used in this example are available in the examples directory that is installed along with ModelSim.)

3 For the files you’re adding, choose whether to reference them from their current location or copy them into the project directory.
Step 3 — Compile the files

1. To compile the files, right click in the Project tab and select **Compile All**. You can also select **Project > Compile All** from the menu bar.

An asterisk next to a file denotes that that file has changed since the last compilation. **Compile Out-of-Date Files** will compile only files that have changed.

2. Once compilation is finished, click the Library tab and you’ll see the two compiled designs.
Step 4 — Simulate a design

1. To simulate one of the designs, either double-click the name or right click the name and select Load. A new tab appears showing the structure of the current active simulation.

At this point you are ready to run the simulation and analyze your results. You often do this by adding signals to the Wave window and running the simulation for a given period of time. See the ModelSim Tutorial for examples.

Other project operations

In addition to the four actions just discussed, the following are common project operations.

Open an existing project

When you leave a ModelSim session, ModelSim will remember the last opened project. You can reopen it for your next session by clicking Open Project in the Welcome to ModelSim dialog. You can also open an existing project by selecting File > Open > Project (Main window).

Close a project

Select File > Close > Project (Main window). This closes the Project tab but leaves the Library and Structure (labeled "Sim" in the graphic above) tabs open in the workspace.

Delete a project

Select File > Delete > Project (Main window).
Customizing project settings

Though the default project settings will work for many designs, it is easy to customize the settings if needed.

Changing compile order

When you compile all files in a project, ModelSim by default compiles the files in the order in which they were added to the project. You have two alternatives for changing the default compile order: 1) select and compile each file individually; 2) specify a custom compile order.

To specify a custom compile order, follow these steps:

1. Right click in an empty area of the Project tab and select Sort by Compile Order.
2. Drag the files into the correct order. Note that you can select multiple files and drag them simultaneously.

Note: Files can be displayed in the Project tab in alphabetical or compile order (using the Sort by Alphabetical Order or Sort by Compile Order commands on the context menu). Keep in mind that the order you see in the Project tab is not necessarily the order in which the files will be compiled.

Grouping files

You can group two or more files in the Project tab. You might do this for organizational purposes or to send the files to the compiler at the same time. For example, you might have one file with a bunch of Verilog define statements and a second file that is a Verilog module. You would want to compile these two files at the same time.

To group files, follow these steps:

1. Right click in an empty area of the Project tab and select Sort by Compile Order.
2. Select the files you want to group.
3. Right click one of the selected files and select Group.
Setting compiler options

The VHDL and Verilog compilers (vcom and vlog, respectively) have numerous options that affect how a design is compiled and subsequently simulated. Outside of a project you can set the defaults for all future simulations using the **Options > Compile** (Main window) command. Inside of a project you can set these options on individual files or a group of files.

To set the compiler options in a project, select the file(s) in the Project tab, right click on the file names, and select **Properties**. The resulting dialog varies depending on the number and type of files you have selected. If you select a single VHDL or Verilog file, you’ll see the General tab and the VHDL or Verilog tab, respectively. On the General tab, you’ll see file properties such as Type, Path, and Size. If you select multiple files, the file properties on the General tab are not listed. Finally, if you select both a VHDL file and a Verilog file, you’ll see all three tabs but no file information on the General tab.

The General tab includes these options:

- **Exclude File from Build**
  Determines whether the file is excluded from the compile.

- **Compile to library**
  Specifies to which library you want to compile the file; defaults to the working library.
• **File Properties**
  A variety of information about the selected file (e.g., type, size, path). Displays only if a single file is selected in the Project tab.

The definitions of the options on the VHDL and Verilog tabs can be found in the section "Setting default compile options" (UM-246).

When setting options on a group of files, keep in mind the following:

• If two or more files have different settings for the same option, the checkbox in the dialog will be "grayed out." If you change the option, you cannot change it back to a "multi-state setting" without cancelling out of the dialog. Once you click OK, ModelSim will set the option the same for all selected files.

• If you select a combination of VHDL and Verilog files, the options you set on the VHDL and Verilog tabs apply only to those file types.

**Accessing projects from the command line**

Generally, projects are used only within the ModelSim graphical user interface. However, standalone tools will use the project file if they are invoked in the project's root directory. If invoked outside the project directory, the MODELSIM environment variable can be set with the path to the project file (<Project_Root_Dir>/<Project_Name>.mpf).

You can also use the `project` command (CR-182) from the command line to perform common operations on new projects. The command is to be used outside of a simulation session.
System initialization

ModelSim goes through numerous steps as it initializes the system during startup. It accesses various files and environment variables to determine library mappings, configure the GUI, check licensing, and so forth.

Files accessed during startup

The table below describes the files that are read during startup. They are listed in the order in which they are accessed.

<table>
<thead>
<tr>
<th>File</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelsim.ini</td>
<td>contains initial tool settings; see &quot;Preference variables located in INI files&quot; (UM-386) for specific details on the modelsim.ini file</td>
</tr>
<tr>
<td>location map file</td>
<td>used by ModelSim tools to find source files based on easily reallocated &quot;soft&quot; paths; default file name is mgc_location_map; see &quot;How location mapping works&quot; (UM-426) for more details</td>
</tr>
<tr>
<td>pref.tcl</td>
<td>contains defaults for fonts, colors, prompts, window positions, and other simulator window characteristics; see &quot;Preference variables located in TCL files&quot; (UM-396) for specific details on the pref.tcl file</td>
</tr>
<tr>
<td>modelsim.tcl</td>
<td>contains user-customized settings for fonts, colors, prompts, window positions, and other simulator window characteristics; see &quot;Preference variables located in TCL files&quot; (UM-396) for specific details on the modelsim.tcl file</td>
</tr>
</tbody>
</table>
Environment variables accessed during startup

The table below describes the environment variables that are read during startup. They are listed in the order in which they are accessed. For more information on environment variables, see "Environment variables" (UM-383).

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODEL_TECH</td>
<td>set by ModelSim to the directory in which the binary executables reside (e.g., ../modeltech/&lt;platform&gt;/)</td>
</tr>
<tr>
<td>MODEL_TECH_OVERRIDE</td>
<td>provides an alternative directory for the binary executables; MODEL_TECH is set to this path</td>
</tr>
<tr>
<td>MODELSIM</td>
<td>identifies path to the modelsim.ini file</td>
</tr>
<tr>
<td>MGC_WD</td>
<td>identifies the Mentor Graphics working directory (set by Mentor Graphics tools)</td>
</tr>
<tr>
<td>MGC_LOCATION_MAP</td>
<td>identifies the path to the location map file; set by ModelSim if not defined</td>
</tr>
<tr>
<td>MODEL_TECH_TCL</td>
<td>identifies the path to all Tcl libraries installed with ModelSim</td>
</tr>
<tr>
<td>HOME</td>
<td>identifies your login directory (UNIX only)</td>
</tr>
<tr>
<td>MGC_HOME</td>
<td>identifies the path to the MGC tool suite</td>
</tr>
<tr>
<td>TCL_LIBRARY</td>
<td>identifies the path to the Tcl library; set by ModelSim to the same path as MODEL_TECH_TCL; must point to libraries supplied by Model Technology</td>
</tr>
<tr>
<td>TK_LIBRARY</td>
<td>identifies the path to the Tk library; set by ModelSim to the same path as MODEL_TECH_TCL; must point to libraries supplied by Model Technology</td>
</tr>
<tr>
<td>TIX_LIBRARY</td>
<td>identifies the path to the Tix library; set by ModelSim to the same path as MODEL_TECH_TCL; must point to libraries supplied by Model Technology</td>
</tr>
<tr>
<td>ITCL_LIBRARY</td>
<td>identifies the path to the [incr]Tcl library; set by ModelSim to the same path as MODEL_TECH_TCL; must point to libraries supplied by Model Technology</td>
</tr>
<tr>
<td>ITK_LIBRARY</td>
<td>identifies the path to the [incr]Tk library; set by ModelSim to the same path as MODEL_TECH_TCL; must point to libraries supplied by Model Technology</td>
</tr>
<tr>
<td>VSIM_LIBRARY</td>
<td>identifies the path to the Tcl files that are used by ModelSim; set by ModelSim to the same path as MODEL_TECH_TCL; must point to libraries supplied by Model Technology</td>
</tr>
<tr>
<td>MTI_LIB_DIR</td>
<td>identifies the path to all Tcl libraries installed with ModelSim</td>
</tr>
</tbody>
</table>
 Initialization sequence

The following list describes in detail ModelSim’s initialization sequence. The sequence includes a number of conditional structures, the results of which are determined by the existence of certain files and the current settings of environment variables.

In the steps below, names in uppercase denote environment variables (except MTI_LIB_DIR which is a Tcl variable). Instances of $(NAME) denote paths that are determined by an environment variable (except $(MTI_LIB_DIR) which is determined by a Tcl variable).

1 Determines the path to the executable directory (../modeltech/<platform>/). Sets MODEL_TECH to this path, unless MODEL_TECH_OVERRIDE exists, in which case MODEL_TECH is set to the same value as MODEL_TECH_OVERRIDE.

2 Finds the modelsim.ini file by evaluating the following conditions:
   - use MODELSIM if it exists; else
   - use $(MGC_WD)/modelsim.ini; else
   - use ./modelsim.ini; else
   - use $(MODEL_TECH)/modelsim.ini; else
   - use $(MGC_HOME)/lib/modelsim.ini; else
   - set path to ./modelsim.ini even though the file doesn’t exist

3 Finds the location map file by evaluating the following conditions:
   - use MGC_LOCATION_MAP if it exists (if this variable is set to "no_map", ModelSim skips initialization of the location map); else
   - use mge_location_map if it exists; else
   - use $(HOME)/mge/mge_location_map; else
   - use $(HOME)/mge_location_map; else
   - use $(MGC_HOME)/etc/mge_location_map; else
   - use $(MGC_HOME)/shared/etc/mge_location_map; else
   - use $(MODEL_TECH)/mge_location_map; else
   - use $(MODEL_TECH)/./mge_location_map; else
   - use no map

4 Reads various variables from the [vsim] section of the modelsim.ini file. See "[vsim] simulator control variables" (UM-388) for more details.
5 Parses any command line arguments that were included when you started ModelSim and reports any problems.

6 Defines the following environment variables:
   • use MODEL_TECH_TCL if it exists; else
   • set MODEL_TECH_TCL=${MODEL_TECH}/../tcl
   • set TCL_LIBRARY=${MODEL_TECH_TCL}/tcl8.0
   • set TK_LIBRARY=${MODEL_TECH_TCL}/tk8.0
   • set TIX_LIBRARY=${MODEL_TECH_TCL}/tix4.1
   • set ITCL_LIBRARY=${MODEL_TECH_TCL}/itcl3.0
   • set ITK_LIBRARY=${MODEL_TECH_TCL}/itk3.0
   • set VSIM_LIBRARY=${MODEL_TECH_TCL}/vsim

7 Initializes the simulator’s Tcl interpreter.

8 Checks for a valid license (a license is not checked out unless specified by a modelsim.ini setting or command line option).

The next four steps relate to initializing the graphical user interface.

9 Sets Tcl variable "MTI_LIB_DIR"=MODEL_TECH_TCL

10 Loads ${MTI_LIB_DIR}/pref.tcl.

11 Loads last working directory, project init, project history, and printer defaults from the registry (Windows) or ${HOME}/.modelsim (UNIX).

12 Finds the modelsim.tcl file by evaluating the following conditions:
   • use MODELSIM_TCL if it exists (if MODELSIM_TCL is a list of files, each file is loaded in the order that it appears in the list); else
   • use ./modelsim.tcl; else
   • use ${HOME}/modelsim.tcl if it exists

That completes the initialization sequence. Also note the following about the modelsim.ini file:

• When you change the working directory within ModelSim, the tool reads the [library], [vcom], and [vlog] sections of the local modelsim.ini file. When you make changes in the compiler options dialog or use the vmap command, the tool updates the appropriate sections of the file.

• The pref.tcl file references the default .ini file via the [GetPrivateProfileString] Tcl command. The .ini file that is read will be the default file defined at the time pref.tcl is loaded.
VHDL contains libraries, which are objects that contain compiled design units; libraries are given names so they may be referenced. Verilog designs simulated within ModelSim are compiled into libraries as well.
Design library contents

A design library is a directory that serves as a repository for compiled design units. The design units contained in a design library consist of VHDL entities, packages, architectures, and configurations; and Verilog modules and UDPs (user defined primitives). The design units are classified as follows:

- **Primary design units**
  Consist of entities, package declarations, configuration declarations, modules, and UDPs. Primary design units within a given library must have unique names.

- **Secondary design units**
  Consist of architecture bodies and package bodies. Secondary design units are associated with a primary design unit. Architectures by the same name can exist if they are associated with different entities.

Design unit information

The information stored for each design unit in a design library is:

- retargetable, executable code
- debugging information
- dependency information

Design library types

There are two kinds of design libraries: working libraries and resource libraries. A working library is the library into which a design unit is placed after compilation. A resource library contains design units that can be referenced within the design unit being compiled. Only one library can be the working library; in contrast, any number of libraries (including the working library itself) can be resource libraries during a compilation.

The library named work has special attributes within ModelSim; it is predefined in the compiler and need not be declared explicitly (i.e. library work). It is also the library name used by the compiler as the default destination of compiled design units. In other words the work library is the working library. In all other aspects it is the same as any other library.
Working with design libraries

The implementation of a design library is not defined within standard VHDL or Verilog. Within ModelSim, design libraries are implemented as directories and can have any legal name allowed by the operating system, with one exception; extended identifiers are not supported for library names.

Creating a library

When you create a project (see "Getting started with projects" (UM-24)), ModelSim automatically creates a working design library. If you don’t create a project, you need to create a working design library before you run the compiler. This can be done from either the command line or from the ModelSim graphic interface.

From the ModelSim prompt or a UNIX/DOS prompt, use this `vlib` command (CR-273):

```bash
vlib <directory_pathname>
```

To create a new library with the ModelSim graphic interface, select Design > Create a New Library (Main window). This brings up a dialog box that allows you to specify the library name and its logical mapping.

The Create a New Library dialog box includes these options:

- **Create a new library and a logical mapping to it**
  Type the new library name into the Library Name field. This creates a library subdirectory in your current working directory, initially mapped to itself. Once created, the mapped library is easily remapped to a different library.

- **Create a map to an existing library**
  Type the new library name into the Library Name field, then type into the Library Maps to field or Browse to select a library name for the mapping.

- **Library Name**
  Type the new library name into this field.
• **Library Maps to**
  Type or **Browse** for a mapping for the specified library. This field can be changed only when the **Create a map to an existing library** option is selected.

When you click **OK**, ModelSim creates the specified library directory 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.

The new map entry is written to the modelsim.ini file in the [Library] section. See "[Library] library path variables" (UM-386) for more information.

★ **Note:** Remember that a design library is a special kind of directory; the only way to create a library is to use the ModelSim GUI or the **vlib** command (CR-273). Do not create libraries using UNIX or Windows commands.

**Managing library contents**

Library contents can be viewed, deleted, recompiled, edited and so on using either the graphic interface or command line.

The Library tab in the Main window workspace provides access to design units (configurations, modules, packages, entities, and architectures) in a library. Note the icons identify whether a unit is an entity (E), a module (M), and so forth.

The Library tab includes these options:

• **Library**
  Select the library you wish to view from the drop-down list. Related command line command is **vdir** (CR-247).
• **DesignUnit/Description list**

Select a plus (+) box to view the associated architecture, or select a minus (−) box to hide the architecture.

The Library tab also has two context menus that you access with your right mouse button (Windows—2nd button, UNIX—3rd button). One menu is accessed by right-clicking a design unit name; the second is accessed by right-clicking a blank area in the Library tab. The graphic below shows the two menus.

The context menu at the left includes the following commands:

- **Load**
  Simulates the selected design unit and opens a structure tab in the workspace. Related command line command is `vsim` (CR-284).

- **Edit**
  Opens the selected design unit in the Source window.

- **Refresh**
  Rebuilds the library image of the selected item(s) without using source code. Related command line command is `vcom` (CR-240) with the -refresh argument.

- **Recompile**
  Recompiles the selected design unit. Related command line command is `vcom` (CR-240).

- **Delete**
  Deletes the selected design unit. Related command line command is `vdel` (CR-246).

Deleting a package, configuration, or entity will remove the design unit from the library. If you delete an entity that has one or more architectures, the entity and all its associated architectures will be deleted.
You can also delete an architecture without deleting its associated entity. Expand the entity, right-click the desired architecture name, and select Delete. You are prompted for confirmation before any design unit is actually deleted.

The second context menu has the following options:

- **Load**
  
  Opens the Load Design dialog box. See "Simulating with the graphic interface" (UM-250) for details. Related command line command is `vsim` (CR-284).

- **Create Library**
  
  Opens the Create a New Library dialog box. See "Creating a library" (UM-39) earlier in this chapter for details. Related command line command is `vlib` (CR-273).

- **View**
  
  Provides various options for displaying design units.

- **Update**
  
  Reloads the library in case any of the design units were modified outside of the current session (e.g., by a script or another user).
Assigning a logical name to a design library

VHDL uses logical library names that can be mapped to ModelSim library directories. By default, ModelSim can find libraries in your current directory (assuming they have the right name), but for it to find libraries located elsewhere, you need to map a logical library name to the pathname of the library.

You can use the GUI, a command, or a project to assign a logical name to a design library.

Library mappings with the GUI

To associate a logical name with a library, select Design > Browse Libraries (Main window). This brings up a dialog box that allows you to view, add, edit, and delete mappings, as shown below:

![Library Browser](image)

The Library Browser dialog box includes these options:

- **Show**
  Choose the mapping and library scope to view from the drop-down list.

- **Library/Type list**
  
  To view the contents of a library
  Select the library, then click the View button. This brings up the Library tab (UM-40) in the Main window. From there you can also delete design units from the library.

  To create a new library mapping
  Click the Add button. This brings up Create a New Library (UM-39) dialog box that allows you to enter a new logical library name and the pathname to which it is to be mapped.

  It is possible to enter the name of a non-existent directory, but the specified directory must exist as a ModelSim library before you can compile design units into it. ModelSim will issue a warning message if you try to map to a non-existent directory.
To edit an existing library mapping
Select the desired mapping entry, then click the Edit button. This brings up a dialog box that allows you to modify the logical library name and the pathname to which it is mapped. Selecting Delete removes an existing library mapping, but it does not delete the library. The library can be deleted with this vdel command (CR-246):

```bash
vdel -lib <library_name> -all
```

Library mapping from the command line
You can issue a command to set the mapping between a logical library name and a directory; its form is:

```bash
vmap <logical_name> <directory_pathname>
```

This command may be invoked from either a UNIX/DOS prompt or from the command line within ModelSim.

When you use vmap (CR-283) this way you are modifying the modelsim.ini file. You can also modify modelsim.ini manually by adding a mapping line. To do this, edit the modelsim.ini file using any text editor and add a line under the [Library] section heading using the syntax:

```ini
<logical_name> = <directory_pathname>
```

More than one logical name can be mapped to a single directory. For example, suppose the modelsim.ini file in the current working directory contains following lines:

```
[Library]
work = /usr/rick/design
myasic = /usr/rick/design
```

This would allow you to use either the logical name work or myasic in a library or use clause to refer to the same design library.

Unix symbolic links
You can also create a UNIX symbolic link to the library using the host platform command:

```bash
ln -s <directory_pathname> <logical_name>
```

The vmap command (CR-283) can also be used to display the mapping of a logical library name to a directory. To do this, enter the shortened form of the command:

```
vmap <logical_name>
```

Library search rules
The system searches for the mapping of a logical name in the following order:

- First the system looks for a modelsim.ini file.
- If the system doesn’t find a modelsim.ini file, or if the specified logical name does not exist in the modelsim.ini file, the system searches the current working directory for a subdirectory that matches the logical name.

An error is generated by the compiler if you specify a logical name that does not resolve to an existing directory.
Working with design libraries

See also

See "ModelSim Commands" (CR-29) for more information about the library management commands, "Graphic Interface" (UM-143) for more information about the graphical user interface, and "Projects and system initialization" (UM-21) for more information about the modelsim.ini file.

Moving a library

Individual design units in a design library cannot be moved. An entire design library can be moved, however, by using standard operating system commands for moving a directory.
Specifying the resource libraries

VHDL resource libraries

Within a VHDL source file, you can use the VHDL library clause to specify logical names of one or more resource libraries to be referenced in the subsequent design unit. The scope of a library clause includes the text region that starts immediately after the library clause and extends to the end of the declarative region of the associated design unit. It does not extend to the next design unit in the file.

Note that the library clause is not used to specify the working library into which the design unit is placed after compilation; the vcom command (CR-240) adds compiled design units to the current working library. By default, this is the library named work. To change the current working library, you can use vcom -work and specify the name of the desired target library.

Predefined libraries

Certain resource libraries are predefined in standard VHDL. The library named std contains the packages standard and textio, which should not be modified. The contents of these packages and other aspects of the predefined language environment are documented in the IEEE Standard VHDL Language Reference Manual, Std 1076-1987 and ANSI/IEEE Std 1076-1993. See also, “Using the TextIO package” (UM-55).

A VHDL use clause can be used to select specific declarations in a library or package that are to be visible within a design unit during compilation. A use clause references the compiled version of the package—not the source.

By default, every design unit is assumed to contain the following declarations:

```vhdl
LIBRARY std, work;
USE std.standard.all
```

To specify that all declarations in a library or package can be referenced, you can add the suffix .all to the library/package name. For example, the use clause above specifies that all declarations in the package standard in the design library named std are to be visible to the VHDL design file in which the use clause is placed. Other libraries or packages are not visible unless they are explicitly specified using a library or use clause.

Another predefined library is work, the library where a design unit is stored after it is compiled as described earlier. There is no limit to the number of libraries that can be referenced, but only one library is modified during compilation.
Alternate IEEE libraries supplied

The installation directory may contain two or more versions of the IEEE library:

- **ieepure**
  Contains only IEEE approved std_logic_1164 packages (accelerated for ModelSim).

- **ieee**
  Contains precompiled Synopsys and IEEE arithmetic packages which have been accelerated by Model Technology including math_complex, math_real, numeric_bit, numeric_std, std_logic_1164, std_logic_misc, std_logic_textio, std_logic_arith, std_logic_signed, std_logic_unsigned, vital_primitives, vital_timing, and vital_memory.

You can select which library to use by changing the mapping in the modelsim.ini file. The modelsim.ini file in the installation directory defaults to the ieee library.

VITAL 2000 library

ModelSim versions 5.5 and later include a separate VITAL 2000 library that contains an accelerated vital_memory package.

You’ll need to add a **use** clause to your VHDL code to access the package. For example:

```vhdl
LIBRARY vital2000;
USE vital2000.vital_memory.all
```

Also, when you compile, use the **-vital2000** switch to vcom (CR-240).

Rebuilding supplied libraries

Resource libraries are supplied precompiled in the modeltech installation directory. If you need to rebuild these libraries, the sources are provided in the vhdl_src directory; a macro file is also provided for Windows platforms (rebldlibs.do). To rebuild the libraries, invoke the DO file from within ModelSim with this command:

```
do rebldlibs.do
```

(Make sure your current directory is the modeltech install directory before you run this file.) Shell scripts are provided for UNIX (rebuild_libs.csh and rebuild_libs.sh). To rebuild the libraries, execute one of the rebuild_libs scripts while in the modeltech directory.

**Note:** Because accelerated subprograms require attributes that are available only under the 1993 standard, many of the libraries are built using vcom (CR-240) with the **-93** option.

Regenerating your design libraries

Depending on your current ModelSim version, you may need to regenerate your design libraries before running a simulation. Check the installation README file to see if your libraries require an update. You can regenerate your design libraries using the **Refresh** command from the Library tab context menu (see "Managing library contents" (UM-40)), or by using the **-refresh** argument to vcom (CR-240) and vlog (CR-274).

From the command line, you would use vcom with the **-refresh** option to update VHDL design units in a library, and vlog with the **-refresh** option to update Verilog design units.
By default, the work library is updated; use `-work <library>` to update a different library. For example, if you have a library named `mylib` that contains both VHDL and Verilog design units:

```
vcom -work mylib -refresh
vlog -work mylib -refresh
```

An important feature of `-refresh` is that it rebuilds the library image without using source code. This means that models delivered as compiled libraries without source code can be rebuilt for a specific release of ModelSim (4.6 and later only). In general, this works for moving forwards or backwards on a release. Moving backwards on a release may not work if the models used compiler switches or directives (Verilog only) that do not exist in the older release.

**Note:** You don't need to regenerate the std, ieee, vital22b, and verilog libraries. Also, you cannot use the `-refresh` option to update libraries that were built before the 4.6 release.

### Verilog resource libraries

ModelSim supports and encourages separate compilation of distinct portions of a Verilog design. The `vlog` (CR-274) compiler is used to compile one or more source files into a specified library. The library thus contains pre-compiled modules and UDPs (and, perhaps, VHDL design units) that are referenced by the simulator as it loads the design. See "Library usage" (UM-72).

### Maintaining 32-bit and 64-bit versions in the same library

It is possible with ModelSim to maintain 32-bit and 64-bit versions of a design in the same library. To do this, you must compile the design with one of the versions (32-bit or 64-bit), and "refresh" the design with the other version. For example:

Using the 32-bit version of ModelSim:

```
vcom file1.vhd
vcom file2.vhd
```

Next, using the 64-bit version of ModelSim:

```
vcom -refresh
```

Do not compile the design with one version, and then recompile it with the other. If you do this, ModelSim will remove the first module, because it could be "stale."
Importing FPGA libraries

ModelSim includes an import wizard for referencing and using vendor FPGA libraries. The wizard scans for and enforces dependencies in the libraries and determines the correct mappings and target directories.

⚠️ **Important:** The FPGA libraries you import must be pre-compiled. Most FPGA vendors supply pre-compiled libraries configured for use with ModelSim.

To import an FPGA library, select **Design > Import Library** (Main window).

Follow the instructions in the wizard to complete the import.
This chapter provides an overview of compilation and simulation for VHDL designs within the ModelSim environment, using the TextIO package with ModelSim; ModelSim’s implementation of the VITAL (VHDL Initiative Towards ASIC Libraries) specification for ASIC modeling; and documentation on ModelSim’s special built-in utilities package.

The TextIO package is defined within the VHDL Language Reference Manuals, IEEE Std 1076-1987 and IEEE Std 1076-1993; it allows human-readable text input from a declared source within a VHDL file during simulation.
Compiling and simulating with the GUI

Many of the examples in this chapter are shown from the command line. For compiling and simulating within a project or the ModelSim GUI, see:

- Getting started with projects (UM-24)
- Compiling with the graphic interface (UM-244)
- Simulating with the graphic interface (UM-250)

ModelSim variables

Several variables are available to control simulation, provide simulator state feedback, or modify the appearance of the ModelSim GUI. To take effect, some variables, such as environment variables, must be set prior to simulation. See Appendix A - ModelSim Variables for a complete listing of ModelSim variables.
Compiling VHDL designs

Creating a design library

Before you can compile your design, you must create a library in which to store the compilation results. Use `vlib` (CR-273) to create a new library. For example:

```
vlib work
```

This creates a library named `work`. By default, compilation results are stored in the `work` library.

▶ **Note:** The `work` library is actually a subdirectory named `work`. This subdirectory contains a special file named `_info`. Do not create libraries using UNIX, MS Windows, or DOS commands – always use the `vlib` command (CR-273).

See "Design libraries" (UM-37) for additional information on working with libraries.

Invoking the VHDL compiler

ModelSim compiles one or more VHDL design units with a single invocation of `vcom` (CR-240), the VHDL compiler. The design units are compiled in the order that they appear on the command line. For VHDL, the order of compilation is important – you must compile any entities or configurations before an architecture that references them.

You can simulate a design containing units written with both the 1076 -1987 and 1076 -1993 versions of VHDL. To do so you will need to compile units from each VHDL version separately. The `vcom` (CR-240) command compiles units written with version 1076 -1987 by default; use the `-93` option with `vcom` (CR-240) to compile units written with version 1076 -1993. You can also change the default by modifying the `modelsim.ini` file (see "Preference variables located in INI files" (UM-386) for more information).

Dependency checking

Dependent design units must be reanalyzed when the design units they depend on are changed in the library. `vcom` (CR-240) determines whether or not the compilation results have changed. For example, if you keep an entity and its architectures in the same source file and you modify only an architecture and recompile the source file, the entity compilation results will remain unchanged and you will not have to recompile design units that depend on the entity.
Simulating VHDL designs

After compiling the design units, you can simulate your designs with vsim (CR-284). This section discusses simulation from the UNIX or Windows/DOS command line. You can also use a project to simulate (see "Getting started with projects" (UM-24)) or the Load Design dialog box (see "Simulating with the graphic interface" (UM-250)).

Note: Simulation normally stops if a failure occurs; however, if a bounds check on a signal fails the simulator will continue running.

Invoking the simulator from the Main window

For VHDL, invoke vsim (CR-284) with the name of the configuration, or entity/architecture pair. Note that if you specify a configuration you may not specify an architecture.

This example invokes vsim (CR-284) on the entity my_asic and the architecture structure:

```
vsim my_asic structure
```

If a design unit name is not specified, vsim (CR-284) will present the Load Design dialog box from which you can choose a configuration or entity/architecture pair. See "Simulating with the graphic interface" (UM-250) for more information.

Selecting the time resolution

The simulation time resolution is 1 ns by default. You can select a specific time resolution with the vsim (CR-284) -t option or from the Load Design dialog box. Available resolutions are: 1x, 10x or 100x of fs, ps, ns, us, ms, or sec.

For example, to run in femtosecond resolution, or 10fs resolution respectively:

```
vsim -t fs topmod
vsim -t 10fs topmod
```

Note that there is no space between the value and the units (i.e., 10fs, not 10 fs).

The default time resolution can also be changed by modifying the Resolution (UM-390) variable in the modelsim.ini file. (See "Preference variables located in INI files" (UM-386) for more information on modifying the modelsim.ini file.) You can view the current resolution by invoking the report command (CR-191) with the simulator state option.

Invoking Code Coverage with Vsim

ModelSim’s Code Coverage feature gives you graphical and report file feedback on how the source code is being executed. It allows line number execution statistics to be kept by the simulator. It can be used during any design phase and in all levels and types of designs. For complete details, see Chapter 10 - Code Coverage.

To acquire code coverage statistics, the -coverage switch must be specified during the command-line invocation of the simulator.

```
vsim -coverage ...
```

This will allow you to use the various code coverage commands: coverage clear (CR-112), coverage reload (CR-113), and coverage report (CR-114).
Using the TextIO package

To access the routines in TextIO, include the following statement in your VHDL source code:

```
USE std.textio.all;
```

A simple example using the package TextIO is:

```
USE std.textio.all;
ENTITY simple_textio IS
END;
ARCHITECTURE simple_behavior OF simple_textio IS
BEGIN
  PROCESS
    VARIABLE i: INTEGER:= 42;
    VARIABLE LLL: LINE;
  BEGIN
    WRITE (LLL, i);
    WRITELINE (OUTPUT, LLL);
    WAIT;
  END PROCESS;
END simple_behavior;
```

Syntax for file declaration

The VHDL’87 syntax for a file declaration is:

```
file identifier : subtype_indication is [ mode ] file_logical_name ;
```

where "file_logical_name" must be a string expression.

The VHDL’93 syntax for a file declaration is:

```
file identifier_list : subtype_indication [ file_open_information ] ;
```

You can specify a full or relative path as the file_logical_name; for example (VHDL’87):

```
file filename : TEXT is in "usr/rick/myfile";
```

Normally if a file is declared within an architecture, process, or package, the file is opened when you start the simulator and is closed when you exit from it. If a file is declared in a subprogram, the file is opened when the subprogram is called and closed when execution RETURNS from the subprogram. Alternatively, the opening of files can be delayed until the first read or write by setting the DelayFileOpen variable in the modelsim.ini file. Also, the number of concurrently open files can be controlled by the ConcurrentFileLimit variable. These variables help you manage a large number of files during simulation. See Appendix A - ModelSim Variables for more details.
Using STD_INPUT and STD_OUTPUT within ModelSim

The standard VHDL’87 TextIO package contains the following file declarations:

```
file input: TEXT is in "STD_INPUT";
file output: TEXT is out "STD_OUTPUT";
```

The standard VHDL’93 TextIO package contains these file declarations:

```
file input: TEXT open read_mode is "STD_INPUT";
file output: TEXT open write_mode is "STD_OUTPUT";
```

STD_INPUT is a file_logical_name that refers to characters that are entered interactively from the keyboard, and STD_OUTPUT refers to text that is displayed on the screen.

In ModelSim, reading from the STD_INPUT file allows you to enter text into the current buffer from a prompt in the Main window. The last line written to the STD_OUTPUT file appears at the prompt.
TextIO implementation issues

Writing strings and aggregates

A common error in VHDL source code occurs when a call to a WRITE procedure does not specify whether the argument is of type STRING or BIT_VECTOR. For example, the VHDL procedure:

```
WRITE (L, "hello");
```

will cause the following error:

```
ERROR: Subprogram "WRITE" is ambiguous.
```

In the TextIO package, the WRITE procedure is overloaded for the types STRING and BIT_VECTOR. These lines are reproduced here:

```
procedure WRITE(L: inout LINE; VALUE: in BIT_VECTOR;
   JUSTIFIED: in SIDE:= RIGHT; FIELD: in WIDTH := 0);

procedure WRITE(L: inout LINE; VALUE: in STRING;
   JUSTIFIED: in SIDE:= RIGHT; FIELD: in WIDTH := 0);
```

The error occurs because the argument "hello" could be interpreted as a string or a bit vector, but the compiler is not allowed to determine the argument type until it knows which function is being called.

The following procedure call also generates an error:

```
WRITE (L, "010101");
```

This call is even more ambiguous, because the compiler could not determine, even if allowed to, whether the argument "010101" should be interpreted as a string or a bit vector.

There are two possible solutions to this problem:

- Use a qualified expression to specify the type, as in:

  ```
  WRITE (L, string'("hello"));
  ```

- Call a procedure that is not overloaded, as in:

  ```
  WRITE_STRING (L, "hello");
  ```

The WRITE_STRING procedure simply defines the value to be a STRING and calls the WRITE procedure, but it serves as a shell around the WRITE procedure that solves the overloading problem. For further details, refer to the WRITE_STRING procedure in the io_utils package, which is located in the file `/modeltech/examples/io_utils.vhd`. 
Reading and writing hexadecimal numbers

The reading and writing of hexadecimal numbers is not specified in standard VHDL. The Issues Screening and Analysis Committee of the VHDL Analysis and Standardization Group (ISAC-VASG) has specified that the TextIO package reads and writes only decimal numbers.

To expand this functionality, ModelSim supplies hexadecimal routines in the package io_utils, which is located in the file /modeltech/examples/io_utils.vhd. To use these routines, compile the io_utils package and then include the following use clauses in your VHDL source code:

```vhdl
use std.textio.all;
use work.io_utils.all;
```

Dangling pointers

Dangling pointers are easily created when using the TextIO package, because WRITELINE de-allocates the access type (pointer) that is passed to it. Following are examples of good and bad VHDL coding styles:

**Bad VHDL** (because L1 and L2 both point to the same buffer):

```vhdl
READLINE (infile, L1); -- Read and allocate buffer
L2 := L1; -- Copy pointers
WRITELINE (outfile, L1); -- Deallocate buffer
```

**Good VHDL** (because L1 and L2 point to different buffers):

```vhdl
READLINE (infile, L1); -- Read and allocate buffer
L2 := new string'(L1.all); -- Copy contents
WRITELINE (outfile, L1); -- Deallocate buffer
```

The ENDLINE function

The ENDLINE function described in the *IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1987* contains invalid VHDL syntax and cannot be implemented in VHDL. This is because access types must be passed as variables, but functions only allow constant parameters.

Based on an ISAC-VASG recommendation the ENDLINE function has been removed from the TextIO package. The following test may be substituted for this function:

```vhdl
(L = NULL) OR (L'LENGTH = 0)
```

The ENDFILE function

In the *VHDL Language Reference Manuals, IEEE Std 1076-1987 and IEEE Std 1076-1993*, the ENDFILE function is listed as:

```vhdl
-- function ENDFILE (L: in TEXT) return BOOLEAN;
```

As you can see, this function is commented out of the standard TextIO package. This is because the ENDFILE function is implicitly declared, so it can be used with files of any type, not just files of type TEXT.
Using alternative input/output files

You can use the TextIO package to read and write to your own files. To do this, just declare an input or output file of type TEXT.

The VHDL’87 declaration is:

```
file myinput : TEXT is in "pathname.dat";
```

The VHDL’93 declaration is:

```
file myinput : TEXT open read_mode is "pathname.dat";
```

Then include the identifier for this file ("myinput" in this example) in the READLINE or WRITELINE procedure call.

Providing stimulus

You can stimulate and test a design by reading vectors from a file, using them to drive values onto signals, and testing the results. A VHDL test bench has been included with the ModelSim install files as an example. Check for this file:

```
<install_dir>/modeltech/examples/stimulus.vhd
```
Obtaining the VITAL specification and source code

VITAL ASIC Modeling Specification
The IEEE 1076.4 VITAL ASIC Modeling Specification is available from the Institute of Electrical and Electronics Engineers, Inc.:
IEEE Customer Service
445 Hoes Lane
Piscataway, NJ 08855-1331
Tel: (800)678-4333 ((908)562-5420 from outside the U.S.)
Fax: (908)981-9667
home page: http://www.ieee.org

VITAL source code
The source code for VITAL packages is provided in the /<install_dir>/modeltech/vhdl_src/vital2.2b, /vital95, or /vital2000 directories.

VITAL packages
VITAL v3.0 accelerated packages are pre-compiled into the ieee library in the installation directory.

Note: By default, ModelSim is optimized for VITAL v3.0. You can, however, revert to VITAL v2.2b by invoking vsim (CR-284) with the -vital2.2b option, and by mapping library vital to <install_dir>/modeltech/vital2.2b.
ModelSim VITAL compliance

A simulator is VITAL compliant if it implements the SDF mapping and if it correctly simulates designs using the VITAL packages, as outlined in the VITAL Model Development Specification. ModelSim is compliant with the IEEE 1076.4 VITAL ASIC Modeling Specification. In addition, ModelSim accelerates the VITAL_Timing and VITAL_Primitives packages. The procedures in these packages are optimized and built into the simulator kernel. By default, vsim (CR-284) uses the optimized procedures. The optimized procedures are functionally equivalent to the IEEE 1076.4 VITAL ASIC Modeling Specification (VITAL v3.0).

VITAL compliance checking

Compliance checking is important in enabling VITAL acceleration; to qualify for global acceleration, an architecture must be VITAL-level-one compliant. vcom (CR-240) automatically checks for VITAL 3.0 compliance on all entities with the VITAL_Level0 attribute set, and all architectures with the VITAL_Level0 or VITAL_Level1 attribute set. It also checks for VITAL 2000 compliance on all architectures using the vital2000 library.

If you are using VITAL 2.2b, you must turn off the compliance checking either by not setting the attributes, or by invoking vcom (CR-240) with the option -novitalcheck. It is, of course, possible to turn off compliance checking for VITAL 3.0 as well; we strongly suggest that you leave checking on to ensure optimal simulation.

VITAL compliance warnings

The following LRM errors are printed as warnings (if they were considered errors they would prevent VITAL level 1 acceleration); they do not affect how the architecture behaves.

- Starting index constraint to DataIn and PreviousDataIn parameters to VITALStateTable do not match (1076.4 section 6.4.3.2.2)
- Size of PreviousDataIn parameter is larger than the size of the DataIn parameter to VITALStateTable (1076.4 section 6.4.3.2.2)
- Signal q_w is read by the VITAL process but is NOT in the sensitivity list (1076.4 section 6.4.3)

The first two warnings are minor cases where the body of the VITAL 3.0 LRM is slightly stricter than the package portion of the LRM. Since either interpretation will provide the same simulation results, we chose to make these two cases just warnings.

The last warning is a relaxation of the restriction on reading an internal signal that is not in the sensitivity list. This is relaxed only for the CheckEnabled parameters of the timing checks, and only if it is not read elsewhere.

You can control the visibility of VITAL compliance-check warnings in your vcom (CR-240) transcript. They can be suppressed by using the vcom -nowarn switch as in vcom -nowarn 6. The 6 comes from the warning level printed as part of the warning, i.e., WARNING[6]. You can also add the following line to your modelsim.ini file in the [vcom] VHDL compiler control variables (UM-386) section.

[vcom]
Show_VitalChecksWarnings = 0
Compiling and Simulating with accelerated VITAL packages

`vcom` (CR-240) automatically recognizes that a VITAL function is being referenced from the `ieee` library and generates code to call the optimized built-in routines.

Optimization occurs on two levels:

- **VITAL Level-0 optimization**
  This is a function-by-function optimization. It applies to all level-0 architectures, and any level-1 architectures that failed level-1 optimization.

- **VITAL Level-1 optimization**
  Performs global optimization on a VITAL 3.0 level-1 architecture that passes the VITAL compliance checker. This is the default behavior.

**Compiler options for VITAL optimization**
Several `vcom` (CR-240) options control and provide feedback on VITAL optimization:

- `-O0` | `-O4`
  Lower the optimization to a minimum with `-O0` (capital oh zero). Optional. Use this to work around bugs, increase your debugging visibility on a specific cell, or when you want to place breakpoints on source lines that have been optimized out.

  Enable optimizations with `-O4` (default).

- `-debugVA`
  Prints a confirmation if a VITAL cell was optimized, or an explanation of why it was not, during VITAL level-1 acceleration.

- `-vital2000`
  Turns on acceleration for the VITAL 2000 `vital_memory` package.

ModelSim VITAL built-ins will be updated in step with new releases of the VITAL packages.
Util package

The util package is included in ModelSim versions 5.5 and later and serves as a container for various VHDL utilities. The package is part of the modelsim_lib library which is located in the modelsim tree and mapped in the default modelsim.ini file.

To access the utilities in the package, you would add lines like the following to your VHDL code:

```vhdl
library modelsim_lib;
use modelsim_lib.util.all;
```

**get_resolution()**

get_resolution() returns the current simulator resolution as a real number. For example, 1 femtosecond corresponds to 1e-15.

**Syntax**

```vhdl
resval := get_resolution();
```

**Returns**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resval</td>
<td>real</td>
<td>The simulator resolution represented as a real</td>
</tr>
</tbody>
</table>

**Arguments**

None

**Related functions**

to_real() (UM-65)
to_time() (UM-66)

**Example**

If the simulator resolution is set to 10ps, and you invoke the command:

```vhdl
resval := get_resolution();
```

the value returned to resval would be 1e-11.
init_signal_spy()

The init_signal_spy() utility mirrors the value of a VHDL signal or Verilog register/wire (called the spy_object) onto an existing VHDL signal or Verilog register (called the dest_object). This allows you to reference signals, registers, or wires at any level of hierarchy from within a VHDL architecture (e.g., a testbench).

This system task works only in ModelSim versions 5.5 and newer.

Syntax

init_signal_spy( spy_object, dest_object, verbose);

Returns

Nothing

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spy_object</td>
<td>string</td>
<td>Required. A full hierarchical path (or relative path with reference to the calling block) to a VHDL signal or Verilog register/wire. Use the path separator to which your simulation is set (i.e., &quot;/&quot; or &quot;.&quot;). A full hierarchical path must begin with a &quot;/&quot; or &quot;.&quot;. The path must be contained within double quotes.</td>
</tr>
<tr>
<td>dest_object</td>
<td>string</td>
<td>Required. A full hierarchical path (or relative path with reference to the calling block) to an existing VHDL signal or Verilog register. Use the path separator to which your simulation is set (i.e., &quot;/&quot; or &quot;.&quot;). A full hierarchical path must begin with a &quot;/&quot; or &quot;.&quot;. The path must be contained within double quotes.</td>
</tr>
<tr>
<td>verbose</td>
<td>integer</td>
<td>Optional. Possible values are 0 or 1. Specifies whether you want a message reported in the Transcript stating that the spy_object’s value is mirrored onto the dest_object. Default is 0, no message.</td>
</tr>
</tbody>
</table>

Related functions

None

Limitations

- When mirroring the value of a Verilog register/wire onto a VHDL signal, the VHDL signal must be of type bit, bit_vector, std_logic, or std_logic_vector.
- Mirroring slices or single bits of a vector is not supported. If you do reference a slice or bit of a vector, the function will assume that you are referencing the entire vector.
**Example**

```vhdl
library modelsim_lib;
use modelsim_lib.util.all;
entity top is
end;

architecture ...
  signal top_sig1 : std_logic;
begin
  ...
  spy_process : process
  begin
    init_signal_spy("/top/uut/inst1/sig1","/top_sig1",1);
    wait;
  end process spy_process;
  ...
end;
```

In this example, the value of "*/top/uut/inst1/sig1" will be mirrored onto "*/top_sig1".

**to_real()**

to_real() converts the physical type time value into a real value with respect to the current simulator resolution. The precision of the converted value is determined by the simulator resolution. For example, if you were converting 1900 fs to a real and the simulator resolution was ps, then the real value would be 2.0 (i.e. 2 ps).

**Syntax**

```vhdl
realval := to_real(timeval);
```

**Returns**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>realval</td>
<td>real</td>
<td>The time value represented as a real with respect to the simulator resolution</td>
</tr>
</tbody>
</table>

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeval</td>
<td>time</td>
<td>The value of the physical type time</td>
</tr>
</tbody>
</table>

**Related functions**

- `get_resolution()` (UM-63)
- `to_time()` (UM-66)
Example
If the simulator resolution is set to ps, and you enter the following function:

```vhdl
realval := to_real(12.99 ns);
```
then the value returned to realval would be 12990.0. If you wanted the returned value to be in units of nanoseconds (ns) instead, you would use the `get_resolution()` (UM-63) function to recalculate the value:

```vhdl
realval := 1e+9 * (to_real(12.99 ns)) * get_resolution();
```

If you wanted the returned value to be in units of femtoseconds (fs), you would enter the function this way:

```vhdl
realval := 1e+15 * (to_real(12.99 ns)) * get_resolution();
```

`to_time()`

to_time converts a real value into a time value with respect to the current simulator resolution. The precision of the converted value is determined by the simulator resolution. For example, if you were converting 5.9 to a time and the simulator resolution was ps, then the time value would be 6 ps.

Syntax

```vhdl
timeval := to_time(realval);
```

Returns

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeval</td>
<td>time</td>
<td>The real value represented as a physical type time with respect to the simulator resolution</td>
</tr>
</tbody>
</table>

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>realval</td>
<td>real</td>
<td>The value of the type real</td>
</tr>
</tbody>
</table>

Related functions

- `get_resolution()` (UM-63)
- `to_real()` (UM-65)

Example
If the simulator resolution is set to ps, and you enter the following function:

```vhdl
timeval := to_time(72.49);
```
then the value returned to timeval would be 72 ps.
# 5 - Verilog Simulation

## Chapter contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compilation</td>
<td>UM-69</td>
</tr>
<tr>
<td>Incremental compilation</td>
<td>UM-70</td>
</tr>
<tr>
<td>Library usage</td>
<td>UM-72</td>
</tr>
<tr>
<td>Verilog-XL compatible compiler options</td>
<td>UM-73</td>
</tr>
<tr>
<td>Verilog-XL <code>uselib</code> compiler directive</td>
<td>UM-75</td>
</tr>
<tr>
<td>Simulation</td>
<td>UM-77</td>
</tr>
<tr>
<td>Invoking the simulator</td>
<td>UM-77</td>
</tr>
<tr>
<td>Simulation resolution limit</td>
<td>UM-77</td>
</tr>
<tr>
<td>Event order issues</td>
<td>UM-78</td>
</tr>
<tr>
<td>Verilog-XL compatible simulator options</td>
<td>UM-79</td>
</tr>
<tr>
<td>Compiling for faster performance</td>
<td>UM-83</td>
</tr>
<tr>
<td>Compiling with <code>-fast</code></td>
<td>UM-83</td>
</tr>
<tr>
<td>Compiling mixed designs with <code>-fast</code></td>
<td>UM-84</td>
</tr>
<tr>
<td>Compiling gate-level designs with <code>-fast</code></td>
<td>UM-84</td>
</tr>
<tr>
<td>Referencing the optimized design</td>
<td>UM-85</td>
</tr>
<tr>
<td>Enabling design object visibility with the <code>+acc</code> option</td>
<td>UM-88</td>
</tr>
<tr>
<td>Using pre-compiled libraries</td>
<td>UM-89</td>
</tr>
<tr>
<td>Cell Libraries</td>
<td>UM-90</td>
</tr>
<tr>
<td>SDF timing annotation</td>
<td>UM-90</td>
</tr>
<tr>
<td>Delay modes</td>
<td>UM-90</td>
</tr>
<tr>
<td>System Tasks</td>
<td>UM-92</td>
</tr>
<tr>
<td>IEEE Std 1364 system tasks</td>
<td>UM-92</td>
</tr>
<tr>
<td>Verilog-XL compatible system tasks</td>
<td>UM-95</td>
</tr>
<tr>
<td><code>Sinit_signal_spy</code></td>
<td>UM-97</td>
</tr>
<tr>
<td>Compiler Directives</td>
<td>UM-99</td>
</tr>
<tr>
<td>IEEE Std 1364 compiler directives</td>
<td>UM-99</td>
</tr>
<tr>
<td>Verilog-XL compatible compiler directives</td>
<td>UM-99</td>
</tr>
<tr>
<td>Verilog PLI/VPI</td>
<td>UM-101</td>
</tr>
<tr>
<td>Registering PLI applications</td>
<td>UM-101</td>
</tr>
<tr>
<td>Registering VPI applications</td>
<td>UM-103</td>
</tr>
<tr>
<td>Compiling and linking PLI/VPI applications</td>
<td>UM-104</td>
</tr>
<tr>
<td>The PLI callback reason argument</td>
<td>UM-111</td>
</tr>
<tr>
<td>The <code>sizetf</code> callback function</td>
<td>UM-112</td>
</tr>
<tr>
<td>PLI object handles</td>
<td>UM-112</td>
</tr>
<tr>
<td>Third party PLI applications</td>
<td>UM-113</td>
</tr>
<tr>
<td>Support for VHDL objects</td>
<td>UM-114</td>
</tr>
<tr>
<td>IEEE Std 1364 ACC routines</td>
<td>UM-115</td>
</tr>
<tr>
<td>IEEE Std 1364 TF routines</td>
<td>UM-116</td>
</tr>
<tr>
<td>Verilog-XL compatible routines</td>
<td>UM-118</td>
</tr>
<tr>
<td>64-bit support in the PLI</td>
<td>UM-118</td>
</tr>
<tr>
<td>PLI/VPI tracing</td>
<td>UM-118</td>
</tr>
</tbody>
</table>
This chapter describes how to compile and simulate Verilog designs with ModelSim Verilog. ModelSim Verilog implements the Verilog language as defined by the IEEE Std 1364, and it is recommended that you obtain this specification as a reference manual.

In addition to the functionality described in the IEEE Std 1364, ModelSim Verilog includes the following features:

- Standard Delay Format (SDF) annotator compatible with many ASIC and FPGA vendor's Verilog libraries
- Value Change Dump (VCD) file extensions for ASIC vendor test tools
- Dynamic loading of PLI/VPI applications
- Compilation into retargetable, executable code
- Incremental design compilation
- Extensive support for mixing VHDL and Verilog in the same design (including SDF annotation)
- Graphic Interface that is common with ModelSim VHDL
- Extensions to provide compatibility with Verilog-XL

The following IEEE Std 1364 functionality is partially implemented in ModelSim Verilog:

- Verilog Procedural Interface (VPI) (see /install_dir/modeltech/docs/technotes/Verilog_VPI.note for details)

Many of the examples in this chapter are shown from the command line. For compiling and simulating within a project or ModelSim's GUI see:

- Getting started with projects (UM-24)
- Compiling with the graphic interface (UM-244)
- Simulating with the graphic interface (UM-250)

**ModelSim variables**

Several variables are available to control simulation, provide simulator state feedback, or modify the appearance of the ModelSim GUI. To take effect, some variables, such as environment variables, must be set prior to simulation. See *Appendix A - ModelSim Variables* for a complete listing of ModelSim variables.
Compilation

Before you can simulate a Verilog design, you must first create a library and compile the Verilog source code into that library. This section provides detailed information on compiling Verilog designs. For information on creating a design library, see Chapter 3 - Design libraries.

The ModelSim Verilog compiler, vlog, compiles Verilog source code into retargetable, executable code, meaning that the library format is compatible across all supported platforms and that you can simulate your design on any platform without having to recompile your design specifically for that platform. As you compile your design, the resulting object code for modules and UDPS is generated into a library. By default, the compiler places results into the work library. You can specify an alternate library with the -work option. The following is a simple example of how to create a work library, compile a design, and simulate it:

Contents of top.v:

```verilog
module top;
    initial $display("Hello world");
endmodule
```

Create the work library:

```
vlib work
```

Compile the design:

```
vlog top.v
   -- Compiling module top
```

View the contents of the work library (optional):

```
vdir
 MODULE top
```

Simulate the design:

```
vsim -c top
   # Loading work.top
   VSIM 1> run -all
   # Hello world
   VSIM 2> quit
```

In this example, the simulator was run without the graphic interface by specifying the -c option. After the design was loaded, the simulator command run all was entered, meaning to simulate until there are no more simulator events. Finally, the quit command was entered to exit the simulator. By default, a log of the simulation is written to the file "transcript" in the current directory.
Incremental compilation

By default, ModelSim Verilog supports incremental compilation of designs, thus saving compilation time when you modify your design. Unlike other Verilog simulators, there is no requirement that you compile the entire design in one invocation of the compiler (although, you may wish to do so to optimize performance; see "Compiling for faster performance" (UM-83)).

You are not required to compile your design in any particular order because all module and UDP instantiations and external hierarchical references are resolved when the design is loaded by the simulator. Incremental compilation is made possible by deferring these bindings, and as a result some errors cannot be detected during compilation. Commonly, these errors include: modules that were referenced but not compiled, incorrect port connections, and incorrect hierarchical references.

The following example shows how a hierarchical design can be compiled in top-down order:

Contents of top.v:

```verilog
module top;
    or2(n1, a, b);
    and2(n2, n1, c);
endmodule
```

Contents of and2.v:

```verilog
module and2(y, a, b);
    output y;
    input a, b;
    and(y, a, b);
endmodule
```

Contents of or2.v:

```verilog
module or2(y, a, b);
    output y;
    input a, b;
    or(y, a, b);
endmodule
```

Compile the design in top down order (assumes work library already exists):

```bash
% vlog top.v
-- Compiling module top

Top level modules:
top
% vlog and2.v
-- Compiling module and2

Top level modules:
and2
% vlog or2.v
-- Compiling module or2

Top level modules:
or2
```
Note that the compiler lists each module as a top level module, although, ultimately, only "top" is a top-level module. If a module is not referenced by another module compiled in the same invocation of the compiler, then it is listed as a top level module. This is just an informative message and can be ignored during incremental compilation. The message is more useful when you compile an entire design in one invocation of the compiler and need to know the top level module names for the simulator. For example,

```
% vlog top.v and2.v or2.v
-- Compiling module top
-- Compiling module and2
-- Compiling module or2

Top level modules:
top
```

The most efficient method of incremental compilation is to manually compile only the modules that have changed. This is not always convenient, especially if your source files have compiler directive interdependencies (such as macros). In this case, you may prefer to always compile your entire design in one invocation of the compiler. If you specify the `-incr` option, the compiler will automatically determine which modules have changed and generate code only for those modules. This is not as efficient as manual incremental compilation because the compiler must scan all of the source code to determine which modules must be compiled.

The following is an example of how to compile a design with automatic incremental compilation:

```
% vlog -incr top.v and2.v or2.v
-- Compiling module top
-- Compiling module and2
-- Compiling module or2

Top level modules:
top
```

Now, suppose that you modify the functionality of the "or2" module:

```
% vlog -incr top.v and2.v or2.v
-- Skipping module top
-- Skipping module and2
-- Compiling module or2

Top level modules:
top
```

The compiler informs you that it skipped the modules "top" and "and2", and compiled "or2".

Automatic incremental compilation is intelligent about when to compile a module. For example, changing a comment in your source code does not result in a recompile; however, changing the compiler command line options results in a recompile of all modules.

**Note:** Changes to your source code that do not change functionality but that do affect source code line numbers (such as adding a comment line) will cause all affected modules to be recompiled. This happens because debug information must be kept current so that ModelSim can trace back to the correct areas of the source code.
Library usage

All modules and UDPs in a Verilog design must be compiled into one or more libraries. One library is usually sufficient for a simple design, but you may want to organize your modules into various libraries for a complex design. If your design uses different modules having the same name, then you are required to put those modules in different libraries because design unit names must be unique within a library.

The following is an example of how you may organize your ASIC cells into one library and the rest of your design into another:

```
% vlib work
% vlib asiclib
% vlog -work asiclib and2.v or2.v
-- Compiling module and2
-- Compiling module or2

Top level modules:
  and2
  or2
% vlog top.v
-- Compiling module top

Top level modules:
  top
```

Note that the first compilation uses the `-work asiclib` option to instruct the compiler to place the results in the `asiclib` library rather than the default `work` library.

Since instantiation bindings are not determined at compile time, you must instruct the simulator to search your libraries when loading the design. The top level modules are loaded from the library named `work` unless you specify an alternate library with the `-lib` option. All other Verilog instantiations are resolved in the following order:

- Search libraries specified with `-Lf` options in the order they appear on the command line.
- Search the library specified in the "Verilog-XL `uselib compiler directive" (UM-75).
- Search libraries specified with `-L` options in the order they appear on the command line.
- Search the `work` library.
- Search the library explicitly named in the special escaped identifier instance name.

It is important to recognize that the work library is not necessarily a library named `work` - the `work` library refers to the library containing the module that instantiates the module or UDP that is currently being searched for. This definition is useful if you have hierarchical modules organized into separate libraries and if sub-module names overlap among the libraries. In this situation you want the modules to search for their sub-modules in the `work` library first. This is accomplished by specifying `-L work` first in the list of search libraries.

For example, assume you have a top level module "top" that instantiates module "modA" from library "libA" and module "modB" from library "libB". Furthermore, "modA" and "modB" both instantiate modules named "cellA", but the definition of "cellA" compiled into "libA" is different from that compiled into "libB". In this case, it is insufficient to just specify "-L libA -L libB" as the search libraries because instantiations of "cellA" from "modB" resolve to the "libA" version of "cellA". The appropriate search library options are "-L work -L libA -L libB".
Verilog-XL compatible compiler options

See vlog (CR-274) for a complete list of compiler options. The options described here are equivalent to Verilog-XL options. Many of these are provided to ease the porting of a design to ModelSim Verilog.

+define+<macro_name>[=<macro_text>]

This option allows you to define a macro from the command line that is equivalent to the following compiler directive:

'define <macro_name> <macro_text>

Multiple +define options are allowed on the command line. A command line macro overrides a macro of the same name defined with the `define compiler directive.

+incdir+<directory>

This option specifies which directories to search for files included with `include compiler directives. By default, the current directory is searched first and then the directories specified by the +incdir options in the order they appear on the command line. You may specify multiple +incdir options as well as multiple directories separated by "+" in a single +incdir option.

+delay_mode_distributed

This option disables path delays in favor of distributed delays. See Delay modes (UM-90) for details.

+delay_mode_path

This option sets distributed delays to zero in favor of path delays. See Delay modes (UM-90) for details.

+delay_mode_unit

This option sets path delays to zero and non-zero distributed delays to one time unit. See Delay modes (UM-90) for details.

+delay_mode_zero

This option sets path delays and distributed delays to zero. See Delay modes (UM-90) for details.

-f <filename>

This option reads more command line arguments from the specified text file. Nesting of -f options is allowed.

+mindelays

This option selects minimum delays from the "min:typ:max" expressions. If preferred, you can defer delay selection until simulation time by specifying the same option to the simulator.

+typdelays

This option selects typical delays from the "min:typ:max" expressions. If preferred, you can defer delay selection until simulation time by specifying the same option to the simulator.

+maxdelays

This option selects maximum delays from the "min:typ:max" expressions. If preferred, you can defer delay selection until simulation time by specifying the same option to the simulator.
+nowarn<mnemonic>
   This option disables the class of warning messages specified by <mnemonic>. This
   option only disables warning messages accompanied by a mnemonic enclosed in square
   brackets. For example,


   This warning message can be disabled with the +nowarnTFMPC option.

-<u>
   This option treats all identifiers in the source code as all uppercase.

### Options supporting source libraries

The following options support source libraries in the same manner as Verilog-XL. Note that
these libraries are source libraries and are very different from the libraries that the
ModelSim compiler uses to store compilation results. You may find it convenient to use
these options if you are porting a design to ModelSim or if you are familiar with these
options and prefer to use them.

Source libraries are searched after the source files on the command line are compiled. If
there are any unresolved references to modules or UDPs, then the compiler searches the
source libraries to satisfy them. The modules compiled from source libraries may in turn
have additional unresolved references that cause the source libraries to be searched again.
This process is repeated until all references are resolved or until no new unresolved
references are found. Source libraries are searched in the order they appear on the command
line.

-v <filename>
   This option specifies a source library file containing module and UDP definitions.
   Modules and UDPs within the file are compiled only if they match previously unresolved
   references. Multiple -v options are allowed.

-y <directory>
   This option specifies a source library directory containing module and UDP definitions.
   Files within this directory are compiled only if the file names match the names of
   previously unresolved references. Multiple -y options are allowed.

+libext+<suffix>
   This option works in conjunction with the -y option. It specifies file extensions for the
   files in a source library directory. By default the compiler searches for files without
   extensions. If you specify the +libext option, then the compiler will search for a file with
   the suffix appended to an unresolved name. You may specify only one +libext option,
   but it may contain multiple suffixes separated by "+". The extensions are tried in the
   order they appear in the +libext option.

+librescan
   This option changes how unresolved references are handled that are added while
   compiling a module or UDP from a source library. By default, the compiler attempts to
   resolve these references as it continues searching the source libraries. If you specify the
   +librescan option, then the new unresolved references are deferred until after the current
   pass through the source libraries. They are then resolved by searching the source libraries
   from the beginning in the order they are specified on the command line.
+nolibcell

By default, all modules compiled from a source library are treated as though they contain a `celldefine` compiler directive. This option disables this default. The `celldefine` directive only affects the PLI Access routines `acc_next_cell` and `acc_next_cell_load`.

-R <simargs>

This option instructs the compiler to invoke the simulator after compiling the design. The compiler automatically determines which top level modules are to be simulated. The command line arguments following `-R` are passed to the simulator, not the compiler. Place the `-R` option at the end of the command line or terminate the simulator command line arguments with a single `"-"` character to differentiate them from compiler command line arguments.

The `-R` option is not a Verilog-XL option, but it is used by ModelSim Verilog to combine the compile and simulate phases together as you may be used to doing with Verilog-XL. It is not recommended that you regularly use this option because you will incur the unnecessary overhead of compiling your design for each simulation run. Mainly, it is provided to ease the transition to ModelSim Verilog.

Verilog-XL `uselib` compiler directive

The `uselib` compiler directive is an alternative source library management scheme to the `-v`, `-y`, and `+libext` compiler options. It has the advantage that a design may reference different modules having the same name. You compile designs that contain `uselib` directive statements using the `-compile_uselibs vlog` switch (described below).

The syntax for the `uselib` directive is:

```
uselib <library_reference>...
```

where `<library_reference>` is:

```
dir=<library_directory> | file=<library_file> | libext=<file_extension> | lib=<library_name>
```

In Verilog-XL, the library references are equivalent to command line options as follows:

```
dir=<library_directory> -y <library_directory>
file=<library_file> -v <library_file>
libext=<file_extension> +libext+<file_extension>
```

For example, the following directive

```
uselib dir=/h/vendorA libext=.v
```

is equivalent to the following command line options:

```
-y /h/vendorA +libext+.v
```

Since the `uselib` directives are embedded in the Verilog source code, there is more flexibility in defining the source libraries for the instantiations in the design. The appearance of a `uselib` directive in the source code explicitly defines how instantiations that follow it are resolved, completely overriding any previous `uselib` directives.

For example, the following code fragment shows how two different modules that have the same name can be instantiated within the same design:

```
uselib dir=/h/vendorA file=.v
NAND2 u1(n1, n2, n3);
```
This allows the NAND2 module to have different definitions in the vendorA and vendorB libraries.

**-compile_uselibs argument**

In ModelSim versions 5.5 and later, use the `-compile_uselibs` argument to `vlog` to reference `uselib` directives. The argument finds the source files referenced in the directive, compiles them into automatically created object libraries, and updates the modelsim.ini file with the logical mappings to the libraries.

When using `-compile_uselibs`, ModelSim determines into what directory to compile the object libraries by choosing, in order, from the following three values:

- The directory name specified by the `-compile_uselibs` argument. For example, `-compile_uselibs=./mydir`
- The directory specified by the MTI_USELIB_DIR environment variable (see "Environment variables" (UM-383))
- A directory named "mti_uselibs" that is created in the current working directory

**Note:** In ModelSim versions prior to 5.5, the library files referenced by the `uselib` directive were not automatically compiled by ModelSim Verilog. To maintain backwards compatibility, this is still the default behavior when `-compile_uselibs` is not used. See **Pre-5.5 release implementation of `uselib` directives** for a description of the pre-5.5 implementation.

**uselib is persistent**

As mentioned above, the appearance of a `uselib` directive in the source code explicitly defines how instantiations that follow it are resolved. This may result in unexpected consequences. For example, consider the following compile command:

```
vlog -compile_uselibs dut.v srtr.v
```

Assume that `dut.v` contains a `uselib` directive. Since `srtr.v` is compiled after `dut.v`, the `uselib` directive is still in effect. When `srtr` is loaded it is using the `uselib` directive from `dut.v` to decide where to locate modules. If this is not what you intend, then you need to put an empty `uselib` at the end of `dut.v` to "close" the previous `uselib` statement.
Simulation

The ModelSim simulator can load and simulate both Verilog and VHDL designs, providing a uniform graphic interface and simulation control commands for debugging and analyzing your designs. The graphic interface and simulator commands are described elsewhere in this manual, while this section focuses specifically on Verilog simulation.

Invoking the simulator

A Verilog design is ready for simulation after it has been compiled into one or more libraries. The simulator may then be invoked with the names of the top level modules (many designs contain only one top level module). For example, if your top level modules are "testbench" and "globals", then invoke the simulator as follows:

```
vsim testbench globals
```

Note: When working with designs that contain optimized code, this syntax may vary. Please see "Compiling for faster performance" (UM-83) for details.

If a top-level module name is not specified, ModelSim will present the Load Design dialog box from which you can choose one or more top-level modules. See “Simulating with the graphic interface” (UM-250) for more information.

After the simulator loads the top level modules, it iteratively loads the instantiated modules and UDPs in the design hierarchy, linking the design together by connecting the ports and resolving hierarchical references. By default, all modules and UDPs are loaded from the library named work.

On successful loading of the design, the simulation time is set to zero, and you must enter a run command to begin simulation. Commonly, you enter run -all to run until there are no more simulation events or until $finish is executed in the Verilog code. You can also run for specific time periods (e.g., run 100 ns). Enter the quit command to exit the simulator.

Simulation resolution limit

The simulator internally represents time as a 64-bit integer in units equivalent to the smallest unit of simulation time, also known as the simulation resolution limit. The resolution limit defaults to the smallest time precision found among all of the 'timescale compiler directives in the design. The time precision is the second number in the 'timescale directive. For example, "10 ps" in the following directive:

```
'timescale 1 ns / 10 ps
```

The time precision should not be unnecessarily small because it will limit the maximum simulation time limit, and it will degrade performance in some cases. If the design contains no 'timescale directives, then the resolution limit defaults to the "resolution" value specified in the modelsim.ini file (default is 1 ns). In any case, you can override the default resolution limit by specifying the -t option on the command line.

For example, to explicitly choose 100 fs resolution:

```
vsim -t 100fs top
```
This forces 100 fs resolution even if the design has finer time precision. As a result, time values with finer precision are rounded to the nearest 100 fs.

Event order issues

The Verilog language is defined such that the simulator is not required to execute simultaneous events in any particular order. Unfortunately, some models are inadvertently written to rely on a particular event order, and these models may behave differently when ported to another Verilog simulator. A model with event order dependencies is ambiguous and should be corrected. For example, the following code is ambiguous:

```verilog
module top;
    reg r;
    initial r = 0;
    initial r = 1;
    initial #10 $display(r);
endmodule
```

The value displayed for "r" depends on the order that the simulator executes the initial constructs that assign to "r". Conceptually, the initial constructs run concurrently and the simulator is allowed to execute them in any order. ModelSim Verilog executes the initial constructs in the order they appear in the module, and the value displayed for "r" is "1". Verilog-XL produces the same result, but a simulator that displays "0" is not incorrect because the code is ambiguous.

Since many models have been developed on Verilog-XL, ModelSim Verilog duplicates Verilog-XL event ordering as much as possible to ease the porting of those models to ModelSim Verilog. However, ModelSim Verilog does not match Verilog-XL event ordering in all cases, and if a model ported to ModelSim Verilog does not behave as expected, then you should suspect that there are event order dependencies.

Tracking down event order dependencies is a tedious task, so ModelSim Verilog aids you with a couple of compiler options:

- `-compat`  
  This option turns off optimizations that result in different event ordering than Verilog-XL. ModelSim Verilog generally duplicates Verilog-XL event ordering, but there are cases where it is inefficient to do so. Using this option does not help you find the event order dependencies, but it allows you to ignore them. Keep in mind that this option does not account for all event order discrepancies, and that using this option may degrade performance.

- `-hazards`  
  This option detects event order hazards involving simultaneous reading and writing of the same register in concurrently executing processes.

  `vsim` (CR-284) detects the following kinds of hazards:

  - **WRITE/WRITE:**
    Two processes writing to the same variable at the same time.

  - **READ/WRITE:**
    One process reading a variable at the same time it is being written to by another process. ModelSim calls this a READ/WRITE hazard if it executed the read first.
• WRITE/READ:
  Same as a READ/WRITE hazard except that ModelSim executed the write first.

  vsim (CR-284) issues an error message when it detects a hazard. The message pinpoints the
  variable and the two processes involved. You can have the simulator break on the statement
  where the hazard is detected by setting the break on assertion level to error.

To enable hazard detection you must invoke vlog (CR-274) with the -hazards option when
you compile your source code and you must also invoke vsim with the -hazards option
when you simulate.

Limitations of hazard detection:

• Reads and writes involving bit and part selects of vectors are not considered for hazard
detection. The overhead of tracking the overlap between the bit and part selects is too
high.
• A WRITE/WRITE hazard is flagged even if the same value is written by both processes.
• A WRITE/READ or READ/WRITE hazard is flagged even if the write does not modify
  the variable's value.
• Glitches on nets caused by non-guaranteed event ordering are not detected.

Verilog-XL compatible simulator options

See vsim (CR-284) for a complete list of simulator options. The options described here are
equivalent to Verilog-XL options. Many of these are provided to ease the porting of a
design to ModelSim Verilog.

+alt_path_delays
  Specify path delays operate in inertial mode by default. In inertial mode, a pending output
  transition is cancelled when a new output transition is scheduled. The result is that an
  output may have no more than one pending transition at a time, and that pulses narrower
  than the delay are filtered. The delay is selected based on the transition from the cancelled
  pending value of the net to the new pending value. The +alt_path_delays option
  modifies the inertial mode such that a delay is based on a transition from the current
  output value rather than the cancelled pending value of the net. This option has no effect
  in transport mode (see +pulse_e/<percent> and +pulse_r/<percent>).

-l <filename>
  By default, the simulation log is written to the file "transcript". The -l option allows you
  to specify an alternate file.

+maxdelays
  This option selects the maximum value in min:typ:max expressions. The default is the
  typical value. This option has no effect if the min:typ:max selection was determined at
  compile time.

+mindelays
  This option selects the minimum value in min:typ:max expressions. The default is the
  typical value. This option has no effect if the min:typ:max selection was determined at
  compile time.

+multisource_int_delays
  This option enables multisource interconnect delays with transport delay behavior and
  pulse handling. ModelSim uses a unique delay value for each driver-to-driven module
interconnect path specified in the SDF file. Pulse handling is configured using the
+\texttt{pulse\_int\_e} and +\texttt{pulse\_int\_r} switches (described below).

+\texttt{no\_cancelled\_e\_msg}
This option disables negative pulse warning messages. By default Vsim issues a warning
and then filters negative pulses on specify path delays. You can drive an X for a negative
pulse using +\texttt{show\_cancelled\_e}.

+\texttt{no\_neg\_tchk}
This option disables negative timing check limits by setting them to zero. By default
negative timing check limits are enabled. This is just the opposite of Verilog-XL, where
negative timing check limits are disabled by default, and they are enabled with the
+\texttt{neg\_tchk} option.

+\texttt{no\_notifier}
This option disables the toggling of the notifier register argument of the timing check
system tasks. By default, the notifier is toggled when there is a timing check violation,
and the notifier usually causes a UDP to propagate an X. Therefore, the +\texttt{no\_notifier}
option suppresses X propagation on timing violations.

+\texttt{no\_path\_edge}
This option causes ModelSim to ignore the input edge specified in a path delay. The result
is that all edges on the input are considered when selecting the output delay. Verilog-XL
always ignores the input edges on path delays.

+\texttt{no\_pulse\_msg}
This option disables the warning message for specify path pulse errors. A path pulse error
occurs when a pulse propagated through a path delay falls between the pulse rejection
limit and pulse error limit set with the +\texttt{pulse\_r} and +\texttt{pulse\_e} options. A path pulse error
results in a warning message, and the pulse is propagated as an X. The +\texttt{no\_pulse\_msg}
option disables the warning message, but the X is still propagated.

+\texttt{no\_show\_cancelled\_e}
This option filters negative pulses on specify path delays so they don’t show on the
output. This is the default behavior of Vsim. You can drive an X for a negative pulse
using +\texttt{show\_cancelled\_e}.

+\texttt{no\_tchk\_msg}
This option disables error messages issued by timing check system tasks when timing
check violations occur. However, notifier registers are still toggled and may result in the
propagation of X’s for timing check violations.

+\texttt{nosdfwarn}
This option disables warning messages during SDF annotation.

+\texttt{notimingchecks}
This option completely disables all timing check system tasks.

+\texttt{nowarn\<mnemonic>}
This option disables the class of warning messages specified by \texttt{\<mnemonic>}. This
option only disables warning messages accompanied by a mnemonic enclosed in square
brackets. For example,

\begin{verbatim}
\end{verbatim}

This warning message can be disabled with the +\texttt{nowarnTFMPC} option.
+ntc_warn
This option enables warning messages from the negative timing constraint algorithm. This algorithm attempts to find a set of delays for the timing check delayed net arguments such that all negative limits can be converted to non-negative limits with respect to the delayed nets. If there is no solution for this set of limits, then the algorithm sets one of the negative limits to zero and recalculates the delays. This process is repeated until a solution is found. A warning message is issued for each negative limit set to zero. By default these warnings are disabled.

+pulse_e/<percent>
This option controls how pulses are propagated through specify path delays, where <percent> is a number between 0 and 100 that specifies the error limit as a percentage of the path delay. A pulse greater than or equal to the error limit propagates to the output in transport mode (transport mode allows multiple pending transitions on an output). A pulse less than the error limit and greater than or equal to the rejection limit (see +pulse_r/<percent>) propagates to the output as an X. If the rejection limit is not specified, then it defaults to the error limit. For example, consider a path delay of 10 along with a +pulse_e/80 option. The error limit is 80% of 10 and the rejection limit defaults to 80% of 10. This results in the propagation of pulses greater than or equal to 8, while all other pulses are filtered. Note that you can force specify path delays to operate in transport mode by using the +pulse_e/0 option.

+pulse_int_e/<percent>
This option is analogous to +pulse_e, except it applies to interconnect delays only.

+pulse_int_r/<percent>
This option is analogous to +pulse_r, except it applies to interconnect delays only.

+pulse_r/<percent>
This option controls how pulses are propagated through specify path delays, where <percent> is a number between 0 and 100 that specifies the rejection limit as a percentage of the path delay. A pulse less than the rejection limit is suppressed from propagating to the output. If the error limit is not specified (see +pulse_e (UM-81)), then it defaults to the rejection limit.

+pulse_e_style_ondetect
This option selects the "on detect" style of propagating pulse errors (see +pulse_e/<percent>). A pulse error propagates to the output as an X, and the "on detect" style is to schedule the X immediately, as soon as it has been detected that a pulse error has occurred. The "on event" style is the default for propagating pulse errors (see +pulse_e_style_onevent).

+pulse_e_style_onevent
This option selects the "on event" style of propagating pulse errors (see +pulse_e/<percent>). A pulse error propagates to the output as an X, and the "on event" style is to schedule the X to occur at the same time and for the same duration that the pulse would have occurred if it had propagated through normally. The "on event" style is the default for propagating pulse errors.

+sdf_nocheck_celltype
By default, the SDF annotator checks that the CELLTYPE name in the SDF file matches the module or primitive name for the CELL instance. It is an error if the names do not match. The +sdf_nocheck_celltype option disables this error check.
+sdfVerbose
This option displays a summary of the design objects annotated for each SDF file.

+showCancelledE
This option causes Vsim to drive a pulse error state ('X') for the duration of negative pulses on specify path delays. By default Vsim filters negative pulses.

+transportIntDelays
By default, interconnect delays operate in inertial mode (pulses smaller than the delay are filtered). The +transportIntDelays option selects transport mode with pulse control for single-source nets (one interconnect path). In transport mode, narrow pulses are propagated through interconnect delays. This option works independent from +multisourceIntDelays.

+transportPathDelays
By default, path delays operate in inertial mode (pulses smaller than the delay are filtered). The +transportPathDelays option selects transport mode for path delays. In transport mode, narrow pulses are propagated through path delays. Note that this option affects path delays only, and not primitives. Primitives always operate in inertial delay mode.

+typdelays
This option selects the typical value in min:typ:max expressions. This is the default. This option has no effect if the min:typ:max selection was determined at compile time.
Compiling for faster performance

This section describes how to use the "-fast" compiler option to analyze and optimize an entire design for improved simulation performance. This option improves performance for RTL, behavioral, and gate-level designs (See below for important information specific to gate-level designs.).

ModelSim's default mode of compilation defers module instantiations, parameter propagation, and hierarchical reference resolution until the time that a design is loaded by the simulator (see "Incremental compilation" (UM-70)). This has the advantage that a design does not have to be compiled all at once, allowing independent compilation of modules without requiring knowledge of the context in which they are used.

Compiling modules independently provides flexibility to the user, but results in less efficient simulation performance in many cases. For example, the compiler must generate code for a module containing parameters as though the parameters are variables that will receive their final values when the design is loaded by the simulator. If the compiler is allowed to analyze the entire design at once, then it can determine the final values of parameters and treat them as constants in expressions, thus generating more efficient code. This is just one example of many other optimizations that require analysis of the entire design.

Compiling with -fast

The "-fast" compiler option allows the compiler to propagate parameters and perform global optimizations. A requirement of using the "-fast" option is that you must compile the source code for your entire design in a single invocation of the compiler. The following is an example invocation of the compiler and its resulting messages:

```
% vlog -fast cpu_rtl.v
-- Compiling module fp_unit
-- Compiling module mult_56
-- Compiling module testbench
-- Compiling module cpu
-- Compiling module i_unit
-- Compiling module mem_mux
-- Compiling module memory32
-- Compiling module op_unit

Top level modules:
    testbench

Analyzing design...
Optimizing 8 modules of which 6 are inlined:
    -- Inlining module i_unit(fast)
    -- Inlining module mem_mux(fast)
    -- Inlining module op_unit(fast)
```
-- Inlining module memory32(fast)
-- Inlining module mult_56(fast)
-- Inlining module fp_unit(fast)
-- Optimizing module cpu(fast)
-- Optimizing module testbench(fast)

The "Analyzing design..." message indicates that the compiler is building the design hierarchy, propagating parameters, and analyzing design object usage. This information is then used in the final step of generating module code optimized for the specific design. Note that some modules are inlined into their parent modules.

Once the design is compiled, it can be simulated in the usual way:

% vsim -c testbench
   # Loading work.testbench(fast)
   # Loading work.cpu(fast)
   VSIM 1> run -all
   VSIM 2> quit

As the simulator loads the design, it issues messages indicating that the optimized modules are being loaded. There are no messages for loading the inlined modules because their code is inlined into their parent modules.

▶ Note: If you want to optimize a very large netlist, you should optimize only the cell libraries using the -fast option. (The -forcecode option should also be specified.) The netlist itself should be compiled with the default settings. Optimizing in this manner reduces compilation time and compiler memory usage significantly.

Compiling mixed designs with -fast

To compile a Verilog module with -fast, all of its children must be Verilog as well. However, a Verilog module compiled with -fast can be instantiated from VHDL as long as the VHDL does not need to modify the parameters of the module.

Compiling gate-level designs with -fast

Gate-level designs often have large netlists that are slow to compile with -fast. In most cases we recommend the following flow for optimizing gate-level designs:

- Compile the cell library using -fast and the -forcecode argument. The -forcecode argument ensures that code is generated for in-lined modules.
- Compile the device under test and testbench without -fast.
- Create separate work directories for the cell library and the rest of the design.

One case where you wouldn’t follow this flow is when the testbench has hierarchical references into the cell library. Optimizing the library alone would result in unresolved references. In such a case, you’ll have to compile the library, design, and testbench with -fast in one invocation of the compiler. The hierarchical reference cells are then not optimized.
Note too that as of ModelSim version 5.5b, several new switches to vlog can be used to further increase optimizations on gate-level designs. The +nocheck arguments are described in the Command Reference under the vlog command (CR-274).

You can use the write report command (CR-309) command and the -debugCellOpt argument to vlog command (CR-274) to obtain information about which cells have and have not been optimized. write report produces a text file that lists all modules. Modules with "(cell)" following their names are optimized cells. For example,

```
Module: top
   Architecture: fast

Module: bottom (cell)
   Architecture: fast
```

In this case, both top and bottom were compiled with -fast, but top was not optimized and bottom was.

The -debugCellOpt argument is used with -fast when compiling the cell library. Using this argument results in Transcript window output that identifies why certain cells were not optimized.

### Referencing the optimized design

The compiler automatically assigns a secondary name to distinguish the design-specific optimized code from the unoptimized code that may coexist in the same library. The default secondary name for optimized code is "fast", and the default secondary name for unoptimized code is "verilog". You may specify an alternate name (other than "fast") for optimized code using the -fast=<option>. For example, to assign the secondary name "opt1" to your optimized code, you would enter the following:

```
% vlog -fast=opt1 cpu_rtl.v
```

If you have multiple designs that use common modules compiled into the same library, then you need to assign a different secondary name for each design so that the optimized code for a module used in one design context is not overwritten with the optimized code for the same module used in another context. This is true even if the designs are small variations of each other, such as different testbenches. For example, suppose you have two testbenches that instantiate and test the same design. You might assign different secondary names as follows:

```
% vlog -fast=t1 testbench1.v design.v
```

```
-- Compiling module testbench1
-- Compiling module design
```

```
Top level modules:
   testbench1
```

```
Analyzing design...
Optimizing 2 modules of which 0 are inlined:
   -- Optimizing module design(t1)
   -- Optimizing module testbench1(t1)
```
% vlog -fast=t2 testbed2.v design.v
-- Compiling module testbench2
-- Compiling module design

Top level modules:
  testbench2

Analyzing design...
Optimizing 2 modules of which 0 are inlined:
-- Optimizing module design(t2)
-- Optimizing module testbench2(t2)

All of the modules within design.v compiled for testbench1 are identified by t1 within the library, whereas for testbench2 they are identified by t2. When the simulator loads testbench1, the instantiations from testbench1 reference the t1 versions of the code. Likewise, the instantiations from testbench2 reference the t2 versions. Therefore, you only need to invoke the simulator on the desired top-level module and the correct versions of code for the lower level instances are automatically loaded.

The only time that you need to specify a secondary name to the simulator is when you have multiple secondary names associated with a top-level module. If you omit the secondary name, then, by default, the simulator loads the most recently generated code (optimized or unoptimized) for the top-level module. You may explicitly specify a secondary name to load specific optimized code (specify "verilog" to load the unoptimized code). For example, suppose you have a top-level testbench that works in conjunction with each of several other top-level modules that only contain defparams that configure the design. In this case, you need to compile the entire design for each combination, using a different secondary name for each. For example,

% vlog -fast=c1 testbench.v design.v config1.v
-- Compiling module testbench
-- Compiling module design
-- Compiling module config1

Top level modules:
  testbench
  config1

Analyzing design...
Optimizing 3 modules of which 0 are inlined:
-- Optimizing module design(c1)
-- Optimizing module testbench(c1)
-- Optimizing module config1(c1)
Compiling for faster performance

% vlog -fast=c2 testbench.v design.v config2.v
  -- Compiling module testbench
  -- Compiling module design
  -- Compiling module config2

Top level modules:
  testbench
  config2

Analyzing design...
Optimizing 3 modules of which 0 are inlined:
  -- Optimizing module design(c2)
  -- Optimizing module testbench(c2)
  -- Optimizing module config2(c2)

Since the module "testbench" has two secondary names, you must specify which one you want when you invoke the simulator. For example,

% vsim 'testbench(c1)' config1

Note that it is not necessary to specify the secondary name for config1, because it has only one secondary name. If you omit the secondary name, the simulator defaults to loading the secondary name specified in the most recent compilation of the module.

If you prefer to use the "Load Design" dialog box to select top-level modules, then those modules compiled with -fast can be expanded to view their secondary names. Click on the one you wish to simulate.

To view the library contents, select Design > Browse Libraries to see the modules and their associated secondary names. Also, you can execute the vdir command (CR-247) on a specific module. For example,

VSIM 1> vdir design
  # MODULE design
  #     Optimized Module t1
  #     Optimized Module t2

Note: In some cases, an optimized module will have "__<n>" appended to its secondary name. This happens when multiple instantiations of a module require different versions of optimized code (for example, when the parameters of each instance are set to different values).
Enabling design object visibility with the +acc option

Some of the optimizations performed by the -fast option impact design visibility to both the user interface and the PLI routines. Many of the nets, ports, and registers are unavailable by name in user interface commands and in the various graphic interface windows. In addition, many of these objects do not have PLI Access handles, potentially affecting the operation of PLI applications. However, a handle is guaranteed to exist for any object that is an argument to a system task or function.

In the early stages of design, you may choose to compile without the -fast option so as to retain full debug capabilities. Alternatively, you may use one or more +acc options in conjunction with -fast to enable access to specific design objects. However, keep in mind that enabling design object access may reduce simulation performance.

The syntax for the +acc option is as follows:

+acc[=<spec>][+<module>][.]

<spec> is one or more of the following characters:

<table>
<thead>
<tr>
<th>&lt;spec&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Enable access to registers (including memories, integer, time, and real types).</td>
</tr>
<tr>
<td>n</td>
<td>Enable access to nets.</td>
</tr>
<tr>
<td>b</td>
<td>Enable access to individual bits of vector nets. This is necessary for PLI applications that require handles to individual bits of vector nets. Also, some user interface commands require this access if you need to operate on net bits.</td>
</tr>
<tr>
<td>p</td>
<td>Enable access to ports. This disables the module inlining optimization, and should be used for PLI applications that require access to port handles, or for debugging (see below).</td>
</tr>
<tr>
<td>c</td>
<td>Enable access to library cells. By default any Verilog module bracketed with a ‘celldefine / ‘endcelldefine may be optimized, and debug and PLI access may be limited. This option keeps module cell visibility.</td>
</tr>
</tbody>
</table>

If <spec> is omitted, then access is enabled for all objects.

<module> is a module name, optionally followed by "," to indicate all children of the module. Multiple modules are allowed, each separated by a "+". If no modules are specified, then all modules are affected.

If your design uses PLI applications that look for object handles in the design hierarchy, then it is likely that you will need to use the +acc option. For example, the built-in $dumpvars system task is an internal PLI application that requires handles to nets and registers so that it can call the PLI routine acc_vcl_add to monitor changes and dump the values to a VCD file. This requires that access is enabled for the nets and registers that it operates on. Suppose you want to dump all nets and registers in the entire design, and that you have the following $dumpvars call in your testbench (no arguments to $dumpvars means to dump everything in the entire design):

```verilog
initial $dumpvars;
```
Then you need to compile your design as follows to enable net and register access for all modules in the design:

```vlog
% vlog -fast +acc=rn testbench.v design.v
```

As another example, suppose you only need to dump nets and registers of a particular instance in the design (the first argument of 1 means to dump just the variables in the instance specified by the second argument):

```vlog
initial $dumpvars(1, testbench.u1);
```

Then you need to compile your design as follows (assuming testbench.u1 refers to a module named "design"):

```vlog
% vlog -fast +acc=rn+design testbench.v design.v
```

Finally, suppose you need to dump everything in the children instances of testbench.u1 (the first argument of 0 means to also include all children of the instance):

```vlog
initial $dumpvars(0, testbench.u1);
```

Then you need to compile your design as follows:

```vlog
% vlog -fast +acc=rn+design. testbench.v design.v
```

To gain maximum performance, it may be necessary to enable the minimum required access within the design.

### Using pre-compiled libraries

When using the -fast option, if the source code is unavailable for any of the modules referenced in a design, then you must search libraries for the precompiled modules using the -L or -Lf argument to `vlog` (CR-274). The compiler optimizes pre-compiled modules the same as if the source code is available. The optimized code for a pre-compiled module is written to the same library in which the module is found.

The compiler automatically searches libraries specified in the `uselib` directive (see Verilog-XL `uselib` compiler directive (UM-75)). If your design exclusively uses `uselib` directives to reference modules in other libraries, then you don’t need to specify library search options to the compiler.

**Note:** If you use `-L` or `-Lf` with the compiler, you must also you use them with `vsim` (CR-284) when you simulate the design.
Cell Libraries

Model Technology is the first Verilog simulation vendor to pass the ASIC Council’s Verilog test suite and achieve the "Library Tested and Approved" designation from Si2 Labs. This test suite is designed to ensure Verilog timing accuracy and functionality and is the first significant hurdle to complete on the way to achieving full ASIC vendor support. As a consequence, many ASIC and FPGA vendors’ Verilog cell libraries are compatible with ModelSim Verilog.

The cell models generally contain Verilog "specify blocks" that describe the path delays and timing constraints for the cells. See section 13 in the IEEE Std 1364-1995 for details on specify blocks, and section 14.5 for details on timing constraints. ModelSim Verilog fully implements specify blocks and timing constraints as defined in IEEE Std 1364 along with some Verilog-XL compatible extensions.

SDF timing annotation

ModelSim Verilog supports timing annotation from Standard Delay Format (SDF) files. See Chapter 12 - Standard Delay Format (SDF) Timing Annotation for details.

Delay modes

Verilog models may contain both distributed delays and path delays. The delays on primitives, UDPs, and continuous assignments are the distributed delays, whereas the port-to-port delays specified in specify blocks are the path delays. These delays interact to determine the actual delay observed. Most Verilog cells use path delays exclusively, with the distributed delays set to zero. For example,

```verbatim
delay_modes:module and2(y, a, b);
  input a, b;
  output y;

  and(y, a, b);
  specify
    (a => y) = 5;
    (b => y) = 5;
  endspecify
endmodule
```

In the above two-input "and” gate cell, the distributed delay for the "and” primitive is zero, and the actual delays observed on the module ports are taken from the path delays. This is typical for most cells, but a complex cell may require non-zero distributed delays to work properly. Even so, these delays are usually small enough that the path delays take priority over the distributed delays. The rule is that if a module contains both path delays and distributed delays, then the larger of the two delays for each path shall be used (as defined by the IEEE Std 1364). This is the default behavior, but you can specify alternate delay modes with compiler directives and options. These options and directives are compatible with Verilog-XL. Compiler delay mode options take precedence over delay mode directives in the source code.
**Distributed delay mode**

In distributed delay mode the specify path delays are ignored in favor of the distributed delays. Select this delay mode with the `+delay_mode_distributed` compiler option or the `delay_mode_distributed` compiler directive.

**Path delay mode**

In path delay mode the distributed delays are set to zero in any module that contains a path delay. Select this delay mode with the `+delay_mode_path` compiler option or the `delay_mode_path` compiler directive.

**Unit delay mode**

In unit delay mode the distributed delays are set to one (the unit is the time_unit specified in the `timescale` directive), and the specify path delays and timing constraints are ignored. Select this delay mode with the `+delay_mode_unit` compiler option or the `delay_mode_unit` compiler directive.

**Zero delay mode**

In zero delay mode the distributed delays are set to zero, and the specify path delays and timing constraints are ignored. Select this delay mode with the `+delay_mode_zero` compiler option or the `delay_mode_zero` compiler directive.
# System Tasks

The IEEE Std 1364 defines many system tasks as part of the Verilog language, and ModelSim Verilog supports all of these along with several non-standard Verilog-XL system tasks. The system tasks listed in this chapter are built into the simulator, although some designs depend on user-defined system tasks implemented with the Programming Language Interface (PLI) or Verilog Procedural Interface (VPI). If the simulator issues warnings regarding undefined system tasks, then it is likely that these system tasks are defined by a PLI/VPI application that must be loaded by the simulator.

## IEEE Std 1364 system tasks

The following system tasks are described in detail in the IEEE Std 1364.

<table>
<thead>
<tr>
<th>Timescale tasks</th>
<th>Simulator control tasks</th>
<th>Simulation time functions</th>
<th>Command line input</th>
</tr>
</thead>
<tbody>
<tr>
<td>$printtimescale</td>
<td>$finish</td>
<td>$realtime</td>
<td>$test$plusargs</td>
</tr>
<tr>
<td>$timeformat</td>
<td>$stop</td>
<td>$stime</td>
<td>$value$plusargs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$time</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Probabilistic distribution functions</th>
<th>Conversion functions</th>
<th>Stochastic analysis tasks</th>
<th>Timing check tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>$dist_chi_square</td>
<td>$bitstoreal</td>
<td>$q_add</td>
<td>$hold</td>
</tr>
<tr>
<td>$dist_erlang</td>
<td>$itor</td>
<td>$q_exam</td>
<td>$nochange</td>
</tr>
<tr>
<td>$dist_exponential</td>
<td>$realtobits</td>
<td>$q_full</td>
<td>$period</td>
</tr>
<tr>
<td>$dist_normal</td>
<td>$rtoi</td>
<td>$q_initialize</td>
<td>$recovery</td>
</tr>
<tr>
<td>$dist_poisson</td>
<td>$signed</td>
<td>$q_remove</td>
<td>$setup</td>
</tr>
<tr>
<td>$dist_t</td>
<td>$unsigned</td>
<td></td>
<td>$setuphold</td>
</tr>
<tr>
<td>$dist_uniform</td>
<td></td>
<td></td>
<td>$skew</td>
</tr>
<tr>
<td>$random</td>
<td></td>
<td></td>
<td>$width</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$removal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$recrem</td>
</tr>
<tr>
<td>Display tasks</td>
<td>PLA modeling tasks</td>
<td>Value change dump (VCD) file tasks</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------</td>
<td>-----------------------------------</td>
<td></td>
</tr>
<tr>
<td>$display</td>
<td>$async$and$array</td>
<td>$dumpall</td>
<td></td>
</tr>
<tr>
<td>$displayb</td>
<td>$async$nand$array</td>
<td>$dumpfile</td>
<td></td>
</tr>
<tr>
<td>$displayh</td>
<td>$async$or$array</td>
<td>$dumpflush</td>
<td></td>
</tr>
<tr>
<td>$displayo</td>
<td>$async$nor$array</td>
<td>$dumplimit</td>
<td></td>
</tr>
<tr>
<td>$monitor</td>
<td>$async$and$plane</td>
<td>$dumpoff</td>
<td></td>
</tr>
<tr>
<td>$monitorb</td>
<td>$async$nand$plane</td>
<td>$dumpon</td>
<td></td>
</tr>
<tr>
<td>$monitorh</td>
<td>$async$or$plane</td>
<td>$dumpvars</td>
<td></td>
</tr>
<tr>
<td>$monitoro</td>
<td>$async$nor$plane</td>
<td>$dumpportson</td>
<td></td>
</tr>
<tr>
<td>$monitoroff</td>
<td>$sync$and$array</td>
<td>$dumpportsoff</td>
<td></td>
</tr>
<tr>
<td>$monitoron</td>
<td>$sync$nand$array</td>
<td>$dumpportsall</td>
<td></td>
</tr>
<tr>
<td>$strobe</td>
<td>$sync$or$array</td>
<td>$dumpportsflush</td>
<td></td>
</tr>
<tr>
<td>$strobeb</td>
<td>$sync$nor$array</td>
<td>$dumpports</td>
<td></td>
</tr>
<tr>
<td>$strobeh</td>
<td>$sync$and$plane</td>
<td>$dumpportslimit</td>
<td></td>
</tr>
<tr>
<td>$strobeo</td>
<td>$sync$nand$plane</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$write</td>
<td>$sync$or$plane</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$writeb</td>
<td>$sync$nor$plane</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$writeh</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$writeo</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
File I/O tasks

$fclose  $fopen  $fwriteh
$fdisplay  $fread  $fwriteo
$fdisplayb  $fscanf  $readmemb
$fdisplayh  $fseek  $readmemh
$fdisplayo  $fstrobe  $srewind
$error  $fstrogeb  $sdf_annotate
$fflush  $fstrobeh  $sformat
$fgetc  $fstrobeo  $sscanf
$fgets  $ftell  $swrite
$fmonitor  $fwrite  $swriteb
$fmonitorb  $fwriteb  $swriteh
$fmonitorh  $fwriteo  $swriteo
$fmonitoro  $ungetc

**Note:** $readmemb and $readmemh match the behavior of Verilog-XL rather than IEEE Std 1364. Specifically, it loads data into memory starting with the lowest address. For example, whether you make the declaration `memory[127:0]` or `memory[0:127]`, ModelSim will load data starting at address 0 and work upwards to address 127.
Verilog-XL compatible system tasks

The following system tasks are provided for compatibility with Verilog-XL. Although they are not part of the IEEE standard, they are described in an annex of the IEEE Std 1364.

\$countdrivers
\$getpattern
\$sreadmemb
\$sreadmemh

The following system tasks are also provided for compatibility with Verilog-XL; they are not described in the IEEE Std 1364.

\$deposit(variable, value);
This system task sets a Verilog register or net to the specified value. \texttt{variable} is the register or net to be changed; \texttt{value} is the new value for the register or net. The value remains until there is a subsequent driver transaction or another \$deposit task for the same register or net. This system task operates identically to the ModelSim force -deposit command.

\$system("operating system shell command");
This system task executes the specified operating system shell command and displays the result. For example, to list the contents of the working directory on Unix:

\$system("ls");

The following system tasks are extended to provide additional functionality for negative timing constraints and an alternate method of conditioning, as does Verilog-XL.

\$recovery(reference event, data_event, removal_limit, recovery_limit, [notifier], [tstamp_cond], [tcheck_cond], [delayed_reference], [delayed_data])
The \$recovery system task normally takes a recovery_limit as the third argument and an optional notifier as the fourth argument. By specifying a limit for both the third and fourth arguments, the \$recovery timing check is transformed into a combination removal and recovery timing check similar to the \$recrem timing check. The only difference is that the removal_limit and recovery_limit are swapped.

\$setuphold(clk_event, data_event, setup_limit, hold_limit, [notifier], [tstamp_cond], [tcheck_cond], [delayed_clk], [delayed_data])
The \$setuphold argument conditions the data_event for the setup check and the clk_event for the hold check. This alternate method of conditioning precludes specifying conditions in the clk_event and data_event arguments.

The tstamp_cond argument conditions the data_event for the hold check and the clk_event for the setup check. This alternate method of conditioning precludes specifying conditions in the clk_event and data_event arguments.

The delayed_clk argument is a net that is continuously assigned the value of the net specified in the clk_event. The delay is non-zero if the setup_limit is negative, zero otherwise.

The delayed_data argument is a net that is continuously assigned the value of the net specified in the data_event. The delay is non-zero if the hold_limit is negative, zero otherwise.

The delayed_clk and delayed_data arguments are provided to ease the modeling of devices that may have negative timing constraints. The model’s logic should reference the delayed_clk and delayed_data nets in place of the normal clk and data nets. This
ensures that the correct data is latched in the presence of negative constraints. The simulator automatically calculates the delays for delayed_clk and delayed_data such that the correct data is latched as long as a timing constraint has not been violated.

The following system tasks are Verilog-XL system tasks that are not implemented in ModelSim Verilog, but have equivalent simulator commands.

\$input("filename")
This system task reads commands from the specified filename. The equivalent simulator command is \texttt{do <filename>}.

\$list\{(hierarchical_name)\}
This system task lists the source code for the specified scope. The equivalent functionality is provided by selecting a module in the graphic interface Structure window. The corresponding source code is displayed in the source window.

\$reset
This system task resets the simulation back to its time 0 state. The equivalent simulator command is \texttt{restart}.

\$restart("filename")
This system task sets the simulation to the state specified by filename, saved in a previous call to \$save. The equivalent simulator command is \texttt{restore <filename>}.

\$save("filename")
This system task saves the current simulation state to the file specified by filename. The equivalent simulator command is \texttt{checkpoint <filename>}.

\$scope(hierarchical_name)
This system task sets the interactive scope to the scope specified by hierarchical_name. The equivalent simulator command is \texttt{environment <pathname>}.

\$showscopes
This system task displays a list of scopes defined in the current interactive scope. The equivalent simulator command is \texttt{show}.

\$showvars
This system task displays a list of registers and nets defined in the current interactive scope. The equivalent simulator command is \texttt{show}. 
$init_signal_spy

The $init_signal_spy() system task mirrors the value of a VHDL signal or Verilog register/wire (called the spy_object) onto an existing Verilog register or VHDL signal (called the dest_object). This system task allows you to reference VHDL signals at any level of hierarchy from within a Verilog module; or, reference Verilog registers/wires at any level of hierarchy from within a Verilog module when there is an interceding VHDL block.

This system task works only in ModelSim versions 5.5 and newer.

Syntax

$init_signal_spy(spy_object, dest_object, verbose)

Returns

Nothing

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spy_object</td>
<td>string</td>
<td>Required. A full hierarchical path (or relative path with reference to the calling block) to a VHDL signal or Verilog register/wire. Use the path separator to which your simulation is set (i.e., &quot;/&quot; or &quot;.&quot;). A full hierarchical path must begin with a &quot;/&quot; or &quot;.&quot;. The path must be contained within double quotes.</td>
</tr>
<tr>
<td>dest_object</td>
<td>string</td>
<td>Required. A full hierarchical path (or relative path with reference to the calling block) to a Verilog register or VHDL signal. Use the path separator to which your simulation is set (i.e., &quot;/&quot; or &quot;.&quot;). A full hierarchical path must begin with a &quot;/&quot; or &quot;.&quot;. The path must be contained within double quotes.</td>
</tr>
<tr>
<td>verbose</td>
<td>integer</td>
<td>Optional. Possible values are 0 or 1. Specifies whether you want a message reported in the Transcript stating that the spy_object’s value is mirrored onto the dest_object. Default is 0, no message.</td>
</tr>
</tbody>
</table>

Limitations

- When mirroring the value of a VHDL signal onto a Verilog register, the VHDL signal must be of type bit, bit_vector, std_logic, or std_logic_vector.
- Mirroring slices or single bits of a vector is not supported. If you do reference a slice or bit of a vector, the function will assume that you are referencing the entire vector.
Example

module ...
...
reg top_sig1;
...
initial
begin
  $init_signal_spy("/top/uut/inst1/sig1";/top_sig1", 1);
  end
...
endmodule

In this example, the value of "/top/uut/inst1/sig1" will be mirrored onto "/top_sig1".
Compiler Directives

ModelSim Verilog supports all of the compiler directives defined in the IEEE Std 1364 and some additional Verilog-XL compiler directives for compatibility.

Many of the compiler directives (such as `timescale`) take effect at the point they are defined in the source code and stay in effect until the directive is redefined or until it is reset to its default by a `resetall` directive. The effect of compiler directives spans source files, so the order of source files on the compilation command line could be significant. For example, if you have a file that defines some common macros for the entire design, then you might need to place it first in the list of files to be compiled.

The `resetall` directive affects only the following directives by resetting them back to their default settings (this information is not provided in the IEEE Std 1364):

`celldefine`  
`default_decay_time`  
`define_nettype`  
`define`  
`else`  
`endcelldefine`  
`endif`  
`ifdef`  
`ifndef`  
`include`  
`line`  
`nounconnected_drive`  
`resetall`  
`timescale`  
`unconnected_drive`  
`undef`  

ModelSim Verilog implicitly defines the following macro:

`define MODEL_TECH

IEEE Std 1364 compiler directives

The following compiler directives are described in detail in the IEEE Std 1364.

`celldefine`  
`default_decay_time`  
`define_nettype`  
`define`  
`else`  
`endcelldefine`  
`endif`  
`ifdef`  
`ifndef`  
`include`  
`line`  
`nounconnected_drive`  
`resetall`  
`timescale`  
`unconnected_drive`  
`undef`

Verilog-XL compatible compiler directives

The following compiler directives are provided for compatibility with Verilog-XL.

`default_decay_time <time>

This directive specifies the default decay time to be used in trireg net declarations that do not explicitly declare a decay time. The decay time can be expressed as a real or integer number, or as infinite to specify that the charge never decays.
'delay_mode_distributed
This directive disables path delays in favor of distributed delays. See Delay modes (UM-90) for details.

'delay_mode_path
This directive sets distributed delays to zero in favor of path delays. See Delay modes (UM-90) for details.

'delay_mode_unit
This directive sets path delays to zero and non-zero distributed delays to one time unit. See Delay modes (UM-90) for details.

'delay_mode_zero
This directive sets path delays and distributed delays to zero. See Delay modes (UM-90) for details.

'uselib
This directive is an alternative to the -v, -y, and +libext source library compiler options. See Verilog-XL 'uselib compiler directive (UM-75) for details.

The following Verilog-XL compiler directives are silently ignored by ModelSim Verilog. Many of these directives are irrelevant to ModelSim Verilog, but may appear in code being ported from Verilog-XL.

'accelerate
'autoexpand_vectornets
'disable_portfaults
'enable_portfaults
'endprotect
'expand_vectornets
'noaccelerate
'noexpand_vectornets
'noremove_gatenames
'noremove_netnames
'nosuppress_faults
'protect
'remove_gatenames
'remove_netnames
'suppress_faults

The following Verilog-XL compiler directives produce warning messages in ModelSim Verilog. These are not implemented in ModelSim Verilog, and any code containing these directives may behave differently in ModelSim Verilog than in Verilog-XL.

'default_trireg_strength
'signed
'unsigned
Verilog PLI/VPI

The Verilog PLI (Programming Language Interface) and VPI (Verilog Procedural Interface) both provide a mechanism for defining system tasks and functions that communicate with the simulator through a C procedural interface. There are many third party applications available that interface to Verilog simulators through the PLI (see Third party PLI applications (UM-113)). In addition, you may write your own PLI/VPI applications.

ModelSim Verilog implements the PLI as defined in the IEEE Std 1364, with the exception of the acc_handle_datapath routine. We did not implement the acc_handle_datapath routine because the information it returns is more appropriate for a static timing analysis tool. In version 5.5d, the VPI is partially implemented as defined in the IEEE Std 1364. The list of currently supported functionality can be found in the following directory:

<install_dir>/modeltech/docs/technotes/Verilog_VPI.note.

The IEEE Std 1364 is the reference that defines the usage of the PLI/VPI routines. This manual only describes details of using the PLI/VPI with ModelSim Verilog.

Registering PLI applications

Each PLI application must register its system tasks and functions with the simulator, providing the name of each system task and function and the associated callback routines. Since many PLI applications already interface to Verilog-XL, ModelSim Verilog PLI applications make use of the same mechanism to register information about each system task and function in an array of s_tfcell structures. This structure is declared in the veriuser.h include file as follows:

```c
typedef int (*p_tffn)();

typedef struct t_tfcell {
    short type;/* USERTASK, USERFUNCTION, or USERREALFUNCTION */
    short data;/* passed as data argument of callback function */
    p_tffn checktf; /* argument checking callback function */
    p_tffn sizetf; /* function return size callback function */
    p_tffn calltf; /* task or function call callback function */
    p_tffn misctf; /* miscellaneous reason callback function */
    char *tfname; /* name of system task or function */
    /* The following fields are ignored by ModelSim Verilog */
    int forwref;
    char *tfveritool;
    char *tferrmessage;
    int hash;
    struct t_tfcell *left_p;
    struct t_tfcell *right_p;
    char *namecell_p;
    int warning_printed;
} s_tfcell, *p_tfcell;
```

The various callback functions (checktf, sizetf, calltf, and misctf) are described in detail in the IEEE Std 1364. The simulator calls these functions for various reasons. All callback functions are optional, but most applications contain at least the calltf function, which is called when the system task or function is executed in the Verilog code. The first argument to the callback functions is the value supplied in the data field (many PLI applications don't
use this field). The type field defines the entry as either a system task (USERTASK) or a system function that returns either a register (USERFUNCTION) or a real (USERREALFUNCTION). The tfname field is the system task or function name (it must begin with $). The remaining fields are not used by ModelSim Verilog.

On loading of a PLI application, the simulator first looks for an init_usertfs function, and then a veriusertfs array. If init_usertfs is found, the simulator calls that function so that it can call mti_RegisterUserTF() for each system task or function defined. The mti_RegisterUserTF() function is declared in veriuser.h as follows:

```c
void mti_RegisterUserTF(p_tfcell usertf);
```

The storage for each usertf entry passed to the simulator must persist throughout the simulation because the simulator de-references the usertf pointer to call the callback functions. It is recommended that you define your entries in an array, with the last entry set to 0. If the array is named veriusertfs (as is the case for linking to Verilog-XL), then you don’t have to provide an init_usertfs function, and the simulator will automatically register the entries directly from the array (the last entry must be 0). For example,

```c
s_tfcell veriusertfs[] = {
    {usertask, 0, 0, 0, abc_calltf, 0, "$abc"},
    {usertask, 0, 0, 0, xyz_calltf, 0, "$xyz"},
    {0}  /* last entry must be 0 */
};
```

Alternatively, you can add an init_usertfs function to explicitly register each entry from the array:

```c
void init_usertfs()
{
    p_tfcell usertf = veriusertfs;
    while (usertf->type)  
        mti_RegisterUserTF(usertf++);
}
```

It is an error if a PLI shared library does not contain a veriusertfs array or an init_usertfs function.

Since PLI applications are dynamically loaded by the simulator, you must specify which applications to load (each application must be a dynamically loadable library, see “Compiling and linking PLI/VPI applications” (UM-104)). The PLI applications are specified as follows:

- As a list in the Veriuser entry in the modelsim.ini file:
  ```ini
  Veriuser = pliapp1.so pliapp2.so pliappn.so
  ```

- As a list in the PLIOBJS environment variable:
  ```
  % setenv PLIOBJS "pliapp1.so pliapp2.so pliappn.so"
  ```

- As a -pli option to the simulator (multiple options are allowed):
  ```
  -pli pliapp1.so -pli pliapp2.so -pli pliappn.so
  ```

The various methods of specifying PLI applications can be used simultaneously. The libraries are loaded in the order listed above. Environment variable references can be used in the paths to the libraries in all cases.
Registering VPI applications

Each VPI application must register its system tasks and functions and its callbacks with the simulator. To accomplish this, one or more user-created registration routines must be called at simulation startup. Each registration routine should make one or more calls to \texttt{vpi\_register\_systf()} to register user-defined system tasks and functions and \texttt{vpi\_register\_cb()} to register callbacks. The registration routines must be placed in a table named \texttt{vlog\_startup\_routines} so that the simulator can find them. The table must be terminated with a 0 entry.

\textbf{Example}

\begin{verbatim}
void RegisterMySystfs( void )
{
    s_cb_data callback;
s_vpi_systf_data systf_data;
    systf_data.type = vpiSysFunc;
systf_data.sysfunctype = vpiSizedFunc;
systf_data.tfname = "$myfunc";
systf_data.calltf = MyFuncCalltf;
systf_data.compiletf = MyFuncCompiletf;
systf_data.sizetf = MyFuncSizetf;
systf_data.user_data = 0;
    vpi_register_systf( &systf_data );

    callback.reason = cbEndOfCompile;
callback.cb_rtn = MyEndOfCompCB;
callback.user_data = 0;
    (void) vpi_register_cb( &callback );

    callback.reason = cbStartOfSimulation;
callback.cb_rtn = MyStartOfSimCB;
callback.user_data = 0;
    (void) vpi_register_cb( &callback );
}

void (*vlog\_startup\_routines[ ] ) = {
    RegisterMySystfs,
    0 /* last entry must be 0 */
};
\end{verbatim}

Loading VPI applications into the simulator is the same as described in \textbf{Registering PLI applications} (UM-101).

\textbf{Registering PLI applications}

Each VPI application must register its system tasks and functions and its callbacks with the simulator. To accomplish this, one or more user-created registration routines must be called at simulation startup. Each registration routine should make one or more calls to \texttt{vpi\_register\_systf()} to register user-defined system tasks and functions and \texttt{vpi\_register\_cb()} to register callbacks. The registration routines must be placed in a table named \texttt{vlog\_startup\_routines} so that the simulator can find them. The table must be terminated with a 0 entry.

\textbf{Example}

\begin{verbatim}
void MyFuncCalltf( PLI_BYTE8 *user_data )
{
    ... }

void MyFuncCompiletf( PLI_BYTE8 *user_data )
{
    ... }

void MyFuncSizetf( PLI_BYTE8 *user_data )
{
    ... }

void MyEndOfCompCB( p_cb_data cb_data_p )
{
    ... }

void MyStartOfSimCB( p_cb_data cb_data_p )
{
    ... }

void RegisterMySystfs( void )
{
    s_cb_data callback;
    s_vpi_systf_data systf_data;
    systf_data.type = vpiSysFunc;
systf_data.sysfunctype = vpiSizedFunc;
systf_data.tfname = "$myfunc";
systf_data.calltf = MyFuncCalltf;
systf_data.compiletf = MyFuncCompiletf;
systf_data.sizetf = MyFuncSizetf;
systf_data.user_data = 0;
    vpi_register_systf( &systf_data );

    callback.reason = cbEndOfCompile;
callback.cb_rtn = MyEndOfCompCB;
callback.user_data = 0;
    (void) vpi_register_cb( &callback );

    callback.reason = cbStartOfSimulation;
callback.cb_rtn = MyStartOfSimCB;
callback.user_data = 0;
    (void) vpi_register_cb( &callback );
}
\end{verbatim}

Loading VPI applications into the simulator is the same as described in \textbf{Registering PLI applications} (UM-101).
PLI and VPI applications can co-exist in the same application object file. In such cases, the applications are loaded at startup as follows:

- If an init_usertfs() function exists, then it is executed and only those system tasks and functions registered by calls to mti_RegisterUserTF() will be defined.
- If an init_usertfs() function does not exist but a veriusertfs table does exist, then only those system tasks and functions listed in the veriusertfs table will be defined.
- If an init_usertfs() function does not exist and a veriusertfs table does not exist, but a vlog_startup_routines table does exist, then only those system tasks and functions and callbacks registered by functions in the vlog_startup_routines table will be defined.

As a result, when PLI and VPI applications exist in the same application object file, they must be registered in the same manner. VPI registration functions that would normally be listed in a vlog_startup_routines table can be called from an init_usertfs() function instead.

**Compiling and linking PLI/VPI applications**

ModelSim Verilog uses operating system calls to dynamically load PLI and VPI applications when the simulator loads a design. Therefore, the applications must be compiled and linked for dynamic loading on a specific operating system. The PLI/VPI routines are declared in the include files located in the ModelSim <install_dir>/modeltech/include directory. The acc_user.h file declares the ACC routines, the veriuser.h file declares the TF routines, and the vpi_user.h file declares the VPI routines.

The following instructions assume that the PLI or VPI application is in a single source file. For multiple source files, compile each file as specified in the instructions and link all of the resulting object files together with the specified link instructions.

**Windows 95/98/2000/NT/Me platforms**

Under Windows ModelSim loads a 32-bit dynamically linked library for each PLI/VPI application. The following compile and link steps are used to create the necessary.dll file (and other supporting files) using the Microsoft Visual C/C++ compiler.

```bash
cl -c -I<install_dir>/modeltech\include app.c
link -dll -export:<init_function> app.obj \  
<install_dir>\modeltech\win32\mtipli.lib out:app.exe
```

For the Verilog PLI, the <init_function> should be "init_usertfs". Alternatively, if there is no init_usertfs function, the <init_function> specified on the command line should be "veriusertfs". For the Verilog VPI, the <init_function> should be "vlog_startup_routines". These requirements ensure that the appropriate symbol is exported, and thus ModelSim can find the symbol when it dynamically loads the DLL.

The PLI and VPI have been tested with DLLs built using Microsoft Visual C/C++ compiler version 4.1 or greater.

The gcc compiler cannot be used to compile PLI/VPI applications under Windows. This is because gcc does not support the Microsoft .lib/.dll format.
Linux platform
Under Linux, ModelSim loads shared objects. Use these gcc or cc compiler commands to create a shared object:

gcc compiler:
```bash
gcc -c -I/<install_dir>/modeltech/include app.c
ld -shared -E -o app.so app.o
```

cc compiler:
```bash
cc -c -I/<install_dir>/modeltech/include app.c
ld -shared -E -o app.so app.o
```

32-bit Solaris platform
Under SUN Solaris, ModelSim loads shared objects. Use these gcc or cc compiler commands to create a shared object:

gcc compiler:
```bash
gcc -c -I/<install_dir>/modeltech/include app.c
ld -G -B symbolic -o app.so app.o
```

cc compiler:
```bash
cc -c -I/<install_dir>/modeltech/include app.c
ld -G -B symbolic -o app.so app.o
```

Note: When using -B symbolic with ld, all symbols are first resolved within the shared library at link time. This will result in a list of undefined symbols. This is only a warning for shared libraries and can be ignored.

If app.so is not in your current directory you must tell Solaris where to search for the shared object. You can do this one of two ways:

- Add a path before app.so in the foreign attribute specification. (The path may include environment variables.)
- Put the path in a UNIX shell environment variable:
  ```bash
  LD_LIBRARY_PATH= <library path without filename>
  ```

64-bit Solaris platform
On a 64-bit Sun system, use the following cc compiler commands to prepare PLI/VPI code for dynamic linking with ModelSim:

```bash
cc -v -xarch=v9 -O -I$/<install_dir>/modeltech/include -c app.c
ld -G app.o -o app.so
```
32-bit HP700 platform

ModelSim loads shared libraries on the HP700 workstation. A shared library is created by creating object files that contain position-independent code (use the +z or -fpic compiler option) and by linking as a shared library (use the -b linker option). Use these gcc or cc compiler commands:

**gcc compiler:**
```
gcc -c -fpic -I/<install_dir>/modeltech/include app.c
dl -b -o app.sl app.o -lc
```

**cc compiler:**
```
c -c +z -I/<install_dir>/modeltech/include app.c
dl -b -o app.sl app.o -lc
```

Note that -fpic may not work with all versions of gcc.

for HP-UX 11.0 users

If you are building the PLI/VPI library under HP-UX 11.0, you should not specify the "-lc" option to the invocation of ld, since this will cause an incorrect version of the standard C library to be loaded.

In other words, build libraries like this:
```
c -c +z -I/<install_dir>/modeltech/include app.c
dl -b -o app.sl app.o
```

If you receive the error "Exec format error" when the simulator is trying to load a PLI/VPI library, then you have most likely built under 11.0 and specified the "-lc" option. Just rebuild without "-lc" (or rebuild on an HP-UX 10.0 machine).

64-bit HP platform

On a 64-bit HP system, use the following cc compiler commands to prepare PLI/VPI code for dynamic linking with ModelSim:
```
cc -v +DA2.0W -O -I<install_dir>/modeltech/include -c app.c
ld -b app.o -o app.so
```
32-bit IBM RS/6000 platform

ModelSim loads shared libraries on the IBM RS/6000 workstation. The shared library must import ModelSim’s PLI/VPI symbols, and it must export the PLI or VPI application’s initialization function or table. ModelSim’s export file is located in the ModelSim installation directory in `rs6000/mti_exports`.

If your PLI/VPI application uses anything from a system library, you’ll need to specify that library when you link your PLI/VPI application. For example, to use the standard C library, specify `-lc` to the ‘ld’ command. The resulting object must be marked as shared reentrant using these `gcc` or `cc` compiler commands for AIX 4.x:

**gcc compiler:**

```
gcc -c -I/<install_dir>/modeltech/include app.c
ld -o app.sl app.o -bE:app.exp \
   -bI:/<install_dir>/modeltech/rs6000/mti_exports -bM:SRE -bnoentry -lc
```

**cc compiler:**

```
cc -c -I/<install_dir>/modeltech/include app.c
ld -o app.sl app.o -bE:app.exp \
   -bI:/<install_dir>/modeltech/rs6000/mti_exports -bM:SRE -bnoentry -lc
```

The app.exp file must export the PLI/VPI initialization function or table. For the PLI, the exported symbol should be “init_userfs”. Alternatively, if there is no init_usertfs function, then the exported symbol should be “veriusertfs”. For the VPI, the exported symbol should be “vlog_startup_routines”. These requirements ensure that the appropriate symbol is exported, and thus ModelSim can find the symbol when it dynamically loads the shared object.

64-bit IBM RS/6000 platform

Only version 4.3 of AIX supports the 64-bit platform. A gcc 64-bit compiler is not available at this time. The cc commands are as follows:

```
cc -c -q64 -I/<install_dir>/modeltech/include app.c
ld -o app.sl app.o -b64 -bE:app.exports \
   -bI:/<install_dir>/modeltech/rs64/mti_exports -bM:SRE -bnoentry -lc
```

**Note:** When using AIX 4.3 in 32-bit mode, you must add the switch `-d use_inttypes` to the compile command lines. This switch prevents a name conflict that occurs between `inttypes.h` and `mti.h`. 
Using 64-bit ModelSim with 32-bit PLI/VPI Applications

If you have 32-bit PLI/VPI applications and wish to use 64-bit ModelSim, you will need to port your code to 64 bits by moving from the ILP32 data model to the LP64 data model. We strongly recommend that you consult the following 64-bit porting guides for the appropriate platform:

Sun

Solaris 7 64-bit Developer's Guide
http://docs.sun.com:80/ab2/coll.45.10/SOL64TRANS/

HP

HP-UX 64-bit Porting and Transition Guide
http://docs.hp.com:80/dynaweb/hpux11/hpuxen1a/0462/@Generic__BookView

HP-UX 11.x Software Transition Kit
http://software.hp.com/STK/

IBM

AIX 64-bit Migration Guide

Specifying the PLI/VPI file to load

The PLI applications are specified as follows:

- As a list in the Veriuser entry in the modelsim.ini file:
  
  Veriuser = pliapp1.so pliapp2.so pliappn.so

- As a list in the PLIOBJS environment variable:
  
  % setenv PLIOBJS "pliapp1.so pliapp2.so pliappn.so"

- As a -pli option to the simulator (multiple options are allowed):
  
  -pli pliapp1.so -pli pliapp2.so -pli pliappn.so

▶ Note: On Windows platforms, the file names shown above should end with ".dll" rather than ".so".

The various methods of specifying PLI applications can be used simultaneously. The libraries are loaded in the order listed above. Environment variable references can be used in the paths to the libraries in all cases.

See also Appendix A - ModelSim Variables for more information on the modelsim.ini file.
**PLI example**

The following example is a trivial, but complete PLI application.

**hello.c:**

```c
#include "veriuser.h"
static hello()
{
    io_printf("Hi there\n");
}
s_tfcell veriusertfs[] = {
    {usertask, 0, 0, 0, hello, 0, "$hello"},
    {0} /* last entry must be 0 */
};
```

**hello.v:**

```vhdl
module hello;
    initial $hello;
endmodule
```

Compile the PLI code for the Solaris operating system:

```bash
% cc -c -I<install_dir>/modeltech/include hello.c
% ld -G -o hello.sl hello.o
```

Compile the Verilog code:

```bash
% vlib work
% vlog hello.v
```

Simulate the design:

```bash
% vsim -c -pli hello.sl hello
# Loading work.hello
# Loading ./hello.sl
VSIM 1> run -all
# Hi there
VSIM 2> quit
```
**VPI example**

The following example is a trivial, but complete VPI application.

```c
#include "vpi_user.h"
static hello() {
    vpi_printf( "Hello world!\n" );
}

void RegisterMyTfs( void ) {
    s_vpi_systf_data systf_data;
    systf_data.type = vpiSysTask;
    systf_data.sysfunctype = vpiSysTask;
    systf_data.tfname = "$hello";
    systf_data.calltf = hello;
    systf_data.compiletf = 0;
    systf_data.sizetf = 0;
    systf_data.user_data = 0;
    vpi_register_systf( &systf_data );
    vpi_free_object( systf_handle );
}

void (*vlog_startup_routines[])( ) = { RegisterMyTfs, 0 };
```

```v
module hello;
    initial $hello;
endmodule
```

Compile the VPI code for the Solaris operating system:

```
% gcc -c -I<install_dir>/include hello.c
% ld -G -o hello.sl hello.o
```

Compile the Verilog code:

```
% vlib work
% vlog hello.v
```

Simulate the design:

```
% vsim -c -pli hello.sl hello
# Loading work.hello
# Loading ./hello.sl
VSIM 1> run -all
# Hello world!
VSIM 2> quit
```

▶ **Note:** A general VPI example can be found in `<install_dir>/modeltech/examples/vpi`. 
The PLI callback reason argument

The second argument to a PLI callback function is the reason argument. The values of the various reason constants are defined in the veriuser.h include file. See IEEE Std 1364 for a description of the reason constants. The following details relate to ModelSim Verilog, and may not be obvious in the IEEE Std 1364. Specifically, the simulator passes the reason values to the misctf callback functions under the following circumstances:

**reason_endofcompile**
For the completion of loading the design.

**reason_finish**
For the execution of the $finish system task or the quit command.

**reason_startofsave**
For the start of execution of the checkpoint command, but before any of the simulation state has been saved. This allows the PLI application to prepare for the save, but it shouldn't save its data with calls to tf_write_save until it is called with reason_save.

**reason_save**
For the execution of the checkpoint command. This is when the PLI application must save its state with calls to tf_write_save.

**reason_startofrestart**
For the start of execution of the restore command, but before any of the simulation state has been restored. This allows the PLI application to prepare for the restore, but it shouldn't restore its state with calls to tf_read_restart until it is called with reason_restart. The reason_startofrestart value is passed only for a restore command, and not in the case that the simulator is invoked with -restore.

**reason_restart**
For the execution of the restore command. This is when the PLI application must restore its state with calls to tf_read_restart.

**reason_reset**
For the execution of the restart command. This is when the PLI application should free its memory and reset its state. We recommend that all PLI applications reset their internal state during a restart as the shared library containing the PLI code might not be reloaded. (See the -keeploaded (CR-286) and -keeploadedrestart (CR-286) vsim arguments for related information.)

**reason_endofreset**
For the completion of the restart command, after the simulation state has been reset but before the design has been reloaded.

**reason_interactive**
For the execution of the $stop system task or any other time the simulation is interrupted and waiting for user input.

**reason_scope**
For the execution of the environment command or selecting a scope in the structure window. Also for the call to acc_set_interactive_scope if the callback_flag argument is non-zero.

**reason_paramvc**
For the change of value on the system task or function argument.
reason_synch
   For the end of time step event scheduled by tf_synchronize.

reason_rosynch
   For the end of time step event scheduled by tf_rosynchronize.

reason_reactivate
   For the simulation event scheduled by tf_setdelay.

reason_paramdrc
   Not supported in ModelSim Verilog.

reason_force
   Not supported in ModelSim Verilog.

reason_release
   Not supported in ModelSim Verilog.

reason_disable
   Not supported in ModelSim Verilog.

The sizetf callback function

A user-defined system function specifies the width of its return value with the sizetf
callback function, and the simulator calls this function while loading the design. The
following details on the sizetf callback function are not found in the IEEE Std 1364:

- If you omit the sizetf function, then a return width of 32 is assumed.
- The sizetf function should return 0 if the system function return value is of Verilog type
  "real".
- The sizetf function should return -32 if the system function return value is of Verilog type
  "integer".

PLI object handles

Many of the object handles returned by the PLI ACC routines are pointers to objects that
naturally exist in the simulation data structures, and the handles to these objects are valid
throughout the simulation, even after the acc_close() routine is called. However, some of
the objects are created on demand, and the handles to these objects become invalid after
acc_close() is called. The following object types are created on demand in ModelSim
Verilog:

- accOperator (acc_handle_condition)
- accWirePath (acc_handle_path)
- accTerminal (acc_handle_terminal, acc_next_cell_load, acc_next_driver, and
  acc_next_load)
- accPathTerminal (acc_next_input and acc_next_output)
- accTchkTerminal (acc_handle_tchkarg1 and acc_handle_tchkarg2)
- accPartSelect (acc_handle_conn, acc_handle_pathin, and acc_handle_pathout)
- accRegBit (acc_handle_by_name, acc_handle_tfarg, and acc_handle_itfarg)

If your PLI application uses these types of objects, then it is important to call acc_close()
to free the memory allocated for these objects when the application is done using them.

If your PLI application places value change callbacks on accRegBit or accTerminal objects,
*do not* call acc_close() while these callbacks are in effect.
Third party PLI applications

Many third party PLI applications come with instructions on using them with ModelSim Verilog. Even without the instructions, it is still likely that you can get it to work with ModelSim Verilog as long as the application uses standard PLI routines. The following guidelines are for preparing a Verilog-XL PLI application to work with ModelSim Verilog.

Generally, a Verilog-XL PLI application comes with a collection of object files and a veriuser.c file. The veriuser.c file contains the registration information as described above in "Registering PLI applications". To prepare the application for ModelSim Verilog, you must compile the veriuser.c file and link it to the object files to create a dynamically loadable object (see "Compiling and linking PLI/VPI applications" (UM-104)). For example, if you have a veriuser.c file and a library archive libapp.a file that contains the application’s object files, then the following commands should be used to create a dynamically loadable object for the Solaris operating system:

```
% cc -c -I<install_dir>/modeltech/include veriuser.c
% ld -G -o app.sl veriuser.o libapp.a
```

That’s all there is to it. The PLI application is ready to be run with ModelSim Verilog. All that’s left is to specify the resulting object file to the simulator for loading using the Veriuser modesim.ini file entry, the -pli simulator option, or the PLIOBJS environment variable (see "Registering PLI applications" (UM-101)).

**Note:** On the HP700 platform, the object files must be compiled as position-independent code by using the +z compiler option. Since, the object files supplied for Verilog-XL may be compiled for static linking, you may not be able to use the object files to create a dynamically loadable object for ModelSim Verilog. In this case, you must get the third party application vendor to supply the object files compiled as position-independent code.
**Support for VHDL objects**

The PLI ACC routines also provide limited support for VHDL objects in either an all VHDL design or a mixed VHDL/Verilog design. The following table lists the VHDL objects for which handles may be obtained and their type and fulltype constants:

<table>
<thead>
<tr>
<th>Type</th>
<th>Fulltype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accArchitecture</td>
<td>accArchitecture</td>
<td>instantiation of an architecture</td>
</tr>
<tr>
<td>accArchitecture</td>
<td>accEntityVitalLevel0</td>
<td>instantiation of an architecture whose entity is marked with the attribute VITAL_Level0</td>
</tr>
<tr>
<td>accArchitecture</td>
<td>accArchVitalLevel0</td>
<td>instantiation of an architecture which is marked with the attribute VITAL_Level0</td>
</tr>
<tr>
<td>accArchitecture</td>
<td>accArchVitalLevel1</td>
<td>instantiation of an architecture which is marked with the attribute VITAL_Level1</td>
</tr>
<tr>
<td>accArchitecture</td>
<td>accForeignArch</td>
<td>instantiation of an architecture which is marked with the attribute FOREIGN and which does not contain any VHDL statements or objects other than ports and generics</td>
</tr>
<tr>
<td>accArchitecture</td>
<td>accForeignArchMixed</td>
<td>instantiation of an architecture which is marked with the attribute FOREIGN and which contains some VHDL statements or objects besides ports and generics</td>
</tr>
<tr>
<td>accBlock</td>
<td>accBlock</td>
<td>block statement</td>
</tr>
<tr>
<td>accForLoop</td>
<td>accForLoop</td>
<td>for loop statement</td>
</tr>
<tr>
<td>accForeign</td>
<td>accShadow</td>
<td>foreign scope created by mti_CreateRegion()</td>
</tr>
<tr>
<td>accGenerate</td>
<td>accGenerate</td>
<td>generate statement</td>
</tr>
<tr>
<td>accPackage</td>
<td>accPackage</td>
<td>package declaration</td>
</tr>
<tr>
<td>accSignal</td>
<td>accSignal</td>
<td>signal declaration</td>
</tr>
</tbody>
</table>

The type and fulltype constants for VHDL objects are defined in the `acc_vhdl.h` include file. All of these objects (except signals) are scope objects that define levels of hierarchy in the Structure window. Currently, the PLI ACC interface has no provision for obtaining handles to generics, types, constants, variables, attributes, subprograms, and processes. However, some of these objects can be manipulated through the ModelSim VHDL foreign interface (mti_* routines). See the FLI Reference Manual for more information.
IEEE Std 1364 ACC routines

ModelSim Verilog supports the following ACC routines, described in detail in the IEEE Std 1364.

<table>
<thead>
<tr>
<th>acc_append_delays</th>
<th>acc_append_pulser</th>
<th>acc_close</th>
</tr>
</thead>
<tbody>
<tr>
<td>acc_collect</td>
<td>acc_compare_handles</td>
<td>acc_configure</td>
</tr>
<tr>
<td>acc_count</td>
<td>acc_fetch_argc</td>
<td>acc_fetch_argv</td>
</tr>
<tr>
<td>acc_fetch_attribute</td>
<td>acc_fetch_attribute_int</td>
<td>acc_fetch_attribute_str</td>
</tr>
<tr>
<td>acc_fetch_defname</td>
<td>acc_fetch_delay_mode</td>
<td>acc_fetch_delays</td>
</tr>
<tr>
<td>acc_fetch_direction</td>
<td>acc_fetch_edge</td>
<td>acc_fetch_fullname</td>
</tr>
<tr>
<td>acc_fetch_fulltype</td>
<td>acc_fetch_index</td>
<td>acc_fetch_location</td>
</tr>
<tr>
<td>acc_fetch_name</td>
<td>acc_fetch_paramtype</td>
<td>acc_fetch_paramval</td>
</tr>
<tr>
<td>acc_fetch_polarity</td>
<td>acc_fetch_precision</td>
<td>acc_fetch_pulser</td>
</tr>
<tr>
<td>acc_fetch_range</td>
<td>acc_fetch_size</td>
<td>acc_fetch_tfarg</td>
</tr>
<tr>
<td>acc_fetch_itfarg</td>
<td>acc_fetch_tfarg_int</td>
<td>acc_fetch_tfarg_int</td>
</tr>
<tr>
<td>acc_fetch_tfarg_str</td>
<td>acc_fetch_tfarg_str</td>
<td>acc_fetch_timescale_info</td>
</tr>
<tr>
<td>acc_fetch_type</td>
<td>acc_fetch_type_str</td>
<td>acc_fetch_value</td>
</tr>
<tr>
<td>acc_free</td>
<td>acc_handle_by_name</td>
<td>acc_handle_calling_mod_m</td>
</tr>
<tr>
<td>acc_handle_condition</td>
<td>acc_handle_conn</td>
<td>acc_handle_hicomm</td>
</tr>
<tr>
<td>acc_handle_interactive_scope</td>
<td>acc_handle_loconn</td>
<td>acc_handle_modpath</td>
</tr>
<tr>
<td>acc_handle_notifier</td>
<td>acc_handle_object</td>
<td>acc_handle_parent</td>
</tr>
<tr>
<td>acc_handle_path</td>
<td>acc_handle_pathin</td>
<td>acc_handle_pathout</td>
</tr>
<tr>
<td>acc_handle_port</td>
<td>acc_handle_scope</td>
<td>acc_handle_simulated_net</td>
</tr>
<tr>
<td>acc_handle_tchk</td>
<td>acc_handle_tchkarg1</td>
<td>acc_handle_tchkarg2</td>
</tr>
<tr>
<td>acc_handle_terminal</td>
<td>acc_handle_tfarg</td>
<td>acc_handle_itfarg</td>
</tr>
<tr>
<td>acc_handle_tfinst</td>
<td>acc_initialize</td>
<td>acc_next</td>
</tr>
<tr>
<td>acc_next_bit</td>
<td>acc_next_cell</td>
<td>acc_next_cell_load</td>
</tr>
<tr>
<td>acc_next_child</td>
<td>acc_next_driver</td>
<td>acc_next_hicomm</td>
</tr>
<tr>
<td>acc_next_input</td>
<td>acc_next_load</td>
<td>acc_next_loconn</td>
</tr>
<tr>
<td>acc_next_modpath</td>
<td>acc_next_net</td>
<td>acc_next_output</td>
</tr>
<tr>
<td>acc_next_parameter</td>
<td>acc_next_port</td>
<td>acc_next_portout</td>
</tr>
</tbody>
</table>
IEEE Std 1364 TF routines

ModelSim Verilog supports the following TF routines, described in detail in the IEEE Std 1364.

<table>
<thead>
<tr>
<th>acc_next_primitive</th>
<th>acc_next_scope</th>
<th>acc_next_specparam</th>
</tr>
</thead>
<tbody>
<tr>
<td>acc_next_tchk</td>
<td>acc_next_terminal</td>
<td>acc_next_topmod</td>
</tr>
<tr>
<td>acc_object_in_typelist</td>
<td>acc_object_of_type</td>
<td>acc_product_type</td>
</tr>
<tr>
<td>acc_product_version</td>
<td>acc_release_object</td>
<td>acc_replace_delays</td>
</tr>
<tr>
<td>acc_replace_pulsere</td>
<td>acc_reset_buffer</td>
<td>acc_set_interactive_scope</td>
</tr>
<tr>
<td>acc_set_pulsere</td>
<td>acc_set_scope</td>
<td>acc_set_value</td>
</tr>
<tr>
<td>acc_vcl_add</td>
<td>acc_vcl_delete</td>
<td>acc_version</td>
</tr>
</tbody>
</table>

**Note:** acc_fetch_paramval() cannot be used on 64-bit platforms to fetch a string value of a parameter. Because of this, the function acc_fetch_paramval_str() has been added to the PLI for this use. acc_fetch_paramval_str() is declared in acc_user.h. It functions in a manner similar to acc_fetch_paramval() except that it returns a char *. acc_fetch_paramval_str() can be used on all platforms.
<table>
<thead>
<tr>
<th>tf_long_to_real</th>
<th>tf_longtime_tostr</th>
<th>tf_message</th>
</tr>
</thead>
<tbody>
<tr>
<td>tf_mipname</td>
<td>tf_imipname</td>
<td>tf_movepvc_flag</td>
</tr>
<tr>
<td>tf_imovepvc_flag</td>
<td>tf_multiply_long</td>
<td>tf_nodeinfo</td>
</tr>
<tr>
<td>tf_inodeinfo</td>
<td>tf_nump</td>
<td>tf_inump</td>
</tr>
<tr>
<td>tf_propagatep</td>
<td>tf_ipropagatep</td>
<td>tf_putlongp</td>
</tr>
<tr>
<td>tf_iputlongp</td>
<td>tf_putp</td>
<td>tf_iputp</td>
</tr>
<tr>
<td>tf_putrealp</td>
<td>tf_iputrealp</td>
<td>tf_read_restart</td>
</tr>
<tr>
<td>tf_real_to_long</td>
<td>tf_rosynchronize</td>
<td>tf_irosynchronize</td>
</tr>
<tr>
<td>tf_scale_longdelay</td>
<td>tf_scale_realdelay</td>
<td>tf_setdelay</td>
</tr>
<tr>
<td>tf_isetdelay</td>
<td>tf_setlongdelay</td>
<td>tf_isetlongdelay</td>
</tr>
<tr>
<td>tf_setrealdelay</td>
<td>tf_isetrealdelay</td>
<td>tf_setworkarea</td>
</tr>
<tr>
<td>tf_isetworkarea</td>
<td>tf_sizep</td>
<td>tf_isizep</td>
</tr>
<tr>
<td>tf_spname</td>
<td>tf_ispname</td>
<td>tf_strdelputp</td>
</tr>
<tr>
<td>tf_istrdelputp</td>
<td>tf_strgetp</td>
<td>tf_istrgetp</td>
</tr>
<tr>
<td>tf_strgettime</td>
<td>tf_strlongdelputp</td>
<td>tf_istrlongdelputp</td>
</tr>
<tr>
<td>tf_strrealdelputp</td>
<td>tf_istrrealdelputp</td>
<td>tf_subtract_long</td>
</tr>
<tr>
<td>tf_synchronize</td>
<td>tf_isynchronize</td>
<td>tf_testpvc_flag</td>
</tr>
<tr>
<td>tf_istestpvc_flag</td>
<td>tf_text</td>
<td>tf_typep</td>
</tr>
<tr>
<td>tf_itypep</td>
<td>tf_unscale_longdelay</td>
<td>tf_unscale_realdelay</td>
</tr>
<tr>
<td>tf_warning</td>
<td>tf_write_save</td>
<td></td>
</tr>
</tbody>
</table>
Verilog-XL compatible routines

The following PLI routines are not defined in IEEE Std 1364, but ModelSim Verilog provides them for compatibility with Verilog-XL.

```c
char *acc_decompile_exp(handle condition)
```

This routine provides similar functionality to the Verilog-XL `acc_decompile_expr` routine. The condition argument must be a handle obtained from the `acc_handle_condition` routine. The value returned by `acc_decompile_exp` is the string representation of the condition expression.

```c
char *tf_dumpfilename(void)
```

This routine returns the name of the VCD file.

```c
void tf_dumpflush(void)
```

A call to this routine flushes the VCD file buffer (same effect as calling `$dumpflush` in the Verilog code).

```c
int tf_getlongsimtime(int *aof_hightime)
```

This routine gets the current simulation time as a 64-bit integer. The low-order bits are returned by the routine, while the high-order bits are stored in the `aof_hightime` argument.

64-bit support in the PLI

The PLI function `acc_fetch_paramval()` cannot be used on 64-bit platforms to fetch a string value of a parameter. Because of this, the function `acc_fetch_paramval_str()` has been added to the PLI for this use. `acc_fetch_paramval_str()` is declared in `acc_user.h`. It functions in a manner similar to `acc_fetch_paramval()` except that it returns a `char *`. `acc_fetch_paramval_str()` can be used on all platforms.

PLI/VPI tracing

The foreign interface tracing feature is available for tracing PLI and VPI function calls. Foreign interface tracing creates two kinds of traces: a human-readable log of what functions were called, the value of the arguments, and the results returned; and a set of C-language files that can be used to replay what the foreign interface code did.

The purpose of tracing files

The purpose of the log file is to aid you in debugging PLI or VPI code. The primary purpose of the replay facility is to send the replay file to MTI support for debugging co-simulation problems, or debugging PLI/VPI problems for which it is impractical to send the PLI/VPI code. We still need you to send the VHDL/Verilog part of the design to actually execute a replay, but many problems can be resolved with the trace only.

Invoking a trace

To invoke the trace, call `vsim` (CR-284) with the `-trace_foreign` option:

```
vsim
 -trace_foreign <action> [-tag <name>]
```
Arguments

<action>
Specifies one of the following actions:

<table>
<thead>
<tr>
<th>Value</th>
<th>Action</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>create log only</td>
<td>writes a local file called &quot;mti_trace_&lt;tag&gt;&quot;</td>
</tr>
<tr>
<td>2</td>
<td>create replay only</td>
<td>writes local files called &quot;mti_data_&lt;tag&gt;.c&quot;, &quot;mti_init_&lt;tag&gt;.c&quot;, &quot;mti_replay_&lt;tag&gt;.c&quot; and &quot;mti_top_&lt;tag&gt;.c&quot;</td>
</tr>
<tr>
<td>3</td>
<td>create both log and replay</td>
<td></td>
</tr>
</tbody>
</table>

-tag <name>
Used to give distinct file names for multiple traces. Optional.

Examples

vsim -trace_foreign 1 mydesign
Creates a logfile.

vsim -trace_foreign 3 mydesign
Creates both a logfile and a set of replay files.

vsim -trace_foreign 1 -tag 2 mydesign
Creates a logfile with a tag of "2".

The tracing operations will provide tracing during all user foreign code-calls, including PLI/VPI user tasks and functions (calltf, checktf, sizetf and misctf routines), and Verilog VCL callbacks.
ModelSim single-kernel simulation (SKS) allows you to simulate designs that are written in VHDL and/or Verilog. This chapter outlines data mapping and the criteria established to instantiate design units between HDLs.

The boundaries between VHDL and Verilog are enforced at the level of a design unit. This means that although a design unit must be either all VHDL or all Verilog, it may instantiate design units from either language. Any instance in the design hierarchy may be a design unit from either HDL without restriction. SKS technology allows the top-level design unit to be either VHDL or Verilog. As you traverse the design hierarchy, instantiations may freely switch back and forth between VHDL and Verilog.
Separate compilers, common libraries

VHDL source code is compiled by `vcom` (CR-240) and the resulting compiled design units (entities, architectures, configurations, and packages) are stored in a library. Likewise, Verilog source code is compiled by `vlog` (CR-274) and the resulting design units (modules and UDPs) are stored in a library.

Libraries can store any combination of VHDL and Verilog design units, provided the design unit names do not overlap (VHDL design unit names are changed to lower case). See "Design libraries" (UM-37) for more information about library management and see the `vcom` (CR-240) and the `vlog` commands.

Mapping data types

Cross-HDL instantiation does not require any extra effort on your part. As ModelSim loads a design it detects cross-HDL instantiations – made possible because a design unit's HDL type can be determined as it is loaded from a library – and the necessary adaptations and data type conversions are performed automatically.

A VHDL instantiation of Verilog may associate VHDL signals and values with Verilog ports and parameters. Likewise, a Verilog instantiation of VHDL may associate Verilog nets and values with VHDL ports and generics. ModelSim automatically maps between the HDL data types as shown below.

VHDL generics

<table>
<thead>
<tr>
<th>VHDL type</th>
<th>Verilog type</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>integer or real</td>
</tr>
<tr>
<td>real</td>
<td>integer or real</td>
</tr>
<tr>
<td>time</td>
<td>integer or real</td>
</tr>
<tr>
<td>physical</td>
<td>integer or real</td>
</tr>
<tr>
<td>enumeration</td>
<td>integer or real</td>
</tr>
<tr>
<td>string</td>
<td>string literal</td>
</tr>
</tbody>
</table>

When a scalar type receives a real value, the real is converted to an integer by truncating the decimal portion.

Type time is treated specially: the Verilog number is converted to a time value according to the `timescale` directive of the module.

Physical and enumeration types receive a value that corresponds to the position number indicated by the Verilog number. In VHDL this is equivalent to `T'VAL(P)`, where T is the type, VAL is the predefined function attribute that returns a value given a position number, and P is the position number.
Verilog parameters

<table>
<thead>
<tr>
<th>VHDL type</th>
<th>Verilog type</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>string</td>
<td>string</td>
</tr>
</tbody>
</table>

The type of a Verilog parameter is determined by its initial value.

VHDL and Verilog ports

The allowed VHDL types for ports connected to Verilog nets and for signals connected to Verilog ports are:

<table>
<thead>
<tr>
<th>Allowed VHDL types</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit</td>
</tr>
<tr>
<td>bit_vector</td>
</tr>
<tr>
<td>std_logic</td>
</tr>
<tr>
<td>std_logic_vector</td>
</tr>
<tr>
<td>vl_logic</td>
</tr>
<tr>
<td>vl_logic_vector</td>
</tr>
</tbody>
</table>

The vl_logic type is an enumeration that defines the full state set for Verilog nets, including ambiguous strengths. The bit and std_logic types are convenient for most applications, but the vl_logic type is provided in case you need access to the full Verilog state set. For example, you may wish to convert between vl_logic and your own user-defined type. The vl_logic type is defined in the vl_types package in the pre-compiled verilog library. This library is provided in the installation directory along with the other pre-compiled libraries (std and ieee). The source code for the vl_types package can be found in the files installed with ModelSim. (See \modeltech\vhdl_src\verilog\vl_types.vhd.)
Verilog states

Verilog states are mapped to std_logic and bit as follows:

<table>
<thead>
<tr>
<th>Verilog</th>
<th>std_logic</th>
<th>bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>HiZ</td>
<td>'Z'</td>
<td>'0'</td>
</tr>
<tr>
<td>Sm0</td>
<td>'L'</td>
<td>'0'</td>
</tr>
<tr>
<td>Sm1</td>
<td>'H'</td>
<td>'1'</td>
</tr>
<tr>
<td>SmX</td>
<td>'W'</td>
<td>'0'</td>
</tr>
<tr>
<td>Me0</td>
<td>'L'</td>
<td>'0'</td>
</tr>
<tr>
<td>Me1</td>
<td>'H'</td>
<td>'1'</td>
</tr>
<tr>
<td>MeX</td>
<td>'W'</td>
<td>'0'</td>
</tr>
<tr>
<td>We0</td>
<td>'L'</td>
<td>'0'</td>
</tr>
<tr>
<td>We1</td>
<td>'H'</td>
<td>'1'</td>
</tr>
<tr>
<td>WeX</td>
<td>'W'</td>
<td>'0'</td>
</tr>
<tr>
<td>La0</td>
<td>'L'</td>
<td>'0'</td>
</tr>
<tr>
<td>La1</td>
<td>'H'</td>
<td>'1'</td>
</tr>
<tr>
<td>LaX</td>
<td>'W'</td>
<td>'0'</td>
</tr>
<tr>
<td>Pu0</td>
<td>'L'</td>
<td>'0'</td>
</tr>
<tr>
<td>Pu1</td>
<td>'H'</td>
<td>'1'</td>
</tr>
<tr>
<td>PuX</td>
<td>'W'</td>
<td>'0'</td>
</tr>
<tr>
<td>St0</td>
<td>'0'</td>
<td>'0'</td>
</tr>
<tr>
<td>St1</td>
<td>'1'</td>
<td>'1'</td>
</tr>
<tr>
<td>StX</td>
<td>'X'</td>
<td>'0'</td>
</tr>
<tr>
<td>Su0</td>
<td>'0'</td>
<td>'0'</td>
</tr>
<tr>
<td>Su1</td>
<td>'1'</td>
<td>'1'</td>
</tr>
<tr>
<td>SuX</td>
<td>'X'</td>
<td>'0'</td>
</tr>
</tbody>
</table>

For Verilog states with ambiguous strength:

- bit receives '0'
- std_logic receives 'X' if either the 0 or 1 strength component is greater than or equal to strong strength
- std_logic receives 'W' if both the 0 and 1 strength components are less than strong strength
VHDL type bit is mapped to Verilog states as follows:

<table>
<thead>
<tr>
<th>bit</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>'0'</td>
<td>St0</td>
</tr>
<tr>
<td>'1'</td>
<td>St1</td>
</tr>
</tbody>
</table>

VHDL type std_logic is mapped to Verilog states as follows:

<table>
<thead>
<tr>
<th>std_logic</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>'U'</td>
<td>StX</td>
</tr>
<tr>
<td>'X'</td>
<td>StX</td>
</tr>
<tr>
<td>'0'</td>
<td>St0</td>
</tr>
<tr>
<td>'1'</td>
<td>St1</td>
</tr>
<tr>
<td>'Z'</td>
<td>HiZ</td>
</tr>
<tr>
<td>'W'</td>
<td>PuX</td>
</tr>
<tr>
<td>'L'</td>
<td>Pu0</td>
</tr>
<tr>
<td>'H'</td>
<td>Pu1</td>
</tr>
<tr>
<td>'-'</td>
<td>StX</td>
</tr>
</tbody>
</table>
**VHDL instantiation of Verilog design units**

Once you have generated a component declaration for a Verilog module, you can instantiate the component just like any other VHDL component. In addition, you can reference a Verilog module in the entity aspect of a component configuration – all you need to do is specify a module name instead of an entity name. You can also specify an optional architecture name, but it will be ignored because Verilog modules do not have architectures.

**Verilog instantiation criteria**

A Verilog design unit may be instantiated from VHDL if it meets the following criteria:

- The design unit is a module (UDPs are not allowed).
- The ports are named ports (Verilog allows unnamed ports).
- The ports are not connected to bidirectional pass switches (it is not possible to handle pass switches in VHDL).

**Component declaration**

A Verilog module that is compiled into a library can be referenced from a VHDL design as though the module is a VHDL entity. The interface to the module can be extracted from the library in the form of a component declaration by running `vgencomp` (CR-248). Given a library and module name, `vgencomp` (CR-248) writes a component declaration to standard output.

The default component port types are:

- `std_logic`
- `std_logic_vector`

Optionally, you can choose:

- `bit` and `bit_vector`
- `vl_logic` and `vl_logic_vector`

**VHDL and Verilog identifiers**

The VHDL identifiers for the component name, port names, and generic names are the same as the Verilog identifiers for the module name, port names, and parameter names. If a Verilog identifier is not a valid VHDL 1076-1987 identifier, it is converted to a VHDL 1076-1993 extended identifier (in which case you must compile the VHDL with the `-93` switch). Any uppercase letters in Verilog identifiers are converted to lowercase in the VHDL identifier, except in the following cases:

- The Verilog module was compiled with the `-93` switch. This means `vgencomp` (CR-248) should use VHDL 1076-1993 extended identifiers in the component declaration to preserve case in the Verilog identifiers that contain uppercase letters.

- The Verilog module, port, or parameter names are not unique unless case is preserved. In this event, `vgencomp` (CR-248) behaves as if the module was compiled with the `-93` switch for those names only.
**Note:** If you use Verilog identifiers where the names are unique by case only, use the `-93` switch when compiling mixed-language designs.

## Examples

<table>
<thead>
<tr>
<th>Verilog identifier</th>
<th>VHDL identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>topmod</td>
<td>topmod</td>
</tr>
<tr>
<td>TOPMOD</td>
<td>topmod</td>
</tr>
<tr>
<td>TopMod</td>
<td>topmod</td>
</tr>
<tr>
<td>top_mod</td>
<td>top_mod</td>
</tr>
<tr>
<td>_topmod</td>
<td>_topmod</td>
</tr>
<tr>
<td>\topmod\</td>
<td>\topmod\</td>
</tr>
</tbody>
</table>

If the Verilog module is compiled with `-93`:

<table>
<thead>
<tr>
<th>Verilog identifier</th>
<th>VHDL identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>topmod</td>
<td>topmod</td>
</tr>
<tr>
<td>TOPMOD</td>
<td>TOPMOD\</td>
</tr>
<tr>
<td>TopMod</td>
<td>TopMod\</td>
</tr>
<tr>
<td>top_mod</td>
<td>top_mod</td>
</tr>
<tr>
<td>_topmod</td>
<td>_topmod\</td>
</tr>
<tr>
<td>\topmod\</td>
<td>\topmod\</td>
</tr>
</tbody>
</table>
vgencomp component declaration

vgencomp (CR-248) generates a component declaration according to these rules:

**Generic clause**
A generic clause is generated if the module has parameters. A corresponding generic is defined for each parameter that has an initial value that does not depend on any other parameters.

The generic type is determined by the parameter's initial value as follows:

<table>
<thead>
<tr>
<th>Parameter value</th>
<th>Generic type</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>string literal</td>
<td>string</td>
</tr>
</tbody>
</table>

The default value of the generic is the same as the parameter's initial value.

Examples

<table>
<thead>
<tr>
<th>Verilog parameter</th>
<th>VHDL generic</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter p1 = 1 - 3;</td>
<td>p1 : integer := -2;</td>
</tr>
<tr>
<td>parameter p2 = 3.0;</td>
<td>p2 : real := 3.000000;</td>
</tr>
<tr>
<td>parameter p3 = &quot;Hello&quot;;</td>
<td>p3 : string := &quot;Hello&quot;;</td>
</tr>
</tbody>
</table>

**Port clause**
A port clause is generated if the module has ports. A corresponding VHDL port is defined for each named Verilog port.

You can set the VHDL port type to bit, std_logic, or vl_logic. If the Verilog port has a range, then the VHDL port type is bit_vector, std_logic_vector, or vl_logic_vector. If the range does not depend on parameters, then the vector type will be constrained accordingly, otherwise it will be unconstrained.

Examples

<table>
<thead>
<tr>
<th>Verilog port</th>
<th>VHDL port</th>
</tr>
</thead>
<tbody>
<tr>
<td>input p1;</td>
<td>p1 : in std_logic;</td>
</tr>
<tr>
<td>output [7:0] p2;</td>
<td>p2 : out std_logic_vector(7 downto 0);</td>
</tr>
<tr>
<td>output [4:7] p3;</td>
<td>p3 : out std_logic_vector(4 to 7);</td>
</tr>
<tr>
<td>inout [width-1:0] p4;</td>
<td>p4 : inout std_logic_vector;</td>
</tr>
</tbody>
</table>
Verilog instantiation of VHDL design units

Configuration declarations are allowed to reference Verilog modules in the entity aspects of component configurations. However, the configuration declaration cannot extend into a Verilog instance to configure the instantiations within the Verilog module.

Verilog instantiation of VHDL design units

You can reference a VHDL entity or configuration from Verilog as though the design unit is a module of the same name (in lower case).

VHDL instantiation criteria

A VHDL design unit may be instantiated from Verilog if it meets the following criteria:

- The design unit is an entity/architecture pair or a configuration declaration.
- The entity ports are of type bit, bit_vector, std_ulogic, std_ulogic_vector, vl_ulogic, vl_ulogic_vector, or their subtypes. The port clause may have any mix of these types.
- The generics are of type integer, real, time, physical, enumeration, or string. String is the only composite type allowed.

Port associations may be named or positional. Use the same port names and port positions that appear in the entity.

Named port associations

Named port associations are not case sensitive – unless a VHDL port name is an extended identifier (1076-1993). If the VHDL port name is an extended identifier, the association is case sensitive and the VHDL identifier’s leading and trailing backslashes are removed before comparison.

Generic associations are provided via the module instance parameter value list. List the values in the same order that the generics appear in the entity. The deparam statement is not allowed for setting generic values.

An entity name is not case sensitive in Verilog instantiations. The entity default architecture is selected from the work library unless specified otherwise.

Verilog does not have the concept of architectures or libraries, so the escaped identifier is employed to provide an extended form of instantiation:

```
\mylib.entity(arch) u1 (a, b, c);
\mylib.entity u1 (a, b, c);
\entity(arch) u1 (a, b, c);
```

If the escaped identifier takes the form of one of the above and is not the name of a design unit in the work library, then the instantiation is broken down as follows:

- library = mylib
- design unit = entity
- architecture = arch

SDF annotation

A mixed VHDL/Verilog design can also be annotated with SDF. See "SDF for Mixed VHDL and Verilog Designs" (UM-330) for more information.
Chapter contents

WLF files (datasets) .................................................. UM-132
  Saving a simulation to a WLF file .................................. UM-132
  Opening datasets .................................................. UM-133
  Viewing dataset structure ......................................... UM-134
  Managing datasets ................................................ UM-136
  Using datasets with ModelSim commands .......................... UM-136
  Restricting the dataset prefix display ............................ UM-137

Virtual Objects (User-defined buses, and more) ........................ UM-138
  Virtual signals .................................................. UM-138
  Virtual functions ............................................... UM-139
  Virtual regions .................................................. UM-140
  Virtual types .................................................... UM-140

Dataset, WLF file, and virtual commands ............................... UM-141

A ModelSim simulation can be saved to a wave log format (WLF) file (using the -wlf <filename> argument to the vsim command (CR-284)) for future viewing or comparison to a current simulation. We use the term "dataset" to refer to a WLF file that has been reopened for viewing.

With ModelSim release 5.3 and later, you can open more than one WLF file for simultaneous viewing. You can also create virtual signals that are simple logical combinations of, or logical functions of, signals from different datasets.
WLF files (datasets)

Wave log format (WLF) files store saved simulation data. Any number of WLF files can be reloaded for viewing or comparing to the active simulation. The term "dataset" refers to a logical name that is assigned to the WLF file when it is reloaded.

A dataset prefix identifies each WLF file that is opened. The current active simulation is prefixed by "sim," while any datasets are prefixed by the name of the WLF file. For example, two datasets are displayed in the Wave window below—the current simulation is shown in the top pane and is indicated by the "sim" prefix; a dataset from a previous simulation is shown in the bottom pane and is indicated by the "gold" prefix.

![Wave window](image)

- **Note:** The simulator time resolution (see Resolution (UM-390)) must be the same for all datasets you’re comparing, including the current simulation.

Saving a simulation to a WLF file

The results of each simulation run are automatically saved to a WLF file called `vsim.wlf` in the current directory. If you run a new simulation in the same directory, the `vsim.wlf` file is overwritten with the new results. Therefore, you should use the `-wlf <filename>` argument to the `vsim` command (CR-284) to specify a different name if you want to save the WLF file.

- **Important:** You must end a simulation session with a quit or quit -sim command in order to produce a valid WLF file. If you don’t end the simulation in this manner, the WLF file will not close properly, and ModelSim will issue an error when you try to open the dataset in subsequent sessions.
Opening datasets

To open a dataset, select either File > Open > Dataset (Main window) or File > Open Dataset (Wave window).

The Open Dataset dialog box includes the following options.

- **Dataset Pathname**
  Identifies the path and filename of the WLF file you want to open.

- **Logical Name for Dataset**
  This is the name by which the dataset will be referred. By default this is the name of the WLF file.

▲ **Important:** You must end a simulation session with a quit or quit -sim command in order to produce a valid WLF file. If you don’t end the simulation in this manner, the WLF file will not close properly, and ModelSim will issue an error when you try to open the dataset in subsequent sessions.
### Viewing dataset structure

In versions 5.5 and later, each dataset you open creates a Structure tab in the Main window workspace. The tab is labeled with the name of the dataset and displays the same data as the "Structure window" (UM-204).

The graphic below shows three Structure tabs: one for the active simulation ("Sim") and one each for two open datasets ("Test" and "Gold").

If you have too many tabs to display in the available space, you can scroll the tabs left or right by clicking and dragging them.

Each Structure tab has a context menu that you access by clicking the right mouse button (Windows—2nd button, UNIX—3rd button) anywhere within the Structure tab.
The Structure tab context menu includes the following options.

- **Save As**
  Writes the HDL item names in the Structure tab to a text file.

- **Sort**
  Sorts the HDL items in the Structure tab by alphabetic (ascending or descending) or declaration order.

- **Expand Selected**
  Shows the hierarchy of the selected HDL item.

- **Collapse Selected**
  Hides the hierarchy of the selected HDL item.

- **Expand All**
  Shows the hierarchy of all HDL items in the list.

- **Collapse All**
  Hides the hierarchy of all HDL items in the list.

- **Find**
  Opens the Find dialog. See "Finding items in the Structure window" (UM-206) for details.
Managing datasets

When you have one or more datasets open, you can manage them using the Dataset Browser. To open the browser, select View > Datasets (Main window).

The Dataset Browser dialog box includes the following options.

- **Open Dataset**
  Opens the View Dataset dialog box (see "Opening datasets" (UM-133)) so you can open additional datasets.

- **Close Dataset**
  Closes the selected dataset. This will also remove the dataset’s Structure tab in the Main window workspace.

- **Make Active**
  Makes the selected dataset "active." You can also effect this change by double-clicking the dataset name. Active dataset means that if you type a region path as part of a command and omit the dataset prefix, the active dataset will be assumed. It is equivalent to typing: env <dataset>: at the VSIM prompt.

- **Rename Dataset**
  Allows you to assign a new logical name for the selected dataset.

Using datasets with ModelSim commands

Multiple datasets can be opened when the simulator is invoked by specifying more than one `vsim -view <filename>` option. By default the dataset prefix will be the filename of the WLF file. A different dataset name can also be specified as an optional qualifier to the `vsim -view` switch on the command line using the following syntax:

```
    -view <dataset>=<filename>
```

For example: `vsim -view foo=vsim.wlf`
Design regions and signal names can be fully specified over multiple WLF files by using the dataset name as a prefix in the path. For example:

```
sim:/top/alu/out
view:/top/alu/out
golden:.top.alu.out
```

Dataset prefixes are not required unless more than one dataset is open, and you want to refer to something outside the default dataset. When more than one dataset is open, ModelSim will automatically prefix names in the Wave and List window with the dataset name. You can change this default by selecting Edit > Display Properties (Wave window) and Prop > Display Props (List window).

ModelSim designates one of the datasets to be the "active" dataset, and refers all names without dataset prefixes to that dataset. The active dataset is displayed in the context path at the bottom of the Main window. When you select a design unit in a dataset’s Structure tab, that dataset becomes active automatically. Alternatively, you can use the Dataset Browser or the environment command (CR-137) to change the active dataset.

ModelSim remembers a "current context" within each open dataset. You can toggle between the current context of each dataset using the environment command (CR-137), specifying the dataset without a path. For example:

```
env foo:
```

sets the active dataset to `foo` and the current context to the context last specified for `foo`. The context is then applied to any unlocked windows.

The current context of the current dataset (usually referred to as just "current context") is used for finding objects specified without a path.

The Signals window can be locked to a specific context of a dataset. Being locked to a dataset means that the window will update only when the content of that dataset changes. If locked to both a dataset and a context (e.g., test: /top/foo), the window will update only when that specific context changes. You specify the dataset to which the window is locked by selecting File > Environment (Signals window).

**Restricting the dataset prefix display**

The default for dataset prefix viewing is set with a variable in `pref.tcl`, `PrefMain(DisplayDatasetPrefix)`. Setting the variable to 1 will display the prefix, setting it to 0 will not. It is set to 1 by default. Either edit the `pref.tcl` file directly or use the Options > Edit Preferences (Main window) command to change the variable value.

Additionally, you can restrict display of the dataset prefix if you use the `environment -nodataset` command to view a dataset. To display the prefix use the `environment` command (CR-137) with the `-dataset` option (you won’t need to specify this option if the variable noted above is set to 1). The `environment` command line switches override the `pref.tcl` variable.
Virtual Objects (User-defined buses, and more)

Virtual objects are signal-like or region-like objects created in the GUI that do not exist in the ModelSim simulation kernel. Beginning with release 5.3, ModelSim supports the following kinds of virtual objects:

- **Virtual signals** (UM-138)
- **Virtual functions** (UM-139)
- **Virtual regions** (UM-140)
- **Virtual types** (UM-140)

Virtual objects are indicated by an orange diamond as illustrated by BUS1 below:

![Wave window with virtual signals](image)

**Virtual signals**

Virtual signals are aliases for combinations or subelements of signals written to the WLF file by the simulation kernel. They can be displayed in the Signals, List, and Wave windows, accessed by the `examine` command, and set using the `force` command. Virtual signals can be created via a menu in the Wave and List windows (Edit > Combine), or with the `virtual signal` command (CR-269). Virtual signals can also be dragged and dropped from the Signals window to the Wave and List windows.

Virtual signals are automatically attached to the design region in the hierarchy that corresponds to the nearest common ancestor of all the elements of the virtual signal. The `virtual signal` command has an `-install <region>` option to specify where the virtual signal should be installed. This can be used to install the virtual signal in a user-defined region in
Virtual Objects (User-defined buses, and more)

order to reconstruct the original RTL hierarchy when simulating and driving a post-synthesis, gate-level implementation.

A virtual signal can be used to reconstruct RTL-level design buses that were broken down during synthesis. The virtual hide command (CR-260) can be used to hide the display of the broken-down bits if you don't want them cluttering up the Signals window.

If the virtual signal has elements from more than one WLF file, it will be automatically installed in the virtual region "virtuals:/Signals."

Virtual signals are not hierarchical – if two virtual signals are concatenated to become a third virtual signal, the resulting virtual signal will be a concatenation of all the subelements of the first two virtual signals.

The definitions of virtuals can be saved to a macro file using the virtual save command (CR-267). By default, when quitting, ModelSim will append any newly-created virtuals (that have not been saved) to the virtuals.do file in the local directory.

If you have virtual signals displayed in the Wave or List window when you save the Wave or List format, you will need to execute the virtuals.do file (or some other equivalent) to restore the virtual signal definitions before you re-load the Wave or List format during a later run. There is one exception: "implicit virtuals" are automatically saved with the Wave or List format.

Implicit and explicit virtuals

An implicit virtual is a virtual signal that was automatically created by ModelSim without your knowledge and without you providing a name for it. An example would be if you expand a bus in the Wave window, then drag one bit out of the bus to display it separately. That action creates a one-bit virtual signal whose definition is stored in a special location, and is not visible in the Signals window or to the normal virtual commands.

All other virtual signals are considered "explicit virtuals".

Virtual functions

Virtual functions behave in the GUI like signals but are not aliases of combinations or elements of signals logged by the kernel. They consist of logical operations on logged signals and can be dependent on simulation time. They can be displayed in the Signals, Wave, and List windows and accessed by the examine command (CR-138), but cannot be set by the force command (CR-144).

Examples of virtual functions include the following:

- a function defined as the inverse of a given signal
- a function defined as the exclusive-OR of two signals
- a function defined as a repetitive clock
- a function defined as "the rising edge of CLK delayed by 1.34 ns"

Virtual functions can also be used to convert signal types and map signal values.

The result type of a virtual signal can be any of the types supported in the GUI expression syntax: integer, real, boolean, std_logic, std_logic_vector, and arrays and records of these types. Verilog types are converted to VHDL 9-state std_logic equivalents and Verilog net strengths are ignored.
Virtual functions can be created using the `virtual function` command (CR-257).

Virtual functions are also implicitly created by ModelSim when referencing bit-selects or part-selects of Verilog registers in the GUI, or when expanding Verilog registers in the Signals, Wave or List windows. This is necessary because referencing Verilog register elements requires an intermediate step of shifting and masking of the Verilog "vreg" data structure.

**Virtual regions**

User-defined design hierarchy regions can be defined and attached to any existing design region or to the virtuals context tree. They can be used to reconstruct the RTL hierarchy in a gate-level design and to locate virtual signals. Thus, virtual signals and virtual regions can be used in a gate-level design to allow you to use the RTL test bench.

Virtual regions are created and attached using the `virtual region` command (CR-266).

**Virtual types**

User-defined enumerated types can be defined in order to display signal bit sequences as meaningful alphanumeric names. The virtual type is then used in a type conversion expression to convert a signal to values of the new type. When the converted signal is displayed in any of the windows, the value will be displayed as the enumeration string corresponding to the value of the original signal.

Virtual types are created using the `virtual type` command (CR-272).
# Dataset, WLF file, and virtual commands

The table below provides a brief description of the actions associated with datasets, WLF files, and virtual commands. For complete details about syntax, arguments, and usage, refer to the *ModelSim Command Reference*.

<table>
<thead>
<tr>
<th>Command name</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset alias</td>
<td>(CR-115) closes the specified dataset</td>
</tr>
<tr>
<td>dataset list</td>
<td>(CR-119) lists all open datasets</td>
</tr>
<tr>
<td>dataset open</td>
<td>(CR-120) opens a dataset</td>
</tr>
<tr>
<td>dataset rename</td>
<td>(CR-121) assigns a new logical name to the specified dataset</td>
</tr>
<tr>
<td>log</td>
<td>(CR-154) creates a WLF file for the current simulation</td>
</tr>
<tr>
<td>nolog</td>
<td>(CR-162) suspends writing of data to the WLF file for the specified signals</td>
</tr>
<tr>
<td>searchlog</td>
<td>(CR-203) searches one or more of the currently open WLF files for a specified condition</td>
</tr>
<tr>
<td>virtual function</td>
<td>(CR-257) creates a new signal that consists of logical operations on existing signals and simulation time</td>
</tr>
<tr>
<td>virtual region</td>
<td>(CR-266) creates a new user-defined design hierarchy region</td>
</tr>
<tr>
<td>virtual signal</td>
<td>(CR-269) creates a new signal that consists of concatenations of signals and subelements</td>
</tr>
<tr>
<td>virtual type</td>
<td>(CR-272) creates a new enumerated type</td>
</tr>
<tr>
<td>vsim</td>
<td>(CR-284) -wlf &lt;filename&gt; creates a WLF file for the simulation which can be reopened as a dataset</td>
</tr>
</tbody>
</table>
The example graphics in this chapter illustrate ModelSim’s graphic interface within a Windows environment; however, ModelSim’s interface is designed to provide consistency across all supported platforms. Your operating system provides the basic window-management frames, while ModelSim controls all internal window features such as menus, buttons, and scroll bars.

Because ModelSim’s graphic interface is based on Tcl/Tk, you are able to customize your simulation environment. Easily-accessible preference variables and configuration commands give you control over the use and placement of windows, menus, menu options, and buttons.
Window overview

The ModelSim simulation and debugging environment consists of nine window types. Multiple windows of each type can be used during simulation (with the exception of the Main window). To make an additional window select View > New (Main window). A brief description of each window follows:

- **Main window** (UM-151)
  The initial window that appears upon startup. All subsequent ModelSim windows are opened from the Main window. This window contains the session transcript.

- **Dataflow window** (UM-166)
  Lets you trace signals and nets through your design by showing related processes.

- **List window** (UM-170)
  Shows the simulation values of selected VHDL signals and variables and Verilog nets and register variables in tabular format.

- **Process window** (UM-184)
  Displays a list of processes in the region currently selected in the Structure window.

- **Signals window** (UM-187)
  Shows the names and current values of VHDL signals, and Verilog nets and register variables in the region currently selected in the Structure window.

- **Source window** (UM-195)
  Displays the HDL source code for the design. (Your source code can remain hidden if you wish, see "Source code security and -nodebug" (UM-420).)

- **Structure window** (UM-204)
  Displays the hierarchy of structural elements such as VHDL component instances, packages, blocks, generate statements, and Verilog model instances, named blocks, tasks and functions. In versions 5.5 and later, this same information is displayed in the Main window workspace.

- **Variables window** (UM-207)
  Displays VHDL constants, generics, variables, and Verilog register variables in the current process and their current values.

- **Wave window** (UM-210)
  Displays waveforms, and current values for the VHDL signals and variables and Verilog nets and register variables you have selected. Current and past simulations can be compared side-by-side in one Wave window.
Common window features

ModelSim's graphic interface provides many features that add to its usability; features common to many of the windows are described below.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Feature applies to these windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick access toolbars (UM-146)</td>
<td>Main, Source, and Wave windows</td>
</tr>
<tr>
<td>Drag and Drop (UM-146)</td>
<td>Dataflow, List, Signals, Source, Structure, Variables, and Wave windows</td>
</tr>
<tr>
<td>Command history (UM-146)</td>
<td>Main window command line</td>
</tr>
<tr>
<td>Automatic window updating (UM-147)</td>
<td>Dataflow, Process, Signals, and Structure windows</td>
</tr>
<tr>
<td>Finding names, searching for values, and locating cursors (UM-147)</td>
<td>various windows</td>
</tr>
<tr>
<td>Sorting HDL items (UM-148)</td>
<td>Process, Signals, Source, Structure, Variables and Wave windows</td>
</tr>
<tr>
<td>Multiple window copies (UM-148)</td>
<td>all windows except the Main window</td>
</tr>
<tr>
<td>Menu tear off (UM-148)</td>
<td>all windows</td>
</tr>
<tr>
<td>Customizing menus and buttons (UM-148)</td>
<td>all windows</td>
</tr>
<tr>
<td>Combining signals into a user-defined bus (UM-149)</td>
<td>List and Wave windows</td>
</tr>
<tr>
<td>Tree window hierarchical view (UM-149)</td>
<td>Structure, Signals, Variables, and Wave windows</td>
</tr>
</tbody>
</table>

- Cut/Copy/Paste/Delete into any entry box by clicking the right mouse button in the entry box.
- Standard cut/copy/paste shortcut keystrokes – ^X/^C/^V – will work in all entry boxes.
- When the focus changes to an entry box, the contents of that box are selected (highlighted). This allows you to replace the current contents of the entry box with new contents with a simple paste command, without having to delete the old value.
- Dialog boxes will appear on top of their parent window (instead of the upper left corner of the screen).
The Main window includes context menus that are accessed by clicking the right mouse button.

The middle mouse button will allow you to paste the following into the transcript window:
- text currently selected in the transcript window,
- a current primary X-Windows selection (can be from another application), or
- contents of the clipboard.

Note: Selecting text in the transcript window makes it the current primary X-Windows selection. This way you can copy transcript window selections to other X-Windows windows (xterm, emacs, etc.).

The Edit > Paste operation in the transcript window will ONLY paste from the clipboard.

All menus highlight their accelerator keys.

Quick access toolbars

Buttons on the Main, Source, and Wave windows provide access to commonly used commands and functions. See, "The Main window toolbar" (UM-160), "The Source window toolbar" (UM-198), and "The Wave window toolbar" (UM-218).

Drag and Drop

Drag and drop of HDL items is possible between the following windows. Using the left mouse button, click and release to select an item, then click and hold to drag it.

- Drag items from these windows:
  Dataflow, List, Signals, Source, Structure, Variables, and Wave windows

- Drop items into these windows:
  Dataflow, List, and Wave windows

Note: Drag and drop works to rearrange items within the List and Wave windows as well.

Command history

Avoid entering long commands twice; use the down and up keyboard arrows to move through the command history for the current simulation.
Automatic window updating

Selecting an item in the following windows automatically updates other related ModelSim windows as indicated below:

<table>
<thead>
<tr>
<th>Select an item in this window</th>
<th>To update these windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataflow window (UM-166)</td>
<td>Process window (UM-184)</td>
</tr>
<tr>
<td>(with a process selected in the center of the window)</td>
<td>Signals window (UM-187)</td>
</tr>
<tr>
<td></td>
<td>Source window (UM-195)</td>
</tr>
<tr>
<td></td>
<td>Structure window (UM-204)</td>
</tr>
<tr>
<td></td>
<td>Variables window (UM-207)</td>
</tr>
<tr>
<td>Process window (UM-184)</td>
<td>Dataflow window (UM-166)</td>
</tr>
<tr>
<td></td>
<td>Signals window (UM-187)</td>
</tr>
<tr>
<td></td>
<td>Structure window (UM-204)</td>
</tr>
<tr>
<td></td>
<td>Variables window (UM-207)</td>
</tr>
<tr>
<td>Signals window (UM-187)</td>
<td>Dataflow window (UM-166)</td>
</tr>
<tr>
<td>Structure window (UM-204)</td>
<td>Process window (UM-184)</td>
</tr>
<tr>
<td></td>
<td>Signals window (UM-187)</td>
</tr>
<tr>
<td></td>
<td>Source window (UM-195)</td>
</tr>
</tbody>
</table>

Finding names, searching for values, and locating cursors

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

- **Search** for HDL item values with the **Edit > Search** menu selection in these windows: List, and Wave windows.

You can also:

- **Locate** time markers in the List window with the **Markers > Goto** menu selection.

- **Locate** time cursors in the Wave window with the **Cursor > Goto** menu selection.

In addition to the menu selections above, the virtual event `<<Find>>` is defined for all windows. The default binding is to `<Key-F19>` in most windows (the Find key on a Sun keyboard). You can bind `<<Find>>` to other events with the Tcl/Tk command `event add`. For example,

```
event add <<Find>> <control-Key-F>
```
**Sorting HDL items**

Use the **Edit > Sort** menu selection in the windows below to sort HDL items in ascending, descending or declaration order.

Process, Signals, Structure, Variables and Wave windows

Names such as net_1, net_10, and net_2 will sort numerically in the Signals and Wave windows.

**Multiple window copies**

Use the **View > New** menu selection from the Main window to create multiple copies of the same window type. The new window will become the default window for that type.

**Saving window layout**

You can save the current positions and sizes of ModelSim windows as a default. Follow these steps to save the layout as a default:

1. Position and size the windows the way you want them to display;

2. Select **Options > Save Preferences** (Main window) and save the `modelsim.tcl` file into the desired directory.

3. Modify the "Working Directory" of your ModelSim shortcut to point at the directory (Windows only), or set the MODELSIM_TCL environment variable to point at the directory (see "Creating environment variables in Windows" for more details).

**Context menus**

Context menus refer to menus that "pop-up" in the middle of the interface by clicking the right mouse button (Windows—2nd button, UNIX—3rd button). The commands on the menu change depending on where in the interface you click. In other words, the menus change based on the context of their use.

**Menu tear off**

All window menus can be "torn off " to create a separate menu window. To tear off, click on the menu, then select the dotted-line button at the top of the menu.

**Customizing menus and buttons**

Menus can be added, deleted, and modified in all windows. Custom buttons can also be added to window toolbars. See

- "Customizing the interface" (UM-273),
- "Customizing menus and buttons" (UM-148), and
- "The Button Adder" (UM-263) for more information.
Combining signals into a user-defined bus

You can collect items of the same type in the List window (UM-170) or Wave window (UM-210) and combine them into a bus. Use the Edit > Combine menu command in either window.

Tree window hierarchical view

ModelSim provides a hierarchical, or "tree view" of some aspects of your design in the Main window Structure tabs and the Structure, Signals, Variables, and Wave windows.

HDL items you can view

Depending on which window you are viewing, one entry is created for each of the following VHDL and Verilog HDL items within the design:

VHDL items
(indicated by a dark blue square icon)
signals, variables, component instantiations, generate statements, block statements, and packages

Verilog items
(indicated by a lighter blue circle icon)
parameters, registers, nets, module instantiations, named forks, named begins, tasks, and functions

Virtual items
(indicated by an orange diamond icon)
virtual signals, buses, and functions, see “Virtual Objects (User-defined buses, and more)” (UM-138) for more information

Viewing the hierarchy

Whenever you see a tree view, as in the Structure window displayed here, you can use the mouse to collapse or expand the hierarchy. Select the symbols as shown below to change the view of the structure.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ + ]</td>
<td>click a plus box to expand the item and view the structure</td>
</tr>
<tr>
<td>[ - ]</td>
<td>click a minus box to hide a hierarchy that has been expanded</td>
</tr>
</tbody>
</table>
**Finding items within tree windows**

You can open the Find dialog box within all windows (except the Dataflow windows) by selecting **Edit > Find** or by using `<control-s>` (Unix) or `<control-f>` (Windows).

Options within the Find dialog box allow you to search unique text-string fields within the specific window. See also,

- "Finding items by name in the List window" (UM-180),
- "Finding HDL items in the Signals window" (UM-192), and
- "Finding items by name or value in the Wave window" (UM-231).
Main window

The Main window is pictured below as it appears when ModelSim is first invoked. Note that your operating system graphic interface provides the window-management frame only; ModelSim handles all internal-window features including menus, buttons, and scroll bars.

The menu bar at the top of the window provides access to a wide variety of simulation commands and ModelSim preferences. The toolbar provides buttons for quick access to the many common commands. The status bar at the bottom of the window gives you information about the data in the active ModelSim window. The menu bar, toolbar, and status bar are described in detail below.
Workspace

The workspace is available in software versions 5.5 and later. It provides convenient access to projects, compiled design units, and simulation/dataset structures. It can be hidden or displayed by selecting the **View > Hide/Show Workspace** command.

The workspace can display four types of tabs, as shown in the graphic below.

- **Project tab**
  Shows all files that are included in the open project. See *Chapter 2 - Projects and system initialization* for details.

- **Library tab**
  Shows compiled design units in the specified library. See "Managing library contents" (UM-40) for details.

- **Structure tabs**
  Shows a hierarchical view of the active simulation and any open datasets. This is the same data that is displayed in the "Structure window" (UM-204). There is one tab for the current simulation and one tab for each open dataset. See "Viewing dataset structure" (UM-134) for details.

- **Compare tab**
  Shows comparison objects that were created by doing a waveform comparison. See *Chapter 11 - Waveform Comparison* for details.
Transcript

The transcript portion of the Main window maintains a running history of commands that are invoked and messages that occur as you work with ModelSim. When a simulation is running, the transcript displays a VSIM prompt, allowing you to enter command-line commands from within the graphic interface.

You can scroll backward and forward through the current work history by using the vertical scrollbar. You can also use arrow keys to recall previous commands, or copy and paste using the mouse within the window; see "The following mouse actions and special keystrokes can be used to edit commands in the entry region of the Main window. They can also be used in editing the file displayed in the Source window and all Notepad windows (enter the notepad command within ModelSim to open the Notepad editor)." (UM-162) for details.

Saving the Main window transcript file

Variable settings determine the filename used for saving the Main window transcript. If either PrefMain(file) in modelsim.tcl, or TranscriptFile in modelsim.ini file is set, then the transcript output is logged to the specified file. By default the TranscriptFile variable in modelsim.ini is set to transcript. If either variable is set, the transcript contents are always saved and no explicit saving is necessary.

If you would like to save an additional copy of the transcript with a different filename, you can use the File > Save Transcript As, or File > Save Transcript menu items. The initial save must be made with the Save Transcript As selection, which stores the filename in the Tcl variable PrefMain(saveFile). Subsequent saves can be made with the Save Transcript selection. Since no automatic saves are performed for this file, it is written only when you invoke a Save command. The file is written to the specified directory and records the contents of the transcript at the time of the save.

Using the saved transcript as a macro (DO file)

Saved transcript files can be used as macros (DO files). See the do command (CR-127) for more information.
The Main window menu bar

The menu bar at the top of the Main window lets you access many ModelSim commands and features. The menus are listed below with brief descriptions of each command’s use.

### File menu

| **New** | provides three options:  
|  | Folder – create a new folder in the current directory  
|  | Source – create a VHDL, Verilog, or Other source file  
|  | Project – create a new project |

| **Open** | provides three options:  
|  | File – open the selected hdl file  
|  | Project – open the selected .mpf project file  
|  | Dataset – open the specified WLF file and assign it the specified dataset name |

| **Close** | provides three options:  
|  | Project – close the currently open project file  
|  | Dataset – close the specified dataset |

| **Delete** | provides one option:  
|  | Project – delete the selected .mpf project file |

| **Change Directory** | change to a different working directory |

| **Save Transcript** | save the current contents of the transcript window to the file indicated with a "Save Transcript As" selection (this selection is not initially available because the transcript is written to the transcript file by default), see "Saving the Main window transcript file" (UM-153) |

| **Save Transcript As...** | save the current contents of the transcript window to a file |

| **Clear Transcript** | clear the Main window transcript display |

| **Options** (all options are set for the current session only) | Transcript File: set a transcript file to save for this session only  
|  | Command History: file for saving command history only, no comments  
|  | Save File: set filename for Save Transcript, and Save Transcript As  
|  | Saved Lines: limit the number of lines saved in the transcript (default is 5000)  
|  | Line Prefix: specify the comment prefix for the transcript  
|  | Update Rate: specify the update frequency for the Main status bar  
|  | ModelSim Prompt: change the title of the ModelSim prompt  
|  | VSIM Prompt: change the title of the VSIM prompt  
|  | Paused Prompt: change the title of the Paused prompt |

| **<path list>** | a list of the most recent working directory changes |

| **Quit** | quit ModelSim |
### Main window

#### Edit menu

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>copy the selected text</td>
</tr>
<tr>
<td>Paste</td>
<td>paste the previously cut or copied text to the left of the currently selected text</td>
</tr>
<tr>
<td>Select All</td>
<td>select all text in the Main window transcript</td>
</tr>
<tr>
<td>Unselect All</td>
<td>deselect all text in the Main window transcript</td>
</tr>
<tr>
<td>Find</td>
<td>search the transcript forward or backward for the specified text string</td>
</tr>
<tr>
<td>Breakpoints</td>
<td>open the Breakpoints dialog box; see &quot;Setting file-line breakpoints&quot; (UM-199) for details</td>
</tr>
</tbody>
</table>

#### Design menu

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browse Libraries</td>
<td>browse all libraries within the scope of the design; see also &quot;Managing library contents&quot; (UM-40)</td>
</tr>
<tr>
<td>Create a New Library</td>
<td>create a new library or map a library to a new name; see &quot;Creating a library&quot; (UM-39)</td>
</tr>
<tr>
<td>Import Library</td>
<td>import FPGA libraries; see &quot;Importing FPGA libraries&quot; (UM-49) for details</td>
</tr>
<tr>
<td>Compile</td>
<td>compile HDL source files into the current project’s work library</td>
</tr>
<tr>
<td>Load Design</td>
<td>initiate simulation by specifying the top level design unit in the Design tab; specify HDL specific simulator settings with the VHDL and Verilog tabs; specify the library to search for design units instantiated from Verilog with the Libraries tab; specify settings relating to the annotation of design timing with the SDF tab</td>
</tr>
<tr>
<td>End Simulation</td>
<td>end the simulation (returns to the ModelSim command line)</td>
</tr>
</tbody>
</table>
View menu

<table>
<thead>
<tr>
<th>All</th>
<th>open all ModelSim windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hide/Show Workspace</td>
<td>hide or show the workspace</td>
</tr>
<tr>
<td>Layout Style(^a)</td>
<td>provides five options:</td>
</tr>
<tr>
<td></td>
<td>Default - restore the window layout to that used for versions 5.5 and later</td>
</tr>
<tr>
<td></td>
<td>Classic - restore the window layout to that used in versions prior to 5.5</td>
</tr>
<tr>
<td></td>
<td>Cascade - Cascade all open windows</td>
</tr>
<tr>
<td></td>
<td>Horizontal - Tile all open windows horizontally</td>
</tr>
<tr>
<td></td>
<td>Vertical - Tile all open windows vertically</td>
</tr>
<tr>
<td>Source</td>
<td>open and/or view the Source window (UM-195)</td>
</tr>
<tr>
<td>Structure</td>
<td>open and/or view the Structure window (UM-204)</td>
</tr>
<tr>
<td>Variables</td>
<td>open and/or view the Variables window (UM-207)</td>
</tr>
<tr>
<td>Signals</td>
<td>open and/or view the Signals window (UM-187)</td>
</tr>
<tr>
<td>List</td>
<td>open and/or view the List window (UM-170)</td>
</tr>
<tr>
<td>Process</td>
<td>open and/or view the Process window (UM-184)</td>
</tr>
<tr>
<td>Wave</td>
<td>open and/or view the Wave window (UM-210)</td>
</tr>
<tr>
<td>Dataflow</td>
<td>open and/or view the Dataflow window (UM-166)</td>
</tr>
<tr>
<td>Datasets</td>
<td>open the Dataset Browser for selecting the current Dataset</td>
</tr>
<tr>
<td>New</td>
<td>create a new window of the specified type</td>
</tr>
<tr>
<td>Other</td>
<td>if the Performance Analyzer and/or Code Coverage is turned on, this selection will allow viewing of: Hierarchical Profile, Ranked Profile, and Source Coverage</td>
</tr>
</tbody>
</table>

\(^a\)You can specify a Layout Style to become the default for ModelSim. After choosing the Layout Style you want, select **Options > Save Preferences** and the layout style will be saved to the PrefMain(layoutStyle) preference variable.

Project menu

<table>
<thead>
<tr>
<th>Compile Order</th>
<th>set the compile order of the files in the open Project; see &quot;Changing compile order&quot; (UM-30) for details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile All</td>
<td>compile all files in the open Project; see &quot;Step 3 — Compile the files&quot; (UM-28) for details</td>
</tr>
<tr>
<td>Add File to Project</td>
<td>add file(s) to the open Project; see &quot;Step 2 — Add files to the project&quot; (UM-27) for details</td>
</tr>
</tbody>
</table>
### Run menu

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run &lt;default&gt;</td>
<td>run simulation for one default run length; change the run length with <strong>Options &gt; Simulation</strong>, or use the Run Length text box on the toolbar</td>
</tr>
<tr>
<td>Run -All</td>
<td>run simulation until you stop it; see also the <code>run</code> command (CR-199)</td>
</tr>
<tr>
<td>Continue</td>
<td>continue the simulation; see also the <code>run</code> command (CR-199) and the <code>-continue</code> option</td>
</tr>
<tr>
<td>Run -Next</td>
<td>run to the next event time</td>
</tr>
<tr>
<td>Step</td>
<td>single-step the simulator; see also the <code>step</code> command (CR-210)</td>
</tr>
<tr>
<td>Step -Over</td>
<td>execute without single-stepping through a subprogram call</td>
</tr>
<tr>
<td>Restart</td>
<td>reload the design elements and reset the simulation time to zero; only design elements that have changed are reloaded; you specify whether to maintain the following after restart—list and wave window environment, breakpoints, logged signals, and virtual definitions; see also the <code>restart</code> command (CR-193)</td>
</tr>
</tbody>
</table>

### Compare menu

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Comparison</td>
<td>start a new comparison</td>
</tr>
<tr>
<td>Comparison Wizard</td>
<td>receive step-by-step assistance while creating a waveform comparison</td>
</tr>
<tr>
<td>Run Comparison</td>
<td>compute differences from time zero until the end of the simulation</td>
</tr>
<tr>
<td>End Comparison</td>
<td>stop difference computation and close the currently open comparison</td>
</tr>
<tr>
<td>Add</td>
<td>provides three options:</td>
</tr>
<tr>
<td></td>
<td>Compare by Signal - specify signals for comparison</td>
</tr>
<tr>
<td></td>
<td>Compare by Region - designate a reference region for a comparison</td>
</tr>
<tr>
<td></td>
<td>Clocks - define clocks to be used in a comparison</td>
</tr>
<tr>
<td>Options</td>
<td>set options for waveform comparisons</td>
</tr>
<tr>
<td>Differences</td>
<td>provides four options:</td>
</tr>
<tr>
<td></td>
<td>Clear - clear all differences from the Wave window</td>
</tr>
<tr>
<td></td>
<td>Show - display differences in a text format in the Main window</td>
</tr>
<tr>
<td></td>
<td>Transcript - save computation differences to a file that can be reloaded later</td>
</tr>
<tr>
<td></td>
<td>Write Report - save computation differences to a text file</td>
</tr>
</tbody>
</table>
| Rules          | provides two options:  
               | Show - display the rules used to set up the waveform comparison  
               | Save - save rules for waveform comparison to a file  
|---------------|----------------------------------------------------------|
| Reload        | load saved differences and rules files  

### Macro menu

| Execute Macro | browse for and execute a DO file (macro)  
|---------------|------------------------------------------|
| Execute Old PE Macro | call and execute an old PE 4.7 macro without changing the macro to SE 5.5d; backslashes can be selected as pathname delimiters  
| Convert Old PE Macro | convert old PE 4.7 macro to SE 5.5d macro without changing the file; backslashes can be selected as pathname delimiters  
| Macro Helper | UNIX only - invoke the Macro Helper tool; see also "The Macro Helper" (UM-264)  
| Tcl Debugger | invoke the Tcl debugger, TDebug; see also "The Tcl Debugger" (UM-265)  
| TclPro Debugger | invoke the TclPro Debugger by Scripits® if installed. TclPro Debugger can be acquired from Scripits.  

### Options menu

| Compile | set both VHDL and Verilog compile options; see "Setting default compile options" (UM-246)  
|---------|------------------------------------------------------------------|
| Simulation | set various simulation options; see "Setting default simulation options" (UM-259)  
| Edit Preferences | set various preference variables; see http://www.model.com/resources/pref_variables/frameset.htm  
| Save Preferences | save current ModelSim settings to a Tcl preference file; see http://www.model.com/resources/pref_variables/frameset.htm  

### Window menu

| Initial Layout | restore all windows to the size and placement of the initial full-screen layout  
|---------------|------------------------------------------------------------------|
| Cascade | cascade all open windows  
| Tile Horizontally | tile all open windows horizontally  
| Tile Vertically | tile all open windows vertically  
| Icon Children | icon all but the Main window  

ModelSim SE User’s Manual
### Main window

<table>
<thead>
<tr>
<th>Icon All</th>
<th>icon all windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deicon All</td>
<td>deicon all windows</td>
</tr>
<tr>
<td>Customize</td>
<td>use the The Button Adder (UM-263) to define and add a button to either the tool or status bar of the specified window</td>
</tr>
<tr>
<td>&lt;window_name&gt;</td>
<td>list of the currently open windows; select a window name to switch to, or show that window if it is hidden; when the source window is available, the source file name is also indicated; open additional windows from the &quot;View menu&quot; (UM-156) in the Main window, or use the view command (CR-250)</td>
</tr>
</tbody>
</table>

### Help menu

<table>
<thead>
<tr>
<th>About ModelSim</th>
<th>display ModelSim application information (e.g., software version)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release Notes</td>
<td>view current release notes with the ModelSim notepad (CR-164)</td>
</tr>
<tr>
<td>Enable Welcome</td>
<td>enable the Welcome screen for starting a new project or opening an existing project when ModelSim is initiated</td>
</tr>
<tr>
<td>Welcome Menu</td>
<td>open the Welcome screen</td>
</tr>
<tr>
<td>Information about Help</td>
<td>view the readme file pertaining to ModelSim’s online documentation</td>
</tr>
<tr>
<td>SE PDF (HTML) Documentation</td>
<td>open and read ModelSim documentation in PDF or HTML format; PDF files can be read with a free Adobe Acrobat reader available on the ModelSim installation CD or from <a href="http://www.adobe.com">www.adobe.com</a></td>
</tr>
<tr>
<td>Tcl Help</td>
<td>open the Tcl command reference (man pages) in Windows help format</td>
</tr>
<tr>
<td>Tcl Syntax</td>
<td>open Tcl syntax details in HTML format</td>
</tr>
<tr>
<td>Tcl Man Pages</td>
<td>open the Tcl /Tk 8.0 manual in HTML format</td>
</tr>
<tr>
<td>Technotes</td>
<td>select a technical note to view from the drop-down list</td>
</tr>
</tbody>
</table>
The Main window toolbar

Buttons on the Main window toolbar give you quick access to these ModelSim commands and functions.

<table>
<thead>
<tr>
<th>Button</th>
<th>Menu equivalent</th>
<th>Command equivalents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile</td>
<td>Design &gt; Compile, also Options &gt; Compile (opens the Compile Options dialog box)</td>
<td>vcom &lt;arguments&gt;, or vlog &lt;arguments&gt;</td>
</tr>
<tr>
<td>Load Design</td>
<td>Design &gt; Load Design</td>
<td>vsim &lt;arguments&gt;</td>
</tr>
<tr>
<td>Copy</td>
<td>Edit &gt; Copy</td>
<td>see: &quot;Mouse and keyboard shortcuts&quot; (UM-162)</td>
</tr>
<tr>
<td>Paste</td>
<td>Edit &gt; Paste</td>
<td>see: &quot;Mouse and keyboard shortcuts&quot; (UM-162)</td>
</tr>
<tr>
<td>Restart</td>
<td>Run &gt; Restart</td>
<td>restart &lt;arguments&gt;</td>
</tr>
<tr>
<td>Run Length</td>
<td>none</td>
<td>run &lt;specific run length&gt;</td>
</tr>
</tbody>
</table>

Main window toolbar buttons
### Main window toolbar buttons

<table>
<thead>
<tr>
<th>Button</th>
<th>Menu equivalent</th>
<th>Command equivalents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run</td>
<td>Run &gt; Run &lt;default_run_length&gt; run (no arguments) see: run (CR-199)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Continue Run</td>
<td>run -continue see: run (CR-199)</td>
</tr>
<tr>
<td></td>
<td>Run -All</td>
<td>run -all see: run (CR-199), see &quot;Assertion settings tab&quot; (UM-260)</td>
</tr>
<tr>
<td>Break</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>Step</td>
<td>Run &gt; Step</td>
<td>step see: step (CR-210)</td>
</tr>
<tr>
<td>Step Over</td>
<td>Run &gt; Step -Over</td>
<td>step -over see: step (CR-210)</td>
</tr>
</tbody>
</table>
The Main window status bar

Fields at the bottom of the Main window provide the following information about the current simulation:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Now</td>
<td>the current simulation time, using the default resolution units (see &quot;Simulating with the graphic interface&quot; (UM-250)), or a larger time unit if one can be used without a fractional remainder</td>
</tr>
<tr>
<td>Delta</td>
<td>the current simulation iteration number</td>
</tr>
<tr>
<td>&lt;dataset name&gt;</td>
<td>name of the current dataset (item selected in the Structure window (UM-204))</td>
</tr>
</tbody>
</table>

Mouse and keyboard shortcuts

The following mouse actions and special keystrokes can be used to edit commands in the entry region of the Main window. They can also be used in editing the file displayed in the Source window and all Notepad windows (enter the notepad command within ModelSim to open the Notepad editor).

<table>
<thead>
<tr>
<th>Mouse - UNIX</th>
<th>Mouse - Windows</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; left-button - click &gt;</td>
<td></td>
<td>move the insertion cursor</td>
</tr>
<tr>
<td>&lt; left-button - press &gt; + drag</td>
<td></td>
<td>select</td>
</tr>
<tr>
<td>&lt; shift - left-button - press &gt;</td>
<td></td>
<td>extend selection</td>
</tr>
<tr>
<td>&lt; left-button - double-click &gt;</td>
<td></td>
<td>select word</td>
</tr>
<tr>
<td>&lt; left-button - double-click &gt; + drag</td>
<td></td>
<td>select word + word</td>
</tr>
<tr>
<td>&lt; control - left-button - click &gt;</td>
<td></td>
<td>move insertion cursor without changing the selection</td>
</tr>
<tr>
<td>&lt; left-button - click &gt; on previous ModelSim or VSIM prompt</td>
<td></td>
<td>copy and paste previous command string to current prompt</td>
</tr>
<tr>
<td>&lt; middle-button - click &gt;</td>
<td>none</td>
<td>paste clipboard</td>
</tr>
<tr>
<td>&lt; middle-button - press &gt; + drag</td>
<td>none</td>
<td>scroll the window</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Keystrokes - UNIX</th>
<th>Keystrokes - Windows</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; left</td>
<td>right - arrow &gt;</td>
<td></td>
</tr>
</tbody>
</table>

ModelSim SE User’s Manual
<table>
<thead>
<tr>
<th>Keystrokes - UNIX</th>
<th>Keystrokes - Windows</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>`&lt; control &gt; &lt; left</td>
<td>right - arrow &gt;`</td>
<td></td>
</tr>
<tr>
<td>`&lt; shift &gt; &lt; left</td>
<td>right</td>
<td>up</td>
</tr>
<tr>
<td>`&lt; control &gt; &lt; shift &gt; &lt; left</td>
<td>right - arrow &gt;`</td>
<td></td>
</tr>
<tr>
<td>`&lt; up</td>
<td>down - arrow &gt;`</td>
<td></td>
</tr>
<tr>
<td>`&lt; control &gt; &lt; up</td>
<td>down &gt;`</td>
<td></td>
</tr>
<tr>
<td><code>&lt; control &gt; &lt; home &gt;</code></td>
<td></td>
<td>move cursor to the beginning of the text</td>
</tr>
<tr>
<td><code>&lt; control &gt; &lt; end &gt;</code></td>
<td></td>
<td>move cursor to the end of the text</td>
</tr>
<tr>
<td><code>&lt; backspace &gt;, &lt; control-h &gt;</code></td>
<td><code>&lt; backspace &gt;</code></td>
<td>delete character to the left</td>
</tr>
<tr>
<td><code>&lt; delete &gt;, &lt; control-d &gt;</code></td>
<td><code>&lt; delete &gt;</code></td>
<td>delete character to the right</td>
</tr>
<tr>
<td>none</td>
<td><code>esc</code></td>
<td>cancel</td>
</tr>
<tr>
<td><code>&lt; alt &gt;</code></td>
<td></td>
<td>activate or inactivate menu bar mode</td>
</tr>
<tr>
<td><code>&lt; alt &gt; &lt; F4 &gt;</code></td>
<td></td>
<td>close active window</td>
</tr>
<tr>
<td><code>&lt; control - a &gt;, &lt; home &gt;</code></td>
<td><code>&lt; home &gt;</code></td>
<td>move cursor to the beginning of the line</td>
</tr>
<tr>
<td><code>&lt; control - b &gt;</code></td>
<td></td>
<td>move cursor left</td>
</tr>
<tr>
<td><code>&lt; control - d &gt;</code></td>
<td></td>
<td>delete character to the right</td>
</tr>
<tr>
<td><code>&lt; control - e &gt;, &lt; end &gt;</code></td>
<td><code>&lt; end &gt;</code></td>
<td>move cursor to the end of the line</td>
</tr>
<tr>
<td><code>&lt; control - f &gt;</code></td>
<td></td>
<td>move cursor right one character</td>
</tr>
<tr>
<td><code>&lt; control - k &gt;</code></td>
<td></td>
<td>delete to the end of line</td>
</tr>
<tr>
<td><code>&lt; control - n &gt;</code></td>
<td></td>
<td>move cursor one line down (Source window only under Windows)</td>
</tr>
<tr>
<td><code>&lt; control - o &gt;</code></td>
<td><code>none</code></td>
<td>insert a newline character in front of the cursor</td>
</tr>
<tr>
<td><code>&lt; control - p &gt;</code></td>
<td></td>
<td>move cursor one line up (Source window only under Windows)</td>
</tr>
<tr>
<td><code>&lt; control - s &gt;</code></td>
<td><code>&lt; control - f &gt;</code></td>
<td>find</td>
</tr>
<tr>
<td><code>&lt; F3 &gt;</code></td>
<td></td>
<td>find next</td>
</tr>
<tr>
<td><code>&lt; control - t &gt;</code></td>
<td></td>
<td>reverse the order of the two characters to the right of the cursor</td>
</tr>
<tr>
<td><code>&lt; control - u &gt;</code></td>
<td></td>
<td>delete line</td>
</tr>
<tr>
<td><code>&lt; control - v &gt;</code></td>
<td><code>PageDn</code></td>
<td>move cursor down one screen</td>
</tr>
<tr>
<td><code>&lt; control - w &gt;</code></td>
<td><code>&lt; control - x &gt;</code></td>
<td>cut the selection</td>
</tr>
<tr>
<td>Keystrokes - UNIX</td>
<td>Keystrokes - Windows</td>
<td>Result</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------</td>
<td>--------</td>
</tr>
<tr>
<td>&lt; control &gt; &lt; left</td>
<td>right - arrow &gt;</td>
<td>move cursor left</td>
</tr>
<tr>
<td>&lt; shift &gt; &lt; left</td>
<td>right</td>
<td>up</td>
</tr>
<tr>
<td>&lt; control &gt; &lt; shift &gt; &lt; left</td>
<td>right - arrow &gt;</td>
<td>extend selection of text by word</td>
</tr>
<tr>
<td>&lt; up</td>
<td>down - arrow &gt;</td>
<td>scroll through command history (in Source window, moves cursor one line up</td>
</tr>
<tr>
<td>&lt; control &gt; &lt; up</td>
<td>down &gt;</td>
<td>moves cursor up</td>
</tr>
<tr>
<td>&lt; control &gt; &lt; home &gt;</td>
<td>move cursor to the beginning of the text</td>
<td></td>
</tr>
<tr>
<td>&lt; control &gt; &lt; end &gt;</td>
<td>move cursor to the end of the text</td>
<td></td>
</tr>
<tr>
<td>&lt; backspace &gt;, &lt; control-h &gt;</td>
<td>&lt; backspace &gt;</td>
<td>delete character to the left</td>
</tr>
<tr>
<td>&lt; delete &gt;, &lt; control-d &gt;</td>
<td>&lt; delete &gt;</td>
<td>delete character to the right</td>
</tr>
<tr>
<td>none</td>
<td>esc</td>
<td>cancel</td>
</tr>
<tr>
<td>&lt; alt &gt;</td>
<td>activate or inactivate menu bar mode</td>
<td></td>
</tr>
<tr>
<td>&lt; alt &gt; &lt; F4 &gt;</td>
<td>close active window</td>
<td></td>
</tr>
<tr>
<td>&lt; control - a &gt;, &lt; home &gt;</td>
<td>&lt; home &gt;</td>
<td>move cursor to the beginning of the line</td>
</tr>
<tr>
<td>&lt; control - b &gt;</td>
<td>move cursor left</td>
<td></td>
</tr>
<tr>
<td>&lt; control - d &gt;</td>
<td>delete character to the right</td>
<td></td>
</tr>
<tr>
<td>&lt; control - e &gt;, &lt; end &gt;</td>
<td>&lt; end &gt;</td>
<td>move cursor to the end of the line</td>
</tr>
<tr>
<td>&lt; control - f &gt;</td>
<td>move cursor right one character</td>
<td></td>
</tr>
<tr>
<td>&lt; control - k &gt;</td>
<td>delete to the end of line</td>
<td></td>
</tr>
<tr>
<td>&lt; control - n &gt;</td>
<td>move cursor one line down (Source window only under Windows)</td>
<td></td>
</tr>
<tr>
<td>&lt; control - o &gt;</td>
<td>none</td>
<td>insert a newline character in front of the cursor</td>
</tr>
<tr>
<td>&lt; control - p &gt;</td>
<td>move cursor one line up (Source window only under Windows)</td>
<td></td>
</tr>
<tr>
<td>&lt; control - s &gt;</td>
<td>&lt; control - f &gt;</td>
<td>find</td>
</tr>
<tr>
<td>&lt; F3 &gt;</td>
<td>find next</td>
<td></td>
</tr>
<tr>
<td>&lt; control - t &gt;</td>
<td>reverse the order of the two characters to the right of the cursor</td>
<td></td>
</tr>
<tr>
<td>&lt; control - u &gt;</td>
<td>delete line</td>
<td></td>
</tr>
<tr>
<td>&lt; control - v &gt;</td>
<td>PageDn</td>
<td>move cursor down one screen</td>
</tr>
<tr>
<td>&lt; control - w &gt;</td>
<td>&lt; control - x &gt;</td>
<td>cut the selection</td>
</tr>
<tr>
<td>Keystrokes - UNIX</td>
<td>Keystrokes - Windows</td>
<td>Result</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------</td>
<td>--------</td>
</tr>
<tr>
<td>\textless{} control - x \textgreater{}, \textless{} control - s \textgreater{}</td>
<td>\textless{} control - s \textgreater{}</td>
<td>save</td>
</tr>
<tr>
<td>\textless{} control - y \textgreater{}, F18</td>
<td>\textless{} control - v \textgreater{}</td>
<td>paste the selection</td>
</tr>
<tr>
<td>none</td>
<td>\textless{} control - a \textgreater{}</td>
<td>select the entire contents of the widget</td>
</tr>
<tr>
<td>\textless{} control - \textbackslash{} \textgreater{}</td>
<td>clear any selection in the widget</td>
<td></td>
</tr>
<tr>
<td>\textless{} control - _\textgreater{}, \textless{} control - / \textgreater{}</td>
<td>\textless{} control - Z \textgreater{}</td>
<td>undoes previous edits in the Source window</td>
</tr>
<tr>
<td>\textless{} meta - &quot;&lt;&quot; \textgreater{}</td>
<td>none</td>
<td>move cursor to the beginning of the file</td>
</tr>
<tr>
<td>\textless{} meta - &quot;&gt;&quot; \textgreater{}</td>
<td>none</td>
<td>move cursor to the end of the file</td>
</tr>
<tr>
<td>\textless{} meta - v \textgreater{}</td>
<td>PageUp</td>
<td>move cursor up one screen</td>
</tr>
<tr>
<td>\textless{} Meta - w \textgreater{}</td>
<td>\textless{} control - c \textgreater{}</td>
<td>copy selection</td>
</tr>
<tr>
<td>\textless{} F8 \textgreater{}</td>
<td></td>
<td>search for the most recent command that matches the characters typed (Main window only)</td>
</tr>
<tr>
<td>\textless{} F9 \textgreater{}</td>
<td></td>
<td>run simulation</td>
</tr>
<tr>
<td>\textless{} F10 \textgreater{}</td>
<td></td>
<td>continue simulation</td>
</tr>
<tr>
<td>\textless{} F11 \textgreater{}</td>
<td></td>
<td>single-step</td>
</tr>
<tr>
<td>\textless{} F12 \textgreater{}</td>
<td></td>
<td>step-over</td>
</tr>
</tbody>
</table>

The Main window allows insertions or pastes only after the prompt; therefore, you don’t need to set the cursor when copying strings to the command line.
Dataflow window

The Dataflow window allows you to trace VHDL signals or Verilog nets and registers through your design. Double-click an item with the left mouse button to move it to the center of the Dataflow display.

**VHDL signals or processes in the Dataflow window:**
- A signal is displayed in the center of the window with all the processes that drive the signal on the left, and all the processes that read the signal on the right.
- A process is displayed with all the signals read by the process shown as inputs on the left of the window, and all the signals driven by the process on the right.

**Verilog nets/registers or processes in the Dataflow window:**
- A net or register is displayed in the center of the window with all the processes that drive the net or register on the left, and all the processes triggered by the net or register on the right.
- A process is displayed with all the nets or registers that trigger the process shown as inputs on the left of the window, and all the nets or registers driven by the process on the right.

**Link to active cursor in Wave window**

In versions 5.5 and later, the value of a signal, net, or register in the Dataflow window is linked to the active cursor in the Wave window. As you move the active cursor in the Wave window, the value of the signal, net, or register in the Dataflow window will update.
Dataflow window menu bar

The following menu commands and button options are available from the Dataflow window menu bar.

File menu

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save Postscript</td>
<td>save the current dataflow view as a Postscript file; see &quot;Saving the Dataflow window as a Postscript file&quot; (UM-169)</td>
</tr>
<tr>
<td>Selection</td>
<td>Selection &gt; Follow Selection updates the Dataflow window when the Process window (UM-184) or Signals window (UM-187) changes; Selection &gt; Fix Selection freezes the view selected from within the Dataflow window</td>
</tr>
<tr>
<td>Close</td>
<td>close this copy of the Dataflow window; you can create a new window with View &gt; New from the &quot;The Main window menu bar&quot; (UM-154)</td>
</tr>
</tbody>
</table>

Window menu

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Layout</td>
<td>restore all windows to the size and placement of the initial full-screen layout</td>
</tr>
<tr>
<td>Cascade</td>
<td>cascade all open windows</td>
</tr>
<tr>
<td>Tile Horizontally</td>
<td>tile all open windows horizontally</td>
</tr>
<tr>
<td>Tile Vertically</td>
<td>tile all open windows vertically</td>
</tr>
<tr>
<td>Icon Children</td>
<td>icon all but the Main window</td>
</tr>
<tr>
<td>Icon All</td>
<td>icon all windows</td>
</tr>
<tr>
<td>Deicon All</td>
<td>deicon all windows</td>
</tr>
<tr>
<td>Customize</td>
<td>use the The Button Adder (UM-263) to define and add a button to either the tool or status bar of the specified window</td>
</tr>
<tr>
<td>&lt;window_name&gt;</td>
<td>list of the currently open windows; select a window name to switch to, or show that window if it is hidden; when the source window is available, the source file name is also indicated; open additional windows from the &quot;View menu&quot; (UM-156) in the Main window, or use the view command (CR-250)</td>
</tr>
</tbody>
</table>
Tracing HDL items with the Dataflow window

The Dataflow window is linked with the Signals window (UM-187) and the Process window (UM-184). To examine a particular process in the Dataflow window, click on the process name in the Process window. To examine a particular HDL item in the Dataflow window, click on the item name in the Signals window.

**With a signal in the center** of the Dataflow window, you can:
- click once on a process name in the Dataflow window to make the Source, Process, Signals, and Variable windows update to show that process,
- click twice on a process name in the Dataflow window to move the process to the center of the Dataflow window

**With a process in the center** of the Dataflow window, you can:
- click twice on an item name to move that item to the center of the Dataflow window.

The backward and forward buttons on the toolbar are analogous to Back and Forward buttons in a web browser. They move backward or forward through previous views of the dataflow.

| ![←] | move backward through dataflow views |
| ![→] | move forward through dataflow views |

The Dataflow window will display the current process when you single-step or when ModelSim hits a breakpoint.
Saving the Dataflow window as a Postscript file

Select **File > Save Postscript** (Dataflow window) to save the current Dataflow view as a Postscript file. Configure the Postscript output with the following dialog box, or use the **Options > Edit Preferences** (Main window) command.

The dialog box has the following options:

- **Postscript File**
  specify the name of the file to save, default is `dataflow.ps`

- **Orientation**
  specify **Landscape** (horizontal) or **Portrait** (vertical) orientation

- **Color Mode**
  specify **Color** (256 colors), **Gray** (gray-scale) or **Mono** (monochrome) color mode

- **Postscript**
  specify Normal Postscript or EPS (Encapsulated Postscript) file type

- **Color Map**
  specify the color mapping from current Dataflow window colors to Postscript colors
List window

The List window displays the results of your simulation run in tabular format. The window is divided into two adjustable panes, which allow you to scroll horizontally through the listing on the right, while keeping time and delta visible on the left.

HDL items you can view

One entry is created for each of the following VHDL and Verilog HDL items within the design:

- **VHDL items**
  - signals and process variables
- **Verilog items**
  - nets and register variables
- **Comparison items**
  - comparison regions and comparison signals; see Chapter 11 - Waveform Comparison for more information
- **Virtual items**
  - Virtual signals and functions

▶ **Note:** Constants, generics, and parameters are not viewable in the List or Wave windows.
The List window menu bar

The following menu commands are available from the List window menu bar.

**File menu**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write List (format)</td>
<td>save the listing as a text file in one of three formats: tabular, events, or TSSI</td>
</tr>
<tr>
<td>Load Format</td>
<td>run a List window format DO file previously saved with Save Format</td>
</tr>
<tr>
<td>Save Format</td>
<td>save the current List window display and signal preferences to a DO (macro) file; running the DO file will reformat the List window to match the display as it appeared when the DO file was created</td>
</tr>
<tr>
<td>Close</td>
<td>close this copy of the List window; you can create a new window with View &gt; New from the &quot;The Main window menu bar&quot; (UM-154)</td>
</tr>
</tbody>
</table>

**Edit menu**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td>cut the selected item field from the listing; see &quot;Editing and formatting HDL items in the List window” (UM-176)</td>
</tr>
<tr>
<td>Copy</td>
<td>copy the selected item field</td>
</tr>
<tr>
<td>Paste</td>
<td>paste the previously cut or copied item to the left of the currently selected item</td>
</tr>
<tr>
<td>Delete</td>
<td>delete the selected item field</td>
</tr>
<tr>
<td>Combine</td>
<td>combine the selected fields into a user-defined bus; keep copies of the original items rather than moving them; see &quot;Combining signals into a user-defined bus” (UM-149)</td>
</tr>
<tr>
<td>Select All</td>
<td>select all signals in the List window</td>
</tr>
<tr>
<td>Unselect All</td>
<td>deselect all signals in the List window</td>
</tr>
<tr>
<td>Find</td>
<td>find the specified item label within the List window</td>
</tr>
<tr>
<td>Search</td>
<td>search the List window for a specified value, or the next transition for the selected signal</td>
</tr>
</tbody>
</table>

**Markers menu**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Marker</td>
<td>add a time marker at the currently selected line</td>
</tr>
<tr>
<td>Delete Marker</td>
<td>delete the selected marker from the listing</td>
</tr>
<tr>
<td>Goto</td>
<td>choose the time marker to go to from a list of current markers</td>
</tr>
</tbody>
</table>
### Prop menu

<table>
<thead>
<tr>
<th>Display Props</th>
<th>set display properties for all items in the window: delta settings, trigger on selection, strobe period, label size, and dataset prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal Props</td>
<td>set label, radix, trigger on/off, and field width for the selected item</td>
</tr>
</tbody>
</table>

### Window menu

<table>
<thead>
<tr>
<th>Initial Layout</th>
<th>restore all windows to the size and placement of the initial full-screen layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cascade</td>
<td>cascade all open windows</td>
</tr>
<tr>
<td>Tile Horizontally</td>
<td>tile all open windows horizontally</td>
</tr>
<tr>
<td>Tile Vertically</td>
<td>tile all open windows vertically</td>
</tr>
<tr>
<td>Icon Children</td>
<td>icon all but the Main window</td>
</tr>
<tr>
<td>Icon All</td>
<td>icon all windows</td>
</tr>
<tr>
<td>Deicon All</td>
<td>deicon all windows</td>
</tr>
<tr>
<td>Customize</td>
<td>use the The Button Adder (UM-263) to define and add a button to either the tool or status bar of the specified window</td>
</tr>
<tr>
<td>&lt;window_name&gt;</td>
<td>list of the currently open windows; select a window name to switch to, or show that window if it is hidden; when the source window is available, the source file name is also indicated; open additional windows from the &quot;View menu&quot; (UM-156) in the Main window, or use the view command (CR-250)</td>
</tr>
</tbody>
</table>
Setting List window display properties

Before you add items to the List window you can set the window’s display properties. To change when and how a signal is displayed in the List window, select Prop > Display Props (List window). The resulting Modify Display Properties dialog box contains options for Trigger Settings and Window Properties.

Window Properties tab

The Window Properties tab includes these options:

- **Signal Names**
  Sets the number of path elements to be shown in the List window. For example, "0" shows the full path. "1" shows only the leaf element.

- **Max Title Rows**
  Sets the maximum number of rows in the name pane.

- **Dataset Prefix: Show All Dataset Prefixes**
  Displays the dataset prefix associated with each signal pathname. Useful for displaying signals from multiple datasets.

- **Dataset Prefix: Show All Except "sim"**
  Displays all dataset prefixes except the one associated with the current simulation – "sim." Useful for displaying signals from multiple datasets.
- **Dataset Prefix: Show No Dataset Prefixes**
  Turns off display of dataset prefixes.

**Trigger settings tab**

The Triggers tab controls the triggering for the display of new lines in the List window. You can specify whether an HDL item trigger or a strobe trigger is used to determine when the List window displays a new line. If you choose **Trigger on: Signals**, then you can choose between collapsed or expanded delta displays. You can also choose a combination of signal or strobe triggers. To use gating, Signals or Strobe or both must be selected.

The Triggers tab includes the following options:

- **Deltas: Expand Deltas**
  When selected with the **Trigger on: Signals** check box, displays a new line for each time step on which items change, including deltas within a single unit of time resolution.

- **Deltas: Collapse Deltas**
  Displays only the final value for each time unit.

- **Deltas: No Deltas**
  Hides simulation cycle (delta) column.

- **Trigger On: Signal Change**
  Triggers on signal changes. Defaults to all signals. Individual signals can be excluded.
from triggering by using the Prop > Signals Props dialog box or by originally adding them with the -notrigger option to the add list command (CR-49).

- **Trigger On: Strobe**
  Triggers on the Strobe Period you specify; specify the first strobe with First Strobe at:

- **Trigger Gating: Use Gating Expression**
  Enables triggers to be gated on (a value of 1) or off (a value of 0) by the specified Expression.

- **Trigger Gating: Expression**
  Enables triggers to be gated on and off by an overriding expression, much like a hardware signal analyzer might be set up to start recording data on a specified setup of address bits and clock edges. Affects the display of data, not the acquisition of the data.

- **Use Expression Builder** (button)
  Opens the Expression Builder to help you write a gating expression. See "The GUI Expression Builder" (UM-269)

- **Expression**
  Enter the expression for trigger gating into this field, or use the Expression Builder (select the Use Expression Builder button). The expression is evaluated when the List window would normally have displayed a row of data (given the trigger on signals and strobe settings above).

- **Trigger Gating: On Duration**
  The duration for gating to remain open after the last list row in which the expression evaluates to true; expressed in x number of default timescale units. Gating is level-sensitive rather than edge-triggered.

List window gating information is saved as configuration statements when the list format is saved. The gating portion of a configuration statement might look like this:

```bash
configure list config -usegating 1
configure list config -gateduration 100
configure list config -gateexpr '{<expression>}
```

### Adding HDL items to the List window

Before adding items to the List window you may want to set the window display properties (see "Setting List window display properties" (UM-173)). You can add items to the List window in several ways.

#### Adding items with drag and drop

You can drag and drop items into the List window from the Signals, Source, Process, Variables, Wave, Dataflow, or Structure window. Select the items in the first window, then drop them into the List window. Depending on what you select, all items or any portion of the design may be added.

#### Adding items from the Main window command line

Invoke the add list (CR-49) command to add one or more individual items; separate the names with a space:

```bash
add list <item_name> <item_name>
```
You can add all the items in the current region with this command:

```
add list *
```

Or add all the items in the design with:

```
add list -r / *
```

**Adding items with a List window format file**

To use a List window format file you must first save a format file for the design you are simulating. The saved format file can then be used as a DO file to recreate the List window formatting. Follow these steps:

- Add HDL items to your List window.
- Edit and format the items to create the view you want (see "Editing and formatting HDL items in the List window" (UM-176)).
- Save the format to a file by selecting File > Save Format (List window).

To use the format (do) file, start with a blank List window, and run the DO file in one of two ways:

- Invoke the `do` (CR-127) command from the command line:
  ```
do <my_list_format>
```
- Select File > Load Format from the List window menu bar.

Select Edit > Select All and Edit > Delete to remove the items from the current List window or create a new, blank List window by selecting View > New > List (Main window). You may find it useful to have two differently formatted windows open at the same time, see "Examining simulation results with the List window" (UM-179).

**Note:** List window format files are design-specific; use them only with the design you were simulating when they were created. If you try to use the wrong format file, ModelSim will advise you of the HDL items it expects to find.

**Editing and formatting HDL items in the List window**

Once you have the HDL items you want in the List window, you can edit and format the list to create the view you find most useful. (See also, "Adding HDL items to the List window" (UM-175))

**To edit an item:**

Select the item’s label at the top of the List window or one of its values from the listing. Move, copy or remove the item by selecting commands from the List window Edit menu (UM-171) menu.

You can also click+drag to move items within the window:

- to select several contiguous items:
  click+drag to select additional items to the right or the left of the original selection
- to select several items randomly:
  Control+click to add or subtract from the selected group
- to move the selected items:
  re-click on one of the selected items, hold and drag it to the new location
To format an item:

Select the item’s label at the top of the List window or one of its values from the listing, then select Prop > Signal Props (List window). The resulting Modify Signal Properties dialog box allows you to set the item’s label, label width, triggering, and radix.

The Modify Signal Properties dialog box includes these options:

- **Signal**
  Shows the full pathname of the selected signal.

- **Label**
  Specifies the label that appears at the top of the List window column.

- **Radix**
  Specifies the radix (base) in which the item value is expressed. The default radix is symbolic, which means that for an enumerated type, the List window lists the actual values of the enumerated type of that item. You can change the default radix for the current simulation using either Options > Simulation (Main window) or the radix command (CR-189). You can change the default radix permanently by editing the DefaultRadix (UM-389) variable in the modelsim.ini file.

For the other radices - binary, octal, decimal, unsigned, hexadecimal, or ASCII - the item value is converted to an appropriate representation in that radix. In the system initialization file, modelsim.tcl, you can specify the list translation rules for arrays of enumerated types for binary, octal, decimal, unsigned decimal, or hexadecimal item values in the design unit.
• **Width**
  Allows you to specify the desired width of the column used to list the item value. The default is an approximation of the width of the current value.

• **Trigger: Triggers line**
  Specifies that a change in the value of the selected item causes a new line to be displayed in the List window.

• **Trigger: Does not trigger line**
  Specifies that a change in the value of the selected item does not affect the List window.

The trigger specification affects the trigger property of the selected item. See also, "Setting List window display properties" (UM-173).
Examining simulation results with the List window

Because you can use the Main window View menu (UM-156) to create a second List window, you can reformat another List window after the simulation run if you decide a different format would reveal the information you’re after. Compare the two illustrations.

The divider bar separates time and delta from values; signal values are listed in symbolic format; and an item change triggers a new line.

<table>
<thead>
<tr>
<th>ns</th>
<th>delta</th>
<th>/top/clk</th>
<th>/top/pw</th>
<th>/top/paddr</th>
<th>/top/pdata</th>
<th>/top/sw</th>
<th>/top/stwb</th>
<th>/top/prdy</th>
<th>/top/paddr</th>
<th>/top/saddr</th>
</tr>
</thead>
<tbody>
<tr>
<td>530</td>
<td>+0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>535</td>
<td>+0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>520</td>
<td>+0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>540</td>
<td>+0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>530</td>
<td>+0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>535</td>
<td>+0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>530</td>
<td>+0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>540</td>
<td>+0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>530</td>
<td>+0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>535</td>
<td>+0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>530</td>
<td>+0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>540</td>
<td>+0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Signal values are listed in decimal format;

<table>
<thead>
<tr>
<th>ns</th>
<th>/top/clk</th>
<th>/top/pw</th>
<th>/top/paddr</th>
<th>/top/pdata</th>
<th>/top/sw</th>
<th>/top/stwb</th>
<th>/top/prdy</th>
<th>/top/paddr</th>
<th>/top/saddr</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>505</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>520</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>540</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>560</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>580</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>585</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>600</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

In the first List window, the HDL items are formatted as symbolic and use an item change to trigger a line; the field width was changed to accommodate the default label width. The window divider maintains the time and delta in the left pane; signals in the right pane can be viewed by scrolling. For the second listing, the item radix for `paddr`, `pdata`, `saddr`, and `sdata` is now decimal.
Finding items by name in the List window

The Find dialog box allows you to search for text strings in the List window. Select **Edit > Find** (List window) to bring up the Find dialog box.

Enter a text string and **Find** it by searching **Right** or **Left** through the List window display. Specify **Name** to search the real pathnames of the items or **Label** to search their assigned names (see “Setting List window display properties” (UM-173)). Checking **Auto Wrap** makes the search continue at the beginning of the window. Note that you can change an item’s label.

Searching for item values in the List window

Select an item in the List window. Select **Edit > Search** (List window) to bring up the List Signal Search dialog box.
**Signal Name(s)** shows a list of the items currently selected in the List window. These items are the subject of the search. The search is based on these options:

- **Search Type: Any Transition**
  Searches for any transition in the selected signal(s).

- **Search Type: Rising Edge**
  Searches for rising edges in the selected signal(s).

- **Search Type: Falling Edge**
  Searches for falling edges in the selected signal(s).

- **Search Type: Search for Signal Value**
  Searches for the value specified in the **Value** field; the value should be formatted using VHDL or Verilog numbering conventions; see "Numbering conventions" (CR-13).

- **Search Type: Search for Expression**
  Searches for the expression specified in the **Expression** field evaluating to a boolean true. Activates the **Builder** button so you can use "The GUI Expression Builder" (UM-269) if desired.
  
  The expression can involve more than one signal but is limited to signals logged in the List window. Expressions can include constants, variables, and DO files. If no expression is specified, the search will give an error. See "Expression syntax" (CR-23) for more information.

- **Search Options: Match Count**
  Indicates the number of transitions or matches to search. You can search for the n-th transition or the n-th match on value.

- **Search Options: Ignore Glitches**
  Ignores zero width glitches in VHDL signals and Verilog nets.

The **Search Results** are indicated at the bottom of the dialog box.

**Note:** If your signal values are displayed in binary radix, see "Searching for binary signal values in the GUI" (CR-22) for details on how signal values are mapped between a binary radix and std_logic.
Setting time markers in the List window

Select Markers > Add Marker (List window) to tag the selected list line with a marker. The marker is indicated by a thin box surrounding the marked line. The selected line uses the same indicator, but its values are highlighted. Delete markers by first selecting the marked line, then selecting Markers > Delete Marker.

Finding a marker

Choose a specific marked line to view by selecting Markers > Goto. The marker name (on the Goto list) corresponds to the simulation time of the selected line.

List window keyboard shortcuts

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;arrow up&gt;</td>
<td>scroll listing up (selects and highlights the line above the currently selected line)</td>
</tr>
<tr>
<td>&lt;arrow down&gt;</td>
<td>scroll listing down (selects and highlights the line below the currently selected line)</td>
</tr>
<tr>
<td>&lt;arrow left&gt;</td>
<td>scroll listing left</td>
</tr>
<tr>
<td>&lt;arrow right&gt;</td>
<td>scroll listing right</td>
</tr>
<tr>
<td>&lt;page up&gt;</td>
<td>scroll listing up by page</td>
</tr>
</tbody>
</table>
Saving List window data to a file

Select File > Write List (format) (List window) to save the List window data in one of these formats:

• **tabular**
  - writes a text file that looks like the window listing

<table>
<thead>
<tr>
<th>ns</th>
<th>delta</th>
<th>/a</th>
<th>/b</th>
<th>/cin</th>
<th>/sum</th>
<th>/cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+0</td>
<td>X</td>
<td>X</td>
<td>U</td>
<td>X</td>
<td>U</td>
</tr>
<tr>
<td>0</td>
<td>+1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>U</td>
</tr>
<tr>
<td>2</td>
<td>+0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>U</td>
</tr>
</tbody>
</table>

• **event**
  - writes a text file containing transitions during simulation

  ```
  @0 +0
  /a X
  /b X
  /cin U
  /sum X
  /cout U
  @0 +1
  /a 0
  /b 1
  /cin 0
  ```

• **TSSI**
  - writes a file in standard TSSI format; see also, the write tssi command (CR-311)

  ```
  0 00000000000000010??????????
  2 00000000000000010????????1?
  3 00000000000000010????????010
  4 000000000000000100000000010
  100 00000001000000010000000010
  ```

You can also save List window output using the write list command (CR-307).
Process window

The Process window displays a list of processes. If View > Active is selected then all processes scheduled to run during the current simulation cycle are displayed along with the pathname of the instance in which each process is located. If View > In Region is selected then only the processes in the currently selected region are displayed.

Each HDL item in the scrollbox is preceded by one of the following indicators:

- **<Ready>**
  Indicates that the process is scheduled to be executed within the current delta time.

- **<Wait>**
  Indicates that the process is waiting for a VHDL signal or Verilog net or variable to change or for a specified time-out period.

- **<Done>**
  Indicates that the process has executed a VHDL wait statement without a time-out or a sensitivity list. The process will not restart during the current simulation run.

If you select a "Ready" process, it will be executed next by the simulator.

When you click on a process in the Process window, the following windows are updated:

<table>
<thead>
<tr>
<th>Window updated</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure window (UM-204)</td>
<td>shows the region in which the process is located</td>
</tr>
<tr>
<td>Variables window (UM-207)</td>
<td>shows the VHDL variables and Verilog register variables in the process</td>
</tr>
<tr>
<td>Source window (UM-195)</td>
<td>shows the associated source code</td>
</tr>
<tr>
<td>Dataflow window (UM-166)</td>
<td>shows the process, the signals, nets, and registers the process reads, and the signals, nets, and registers driven by the process</td>
</tr>
<tr>
<td>Source window (UM-195)</td>
<td>shows the signals, nets, and registers declared in the region in which the process is located</td>
</tr>
</tbody>
</table>
The Process window menu bar

The following menu commands are available from the Process window menu bar.

**File menu**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save As</td>
<td>Save the process tree to a text file viewable with the ModelSim notepad (CR-164)</td>
</tr>
<tr>
<td>Environment</td>
<td><strong>Follow Context Selection</strong>: update the window based on the selection in the Structure window (UM-204); <strong>Fix to Current Context</strong>: maintain the current view, do not update</td>
</tr>
<tr>
<td>Close</td>
<td>Close this copy of the Process window; you can create a new window with View &gt; New from the &quot;The Main window menu bar&quot; (UM-154)</td>
</tr>
</tbody>
</table>

**Edit menu**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>Copy the selected process’ full name</td>
</tr>
<tr>
<td>Sort</td>
<td>Sort the process list in either ascending, descending, or declaration order</td>
</tr>
<tr>
<td>Select All</td>
<td>Select all processes in the Process window</td>
</tr>
<tr>
<td>Unselect All</td>
<td>Deselect all processes in the Process window</td>
</tr>
<tr>
<td>Find</td>
<td>Find the specified text string within the process list; choose the Status (ready, wait or done), the Process label, or the path to search, and the search direction: down or up</td>
</tr>
</tbody>
</table>

**View menu**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Display all the processes that are scheduled to run during the current simulation cycle</td>
</tr>
<tr>
<td>In Region</td>
<td>Display any processes that exist in the region that is selected in the Structure window</td>
</tr>
</tbody>
</table>
**Window menu**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Layout</td>
<td>restore all windows to the size and placement of the initial full-screen layout</td>
</tr>
<tr>
<td>Cascade</td>
<td>cascade all open windows</td>
</tr>
<tr>
<td>Tile Horizontally</td>
<td>tile all open windows horizontally</td>
</tr>
<tr>
<td>Tile Vertically</td>
<td>tile all open windows vertically</td>
</tr>
<tr>
<td>Icon Children</td>
<td>icon all but the Main window</td>
</tr>
<tr>
<td>Icon All</td>
<td>icon all windows</td>
</tr>
<tr>
<td>Deicon All</td>
<td>deicon all windows</td>
</tr>
<tr>
<td>Customize</td>
<td>use the <em>The Button Adder</em> (UM-263) to define and add a button to either the tool or status bar of the specified window</td>
</tr>
<tr>
<td>&lt;window_name&gt;</td>
<td>list of the currently open windows; select a window name to switch to, or show that window if it is hidden; when the source window is available, the source file name is also indicated; open additional windows from the &quot;View menu&quot; (UM-156) in the Main window, or use the <code>view</code> command (CR-250)</td>
</tr>
</tbody>
</table>
Signals window

The Signals window is divided into two window panes. The left pane shows the names of HDL items in the current region (which is selected in the Structure window). The right pane shows the values of the associated HDL items at the end of the current run. The data in this pane is similar to that shown in the Wave window (UM-210), except that the values do not change dynamically with movement of the selected Wave window cursor.

You can double-click a signal and it will highlight that signal in the Source window (opening a Source window if one is not open already).

Horizontal scroll bars for each window pane allow scrolling to the right or left in each pane individually. The vertical scroll bar will scroll both panes together.

The HDL items can be sorted in ascending, descending, or declaration order.

HDL items you can view
One entry is created for each of the following VHDL and Verilog items within the design:

**VHDL items**
signals

**Verilog items**
 nets, register variables, named events, and module parameters

**Virtual items**
(indicated by an orange diamond icon)
virtual signals and virtual functions; see "Virtual signals" (UM-138) for more information

The names of any VHDL composite types (arrays and record types) are shown in a hierarchical fashion. Hierarchy also applies to Verilog nets and vector memories. (Verilog vector registers do not have hierarchy because they are not internally represented as arrays.)

Hierarchy is indicated in typical ModelSim fashion with plus (expandable), minus (expanded), and blank (single level) boxes.

See "Tree window hierarchical view" (UM-149) for more information.
### The Signals window menu bar

The following menu commands are available from the Signals window menu bar.

#### File menu

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save As</td>
<td>Save the signals tree to a text file viewable with the ModelSim notepad (CR-164)</td>
</tr>
<tr>
<td>Environment</td>
<td>Allow the window contents to change based on the current environment; or, fix to a specific context or dataset</td>
</tr>
<tr>
<td>Close</td>
<td>Close this copy of the Signals window; you can create a new window with View &gt; New from the &quot;The Main window menu bar&quot; (UM-154)</td>
</tr>
</tbody>
</table>

#### Edit menu

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>Copy the current selection in the Signals window</td>
</tr>
<tr>
<td>Sort</td>
<td>Sort the signals tree in either ascending, descending, or declaration order</td>
</tr>
<tr>
<td>Select All</td>
<td>Select all items in the Signals window</td>
</tr>
<tr>
<td>Unselect All</td>
<td>Unselect all items in the Signals window</td>
</tr>
<tr>
<td>Expand Selected</td>
<td>Expand the hierarchy of the selected items</td>
</tr>
<tr>
<td>Collapse Selected</td>
<td>Collapse the hierarchy of the selected items</td>
</tr>
<tr>
<td>Expand All</td>
<td>Expand the hierarchy of all items that can be expanded</td>
</tr>
<tr>
<td>Collapse All</td>
<td>Collapse the hierarchy of all expanded items</td>
</tr>
<tr>
<td>Force</td>
<td>Apply stimulus to the specified Signal Name; specify Value, Kind (Freeze/Drive/Deposit), Delay, and Cancel; see also the force command (CR-144)</td>
</tr>
<tr>
<td>Noforce</td>
<td>Remove the effect of any active force command (CR-144) on the selected HDL item; see also the noforce command (CR-161)</td>
</tr>
<tr>
<td>Clock</td>
<td>Define clock signals by Signal Name, Period, Duty Cycle, Offset, and whether the first edge is rising or falling, see “Defining clock signals” (UM-194)</td>
</tr>
<tr>
<td>Justify Values</td>
<td>Justify values to the left or right margins of the window pane</td>
</tr>
<tr>
<td>Find</td>
<td>Find the specified text string within the Signals window; choose the Name or Value field to search and the search direction: down or up; see also the search command (CR-201)</td>
</tr>
</tbody>
</table>
View menu

<table>
<thead>
<tr>
<th>Wave/List/Log</th>
<th>place the Selected Signals, Signals in Region, or Signals in Design in the Wave window (UM-210), List window (UM-170), or WLF file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter</td>
<td>choose the port and signal types to view (Input Ports, Output Ports, InOut Ports and Internal Signals) in the Signals window</td>
</tr>
</tbody>
</table>

Window menu

<table>
<thead>
<tr>
<th>Initial Layout</th>
<th>restore all windows to the size and placement of the initial full-screen layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cascade</td>
<td>cascade all open windows</td>
</tr>
<tr>
<td>Tile Horizontally</td>
<td>tile all open windows horizontally</td>
</tr>
<tr>
<td>Tile Vertically</td>
<td>tile all open windows vertically</td>
</tr>
<tr>
<td>Icon Children</td>
<td>icon all but the Main window</td>
</tr>
<tr>
<td>Icon All</td>
<td>icon all windows</td>
</tr>
<tr>
<td>Deicon All</td>
<td>deicon all windows</td>
</tr>
<tr>
<td>Customize</td>
<td>use the The Button Adder (UM-263) to define and add a button to either the tool or status bar of the specified window</td>
</tr>
<tr>
<td>&lt;window_name&gt;</td>
<td>list of the currently open windows; select a window name to switch to, or show that window if it is hidden; when the source window is available, the source file name is also indicated; open additional windows from the &quot;View menu&quot; (UM-156) in the Main window, or use the view command (CR-250)</td>
</tr>
</tbody>
</table>

Selecting HDL item types to view

The View > Filter menu selection allows you to specify which HDL items are shown in the Signals window. Multiple options can be selected.
Forcing signal and net values

The Edit > Force command displays a dialog box that allows you to apply stimulus to the selected signal or net. Multiple signals can be selected and forced; the force dialog box remains open until all of the signals are either forced, skipped, or you close the dialog box. To cancel a force command, use the Edit > NoForce command. See also the force command (CR-144).

The Force dialog box includes these options:

- **Signal Name**
  Specifies the signal or net for the applied stimulus.

- **Value**
  Initially displays the current value, which can be changed by entering a new value into the field. A value can be specified in radixes other than decimal by using the form (for VHDL and Verilog, respectively):
  
  \[ \text{base}\#\text{value -or- b|o|d|h'value} \]
  
  For example, \( 16\#EE \) or \( h'EE \), specifies the hexadecimal value EE.

- **Kind: Freeze**
  Freezes the signal or net at the specified value until it is forced again or until it is unforced with a noforce command (CR-161).
  
  **Freeze** is the default for Verilog nets and unresolved VHDL signals and **Drive** is the default for resolved signals.

  If you prefer **Freeze** as the default for resolved and unresolved signals, you can change the default force kind in the modelsim.ini file; see "Projects and system initialization" (UM-21).

- **Kind: Drive**
  Attaches a driver to the signal and drives the specified value until the signal or net is forced again or until it is unforced with a noforce command (CR-161). This value is illegal for unresolved VHDL signals.
• **Kind:** Deposit
  Sets the signal or net to the specified value. The value remains until there is a subsequent driver transaction, or until the signal or net is forced again, or until it is unforced with a `noforce` command (CR-161).

• **Delay For**
  Allows you to specify how many time units from the current time the stimulus is to be applied.

• **Cancel After**
  Cancels the `force` command (CR-144) after the specified period of simulation time.

• **OK**
  When you click the OK button, a `force` command (CR-144) is issued with the parameters you have set, and is echoed in the Main window. If more than one signal is selected to force, the next signal down appears in the dialog box each time the OK button is selected. Unique force parameters can be set for each signal.

### Adding HDL items to the Wave and List windows or a WLF file

Before adding items to the List or Wave window you may want to set the window display properties (see "Setting List window display properties" (UM-173)). Once display properties have been set, you can add items to the windows or WLF file in several ways.

#### Adding items with the Signals window View menu

Use the **View** menu with either the **Wave**, **List**, or **Log** selection to add HDL items to the **Wave window** (UM-210), **List window** (UM-170), or a wave log format (WLF) file, respectively.

The WLF file is written as an archive file in binary format and is used to drive the List and Wave windows at a later time. Once signals are added to the WLF file they cannot be removed. If you begin a simulation by invoking `vsim` (CR-284) with the `-view <WLF_fileame>` option, ModelSim reads the WLF file to drive the Wave and List windows.

Choose one of the following options (ModelSim opens the target window for you):

• **Selected Signal**
  Lists only the item(s) selected in the Signals window.

• **Signals in Region**
  Lists all items in the region that is selected in the Structure window.

• **Signals in Design**
  Lists all items in the design.
Adding items from the Main window command line

Another way to add items to the Wave or List window or the WLF file is to enter the one of the following commands at the VSIM prompt (choose either the add list (CR-49), add wave (CR-58), or log (CR-154) command):

```
add list | add wave | log <item_name> <item_name>
```

You can add all the items in the current region with this command:

```
add list | add wave | log *
```

Or add all the items in the design with:

```
add list | add wave | log -r */
```

If the target window (Wave or List) is closed, ModelSim opens it when you when you invoke the command.

Finding HDL items in the Signals window

To find the specified text string within the Signals window, choose the Name or Value field to search and the search direction: Down or Up.

Setting signal breakpoints

You can set signal breakpoints (a.k.a., when breakpoints; see the when command (CR-298) for more details) via a context menu in the Signal window. When statements instruct ModelSim to perform actions when the specified conditions are met. For example, you can break on a signal value or at a specific simulator time (see "Time-based breakpoints" (CR-300)). When a breakpoint is hit, a message appears in the transcript window about which signal caused the breakpoint.

To access the breakpoint commands, select a signal and then click your right mouse button (2nd button in Windows; 3rd button in UNIX). To set a breakpoint on a selected signal, select Add Breakpoint from the context menu. To remove a breakpoint from a selected signal, select Remove Signal Breakpoint. To remove all breakpoints in the current region, select Remove All Signal Breakpoints. To see a list of currently set breakpoints, select Show Breakpoints.
The **Edit Breakpoint** command opens the **Edit When** dialog box.

The **Edit When** dialog includes the following options:

- **Condition**
  The condition(s) to be met for the specified command(s) to be executed. Required. See the **when** command (CR-298) for more information on creating the condition statement.

- **Opt. Label**
  An optional text label for the when statement.

- **Command(s)**
  The command(s) to be executed when the specified condition is met. Any ModelSim or Tcl command or series of commands are valid with one exception—the **run** command (CR-199) cannot be used.

The **Edit All Breakpoints** command opens the Breakpoints dialog box. See "Setting file-line breakpoints" (UM-199) for details.
Defining clock signals

Select Edit > Clock to define clock signals by Name, Period, Duty Cycle, Offset, and whether the first rising edge is rising or falling. You can also specify a simulation period after which the clock definition should be cancelled.

For clock signals starting on the rising edge, the definition for Period, Offset, and Duty Cycle is as follows:

If the signal type is std_logic, std_ulogic, bit, verilog wire, verilog net, or any other logic type where 1 and 0 are valid, then 1 is the default High Value and 0 is the default Low Value. For other signal types, you will need to specify a High Value and a Low Value for the clock.
Source window

The Source window allows you to view and edit your HDL source code. When you first load a design, the source file will display automatically if the Source window is open. Alternatively, you can select an item in the Structure window (UM-204) or use the File > Open command (Source window) to add a file to the window. (Your source code can remain hidden if you wish; see "Source code security and -nodebug" (UM-420)).

The window is divided into two panes—the left-hand pane contains line numbers, and the right-hand pane contains the source file. The pathname of the source file is indicated in the header of the Source window.

As shown in the picture below, you may also see the following in the left-hand pane:

- Green line numbers—denote executable lines
- Blue arrow—denotes a process that you have selected in the Process window (UM-184)
- Red circles—denote file-line breakpoints; hollow circles denote breakpoints that are currently disabled

If you hold your mouse pointer over an HDL item in the right-hand pane, a “pop-up” will show you the full pathname of the item and its current value.
The Source window menu bar

The following menu commands are available from the Source window menu bar.

**File menu**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>edit a new (VHDL, Verilog or Other) source file</td>
</tr>
<tr>
<td>Open</td>
<td>select a source file to open</td>
</tr>
<tr>
<td>Use Source</td>
<td>specify an alternative file to use for the current source file; this</td>
</tr>
<tr>
<td></td>
<td>alternative source mapping exists for the current simulation only</td>
</tr>
<tr>
<td>Source Directory</td>
<td>add to a list of directories (the SourceDir variable in modelsim.tcl)</td>
</tr>
<tr>
<td></td>
<td>to search for source files</td>
</tr>
<tr>
<td>Properties</td>
<td>list a variety of information about the source file; for example, file</td>
</tr>
<tr>
<td></td>
<td>type, file size, file modification date</td>
</tr>
<tr>
<td>Save</td>
<td>save the current source file</td>
</tr>
<tr>
<td>Save As</td>
<td>save the current source file with a different name</td>
</tr>
<tr>
<td>Compile</td>
<td>compile HDL source files</td>
</tr>
<tr>
<td>Close</td>
<td>close this copy of the Source window; you can create a new window with</td>
</tr>
<tr>
<td></td>
<td>View &gt; New from the &quot;The Main window menu bar&quot; (UM-154)</td>
</tr>
</tbody>
</table>

**Edit menu**

To edit a source file, make sure the Read Only option in the Source Options dialog box is *not* selected (use the Edit > read only (Source menu) selection).

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;editing option&gt;</td>
<td>basic editing options include: Undo, Cut, Copy, Paste, Select All, and Unselect All; see &quot;The following mouse actions and special keystrokes can be used to edit commands in the entry region of the Main window. They can also be used in editing the file displayed in the Source window and all Notepad windows (enter the notepad command within ModelSim to open the Notepad editor).&quot; (UM-162)</td>
</tr>
<tr>
<td>Find</td>
<td>find the specified text string or regular expression within the source file;</td>
</tr>
<tr>
<td></td>
<td>there is an option to match case or search backwards</td>
</tr>
<tr>
<td>Find Next</td>
<td>find the next occurrence of a string specified with the Find command</td>
</tr>
<tr>
<td>Replace</td>
<td>find the specified text string or regular expression and replace it with</td>
</tr>
<tr>
<td></td>
<td>the specified text string or regular expression</td>
</tr>
<tr>
<td>Previous Coverage Miss</td>
<td>when simulating with Code Coverage (UM-285), finds the previous</td>
</tr>
<tr>
<td></td>
<td>line of code that was not used in the simulation</td>
</tr>
<tr>
<td><strong>Next Coverage Miss</strong></td>
<td>when simulating with <strong>Code Coverage</strong> (UM-285), finds the next line of code that was not used in the simulation</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Breakpoints</strong></td>
<td>add, edit, or delete file-line and signal breakpoints; see &quot;Setting file-line breakpoints&quot; (UM-199)</td>
</tr>
<tr>
<td><strong>read only</strong></td>
<td>toggle the read-only status of the current source file</td>
</tr>
</tbody>
</table>

**Object menu**

<table>
<thead>
<tr>
<th><strong>Describe</strong></th>
<th>display information about the selected HDL item; same as the <strong>describe</strong> command (CR-123); the item name is shown in the title bar</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Examine</strong></td>
<td>display the current value of the selected HDL item; same as the <strong>examine</strong> (CR-138) command; the item name is shown in the title bar</td>
</tr>
</tbody>
</table>

**Options menu**

<table>
<thead>
<tr>
<th><strong>Colorize Source</strong></th>
<th>colorize key words, variables, and comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Highlight Executable Lines</strong></td>
<td>highlight the line numbers of executable lines</td>
</tr>
<tr>
<td><strong>Middle Mouse Button Paste</strong></td>
<td>enable/disable pasting by pressing the middle-mouse button</td>
</tr>
<tr>
<td><strong>Verilog Highlighting</strong></td>
<td>specify Verilog-style colorizing</td>
</tr>
<tr>
<td><strong>VHDL Highlighting</strong></td>
<td>specify VHDL-style colorizing</td>
</tr>
<tr>
<td><strong>Freeze File</strong></td>
<td>maintain the same source file in the Source window (useful when you have two Source windows open; one can be updated from the <strong>Structure window</strong> (UM-204), the other frozen)</td>
</tr>
<tr>
<td><strong>Freeze View</strong></td>
<td>disable updating the source view from the <strong>Process window</strong> (UM-184)</td>
</tr>
</tbody>
</table>

**Window menu**

<table>
<thead>
<tr>
<th><strong>Initial Layout</strong></th>
<th>restore all windows to the size and placement of the initial full-screen layout</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cascade</strong></td>
<td>cascade all open windows</td>
</tr>
<tr>
<td><strong>Tile Horizontally</strong></td>
<td>tile all open windows horizontally</td>
</tr>
<tr>
<td><strong>Tile Vertically</strong></td>
<td>tile all open windows vertically</td>
</tr>
<tr>
<td><strong>Icon Children</strong></td>
<td>icon all but the Main window</td>
</tr>
</tbody>
</table>
The Source window toolbar

Buttons on the Source window toolbar give you quick access to these ModelSim commands and functions.

<table>
<thead>
<tr>
<th>Icon All</th>
<th>icon all windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deicon All</td>
<td>deicon all windows</td>
</tr>
<tr>
<td>Customize</td>
<td>use the The Button Adder (UM-263) to define and add a button to either the tool or status bar of the specified window</td>
</tr>
<tr>
<td>&lt;window_name&gt;</td>
<td>list of the currently open windows; select a window name to switch to, or show that window if it is hidden; when the source window is available, the source file name is also indicated; open additional windows from the &quot;View menu&quot; (UM-156) in the Main window, or use the view command (CR-250)</td>
</tr>
</tbody>
</table>

### Source window toolbar buttons

<table>
<thead>
<tr>
<th>Button</th>
<th>Menu equivalent</th>
<th>Other equivalents</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Compile Source File" /></td>
<td>File &gt; Compile</td>
<td>use vcom or vlog command at the VSIM prompt</td>
</tr>
<tr>
<td><img src="image" alt="Open Source File" /></td>
<td>File &gt; Open</td>
<td>select an HDL item in the Structure window, the associated source file is loaded into the Source window</td>
</tr>
<tr>
<td><img src="image" alt="Save Source File" /></td>
<td>File &gt; Save</td>
<td>none</td>
</tr>
</tbody>
</table>
You can set breakpoints three different ways:

- Using the command line; see the `bp` (CR-69) (breakpoint) command for details
- Using your mouse in the Source window
- Using the `Edit > Breakpoints` menu selection

### Setting breakpoints with your mouse

To set a breakpoint with your mouse, click on a green line number at the left side of the window (breakpoints can be set only on executable lines). The breakpoints are toggles – click once to create the colored dot; click again to disable or enable the breakpoint. To delete the breakpoint completely, click the colored dot with your right mouse button, and select **Remove Breakpoint**.
Setting breakpoints with the Edit > Breakpoints command

Selecting Edit > Breakpoints (Source window) opens the dialog box shown below.

The Breakpoints dialog box allows you to create and manage both file-line and signal breakpoints (a.k.a., when breakpoints). For details on signal breakpoints, see "Setting signal breakpoints" (UM-192) and the when command (CR-298).
You can enable and disable existing breakpoints by checking or unchecking the box next to the breakpoint’s name. To add a new file-line breakpoint, select Add BP (or Edit Selected for an existing file-line breakpoint).

The Add/Edit Breakpoint dialog box includes the following options:

- **File Name**
  The file name in which you want to set the breakpoint. Required. The button next to this field allows you to browse to select a file.

- **Line #**
  The line number on which you want to set the breakpoint. Required.

- **Condition**
  The condition(s) that determine whether the breakpoint is hit. See the `bp` command (CR-69) for more information on creating the condition statement.

- **Instance**
  Specify a region in which the breakpoint should be set. If left blank, the breakpoint affects every instance in the design.

- **Command(s)**
  One or more commands that you want executed at the breakpoint.
Editing the source file in the Source window

Several toolbar buttons (shown above), mouse actions, and special keystrokes can be used to edit the source file in the Source window. See "The following mouse actions and special keystrokes can be used to edit commands in the entry region of the Main window. They can also be used in editing the file displayed in the Source window and all Notepad windows (enter the notepad command within ModelSim to open the Notepad editor)." (UM-162) for a list of mouse and keyboard editing options.

Checking HDL item values and descriptions

There are two quick methods to determine the value and description of an HDL item displayed in the Source window:

- select an item, then chose Object > Examine or Object > Description from the Source window menu
- select an item with the right mouse button to see an examine pop-up (select "now" to examine the current simulation time in VHDL code)

You can also invoke the examine (CR-138) and/or describe (CR-123) command on the command line or in a macro.

Finding and replacing in the Source window

The Find dialog box allows you to find and replace text strings or regular expressions in the Source window. Select Edit > Find or Edit > Replace to bring up the Find dialog box. If you select Edit > Find, the Replace field is absent from the dialog.

Enter the value to search for in the Find field. If you are doing a replace, enter the appropriate value in the Replace field. Optionally specify whether the entries are case sensitive and whether to search backwards from the current cursor location. Check the Regular expression checkbox if you are using regular expressions.
Setting tab stops in the Source window

You can set tab stops in the Source window by selecting the Main window Options > Edit Preferences command. Follow these steps:

1. Select the By Names tab.

2. Select Source in the first column, and then select the "tabs" item in the second column.

3. Press the Change Value button.

4. In the dialog that appears, enter a single number "n", which sets a tab stop every n characters (where a character width is the width of the "8" character).

   or

Enter a list of screen distances for the tab stops. For instance,

21 49 77 105 133 161 189 217 245 273 301 329 357 385 413 441 469

The number 21 or 21p means 21 pixels; the number 3c means three centimeters; the number 1i means one inch.

⚠️ Important: Do not use quotes or braces in the list (i.e., "21 49" or {21 49}). This will cause the GUI to hang.

You can also set tab stops using the PrefSource(tabs) Tcl preference variable.
Structure window

Note: In ModelSim versions 5.5 and later the information contained in the Structure window is shown in the structure tabs of the Main window Workspace (UM-152). The Structure window will not display by default. You can display the Structure window at any time by selecting View > Structure (Main window). The discussion below applies to both the Structure window and the structure tabs in the Workspace.

The Structure window provides a hierarchical view of the structure of your design. An entry is created by each HDL item within the design. (Your design structure can remain hidden if you wish, see "Source code security and -nodebug" (UM-420).)

HDL items you can view

The following HDL items for VHDL and Verilog are represented by hierarchy within Structure window.

VHDL items
(indicated by a dark blue square icon)
component instantiation, generate statements, block statements, and packages

Verilog items
(indicated by a lighter blue circle icon)
module instantiations, named forks, named begins, tasks, and functions

Virtual items
(indicated by an orange diamond icon)
virtual regions; see "Virtual Objects (User-defined buses, and more)" (UM-138) for more information.

You can expand and contract the display to view the hierarchical structure by clicking on the boxes that contain "+" or "+". Clicking "+" expands the hierarchy so the sub-elements of that item can be seen. Clicking "-" contracts the hierarchy.

The first line of the Structure window indicates the top-level design unit being simulated. By default, this is the only level of the hierarchy that is expanded upon opening the Structure window.
Instance name components in the Structure window

An instance name displayed in the Structure window consists of the following parts:

- **instantiation label**
  Indicates the label assigned to the component or module instance in the instantiation statement.

- **entity or module**
  Indicates the name of the entity or module that has been instantiated.

- **architecture**
  Indicates the name of the architecture associated with the entity (not present for Verilog).

When you select a region in the Structure window, it becomes the current region and is highlighted; the Source window (UM-195) and Signals window (UM-187) change dynamically to reflect the information for that region. This feature provides a useful method for finding the source code for a selected region because the system keeps track of the pathname where the source is located and displays it automatically, without the need for you to provide the pathname.

Also, when you select a region in the Structure window, the Process window (UM-184) is updated if In Region is selected in that window; the Process window will in turn update the Variables window (UM-207).

The Structure window menu bar

The following menu commands are available from the Structure window menu bar. Some of the commands are also available from a context menu in a Structure tab of the Main window workspace.

### File menu

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save As</td>
<td>save the structure tree to a text file viewable with the ModelSim notepad (CR-164)</td>
</tr>
<tr>
<td>Environment</td>
<td>allow the window contents to change when the active dataset is changed; or, fix to a specific dataset</td>
</tr>
<tr>
<td>Close</td>
<td>close this copy of the Structure window; you can create a new window with View &gt; New from the &quot;The Main window menu bar&quot; (UM-154)</td>
</tr>
</tbody>
</table>

### Edit menu

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>copy the current selection in the Structure window</td>
</tr>
<tr>
<td>Sort</td>
<td>sort the structure tree in either ascending, descending, or declaration order</td>
</tr>
<tr>
<td>Expand Selected</td>
<td>expand the hierarchy of the selected item</td>
</tr>
</tbody>
</table>
Finding items in the Structure window

The Find dialog box allows you to search for text strings in the Structure window. Select **Edit > Find** (Structure window) to bring up the Find dialog box.

Enter the value to search for in the **Find** field. Specify whether you are looking for an **Instance**, **Entity/Module**, or **Architecture**. Also specify which direction to search. Check **Auto Wrap** to have the search continue at the top of the window.

---

<table>
<thead>
<tr>
<th>Collapse Selected</th>
<th>collapse the hierarchy of the selected item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expand All</td>
<td>expand the hierarchy of all items that can be expanded</td>
</tr>
<tr>
<td>Collapse All</td>
<td>collapse the hierarchy of all expanded items</td>
</tr>
<tr>
<td>Find</td>
<td>find the specified text string within the structure tree; see &quot;Finding items in the Structure window&quot; (UM-206)</td>
</tr>
</tbody>
</table>

**Window menu**

<table>
<thead>
<tr>
<th>Initial Layout</th>
<th>restore all windows to the size and placement of the initial full-screen layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cascade</td>
<td>cascade all open windows</td>
</tr>
<tr>
<td>Tile Horizontally</td>
<td>tile all open windows horizontally</td>
</tr>
<tr>
<td>Tile Vertically</td>
<td>tile all open windows vertically</td>
</tr>
<tr>
<td>Icon Children</td>
<td>icon all but the Main window</td>
</tr>
<tr>
<td>Icon All</td>
<td>icon all windows</td>
</tr>
<tr>
<td>Deicon All</td>
<td>deicon all windows</td>
</tr>
<tr>
<td>Customize</td>
<td>use the <strong>The Button Adder</strong> (UM-263) to define and add a button to either the tool or status bar of the specified window</td>
</tr>
<tr>
<td>&lt;window_name&gt;</td>
<td>list of the currently open windows; select a window name to switch to, or show that window if it is hidden; when the source window is available, the source file name is also indicated; open additional windows from the &quot;View menu&quot; (UM-156) in the Main window, or use the <strong>view</strong> command (CR-250)</td>
</tr>
</tbody>
</table>
Variables window

The Variables window is divided into two window panes. The left pane lists the names of HDL items within the current process. The right pane lists the current value(s) associated with each name. The pathname of the current process is displayed at the bottom of the window. (The internal variables of your design can remain hidden if you wish, see "Source code security and -nodebug" (UM-420).)

HDL items you can view

The following HDL items for VHDL and Verilog are viewable within the Variables window.

**VHDL items**

- constants, generics, and variables

**Verilog items**

- register variables

The names of any VHDL composite types (arrays and record types) are shown in a hierarchical fashion. Hierarchy also applies to Verilog vector memories. (Verilog vector registers do not have hierarchy because they are not internally represented as arrays.) Hierarchy is indicated in typical ModelSim fashion with plus (expandable) and minus (expanded). See "Tree window hierarchical view" (UM-149) for more information.

To change the value of a VHDL variable, constant, or generic or a Verilog register variable, move the pointer to the desired name and click to highlight the selection. Select **Edit > Change** (Variables window) to bring up a dialog box that lets you specify a new value. Note that "Variable Name" is a term that is used loosely in this case to signify VHDL constants and generics as well as VHDL and Verilog register variables. You can enter any value that is valid for the variable. An array value must be specified as a string (without surrounding quotation marks). To modify the values in a record, you need to change each field separately.

Click on a process in the Process window to change the Variables window.
The Variables window menu bar

The following menu commands are available from the Variables window menu bar.

### File menu

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save As</td>
<td>save the variables tree to a text file viewable with the ModelSim notepad (CR-164)</td>
</tr>
<tr>
<td>Environment</td>
<td><strong>Follow Process Selection</strong>: update the window based on the selection in the Process window (UM-184); <strong>Fix to Current Process</strong>: maintain the current view, do not update</td>
</tr>
<tr>
<td>Close</td>
<td>close this copy of the Variables window; you can create a new window with View &gt; New from the &quot;The Main window menu bar&quot; (UM-154)</td>
</tr>
</tbody>
</table>

### Edit menu

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>copy the selected items in the Variables window</td>
</tr>
<tr>
<td>Sort</td>
<td>sort the variables tree in either ascending, descending, or declaration order</td>
</tr>
<tr>
<td>Select All</td>
<td>select all items in the Variables window</td>
</tr>
<tr>
<td>Unselect All</td>
<td>deselect all items in the Variables window</td>
</tr>
<tr>
<td>Expand Selected</td>
<td>expand the hierarchy of the selected item</td>
</tr>
<tr>
<td>Collapse Selected</td>
<td>collapse the hierarchy of the selected item</td>
</tr>
<tr>
<td>Expand All</td>
<td>expand the hierarchy of all items that can be expanded</td>
</tr>
<tr>
<td>Collapse All</td>
<td>collapse the hierarchy of all expanded items</td>
</tr>
<tr>
<td>Change</td>
<td>change the value of the selected HDL item</td>
</tr>
<tr>
<td>Justify Values</td>
<td>justify values to the left or right margins of the window pane</td>
</tr>
<tr>
<td>Find</td>
<td>find the specified text string within the variables tree; choose the Name or Value field to search and the search direction: Down or Up</td>
</tr>
</tbody>
</table>

### View menu

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wave/List/Log</td>
<td>place the Selected Variables or Variables in Region in the Wave window (UM-210), List window (UM-170), or WLF file</td>
</tr>
</tbody>
</table>
### Window menu

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Layout</td>
<td>restore all windows to the size and placement of the initial full-screen layout</td>
</tr>
<tr>
<td>Cascade</td>
<td>cascade all open windows</td>
</tr>
<tr>
<td>Tile Horizontally</td>
<td>tile all open windows horizontally</td>
</tr>
<tr>
<td>Tile Vertically</td>
<td>tile all open windows vertically</td>
</tr>
<tr>
<td>Icon Children</td>
<td>icon all but the Main window</td>
</tr>
<tr>
<td>Icon All</td>
<td>icon all windows</td>
</tr>
<tr>
<td>Deicon All</td>
<td>deicon all windows</td>
</tr>
<tr>
<td>Customize</td>
<td>use the The Button Adder (UM-263) to define and add a button to either the tool or status bar of the specified window</td>
</tr>
<tr>
<td>&lt;window_name&gt;</td>
<td>list of the currently open windows; select a window name to switch to, or show that window if it is hidden; when the source window is available, the source file name is also indicated; open additional windows from the &quot;View menu&quot; (UM-156) in the Main window, or use the <code>view</code> command (CR-250)</td>
</tr>
</tbody>
</table>
Wave window

The Wave window, like the List window, allows you to view the results of your simulation. In the Wave window, however, you can see the results as HDL waveforms and their values.

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

Pathname pane

The pathname pane displays signal pathnames. Signals can be displayed with full pathnames, as shown here, or with only the leaf element displayed. You can increase the size of the pane by clicking and dragging on the right border. The selected signal is highlighted.

The white bar along the left margin indicates the selected dataset (see Splitting Wave window panes (UM-222)).
Values pane

A values pane displays the values of the displayed signals.

The radix for each signal can be symbolic, binary, octal, decimal, unsigned, hexadecimal, ASCII, or default. The default radix can be set by selecting Options > Simulation (Main window) (see "Setting default simulation options" (UM-259)).

The data in this pane is similar to that shown in the Signals window (UM-187), except that the values change dynamically whenever a cursor in the waveform pane (below) is moved.

Waveform pane

The waveform pane displays the waveforms that correspond to the displayed signal pathnames. It also displays up to 20 cursors. Signal values can be displayed in analog step, analog interpolated, analog backstep, literal, logic, and event formats. Each signal can be formatted individually. The default format is logic.

If you rest your mouse pointer on a signal in the waveform pane, a popup displays with information about the signal. You can toggle this popup on and off in the Wave Window Properties dialog (see "Setting Wave window display properties" (UM-229)).
Cursor panes

There are two cursor panes. The left pane shows the current simulation time and the value for each cursor. The top-most value is the current simulation time. The selected cursor’s value is highlighted. You can select a cursor by selecting its value in the left pane.

The right pane shows the absolute time value for each cursor and relative time between cursors. Up to 20 cursors can be displayed.

HDL items you can view

VHDL items
(indicated by a dark blue square)
signals and process variables

Verilog items
(indicated by a light blue circle)
nets, register variables, and named events

Virtual items
(indicated by an orange diamond)
virtual signals, buses, and functions, see; "Virtual Objects (User-defined buses, and more)" (UM-138) for more information

Comparison items
(indicated by a yellow triangle)
comparison region and comparison signals; see Chapter 11 - Waveform Comparison for more information

Note: Constants, generics, and parameters are not viewable in the List or Wave windows.

The data in the item values pane is very similar to the Signals window, except that the values change dynamically whenever a cursor in the waveform pane is moved.

At the bottom of the waveform pane you can see a time line, tick marks, and a readout of each cursor’s position. As you click and drag to move a cursor, the time value at the cursor location is updated at the bottom of the cursor.
You can resize the window panes by clicking on the bar between them and dragging the bar to a new location.

Waveform and signal-name formatting are easily changed via the Format menu (UM-217). You can reuse any formatting changes you make by saving a Wave window format file, see "Adding items with a Wave window format file" (UM-213).

### Adding HDL items in the Wave window

Before adding items to the Wave window you may want to set the window display properties (see "Setting Wave window display properties" (UM-229)). You can add items to the Wave window in several ways.

#### Adding items from the Signals window with drag and drop

You can drag and drop items into the Wave window from the List, Process, Signals, Source, Structure, or Variables window. Select the items in the first window, then drop them into the Wave window. Depending on what you select, all items or any portion of the design can be added.

#### Adding items from the Main window command line

To add specific HDL items to the window, enter (separate the item names with a space):

```
add wave <item_name> <item_name>
```

You can add all the items in the current region with this command:

```
add wave *
```

Or add all the items in the design with:

```
add wave -r /*
```

#### Adding items with a Wave window format file

To use a Wave window format file you must first save a format file for the design you are simulating. Follow these steps:

1. Add the items you want in the Wave window with any method shown above.
2. Edit and format the items, see "Editing and formatting HDL items in the Wave window" (UM-224) to create the view you want.
3. Save the format to a file by selecting File > Save Format (Wave window).

To use the format file, start with a blank Wave window and run the DO file in one of two ways:

- Invoke the do command (CR-127) from the command line:
  
  ```
do <my_wave_format>
  ```

- Select File > Load Format (Wave window).
Use **Edit > Select All** and **Edit > Delete** to remove the items from the current Wave window, use the delete command (CR-122) with the wave option, or create a new, blank Wave window with **View > New > Wave** (Main window).

**Note:** Wave window format files are design-specific; use them only with the design you were simulating when they were created.

### The Wave window menu bar

![Wave window menu bar](image)

The following menu commands and button options are available from the Wave window menu bar. If you see a dotted line at the top of a drop-down menu, you can select it to create a separate menu window. Many of these commands are also available via a context menu by clicking your right mouse button within the wave window itself.

#### File menu

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Dataset</td>
<td>open a dataset</td>
</tr>
<tr>
<td>New Divider</td>
<td>insert a divider at the current location</td>
</tr>
<tr>
<td>New Group</td>
<td>setup a new group element – a container for other items that can be moved, cut and pasted like other objects (NOT CURRENTLY IMPLEMENTED)</td>
</tr>
<tr>
<td>Save Format</td>
<td>save the current Wave window display and signal preferences to a DO (macro) file; running the DO file will reformat the Wave window to match the display as it appeared when the DO file was created</td>
</tr>
<tr>
<td>Load Format</td>
<td>run a Wave window format (DO) file previously saved with Save Format</td>
</tr>
<tr>
<td>Page Setup</td>
<td>setup page for printing; options include: paper size, margins, label width, cursors, color, scaling and orientation</td>
</tr>
<tr>
<td>Print (Windows only)</td>
<td>send the contents of the Wave window to a selected printer; options include: All signals – print all signals Current View – print signals in current view for the time displayed Selected – print all or current view signals for user-designated time</td>
</tr>
</tbody>
</table>
| **Print Postscript** | save or print the waveform display as a Postscript file; options include:  
All Signals – print all signals  
Current View – print signals in current view for the time displayed  
Selected – print all or current view signals for user-designated time |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>New Window Pane</strong></td>
<td>split the pathname, values and waveform window panes to provide room for a new waveset</td>
</tr>
<tr>
<td><strong>Remove Window Pane</strong></td>
<td>remove window split and active waveset</td>
</tr>
<tr>
<td><strong>Refresh Display</strong></td>
<td>clear the Wave window, empty the file cache, and rebuild the window from scratch</td>
</tr>
<tr>
<td><strong>Close</strong></td>
<td>close this copy of the Wave window; you can create a new window with View &gt; New from ”The Main window menu bar” (UM-154)</td>
</tr>
</tbody>
</table>

**Edit menu**

<table>
<thead>
<tr>
<th><strong>Cut</strong></th>
<th>cut the selected item and waveform from the Wave window; see ”Editing and formatting HDL items in the Wave window” (UM-224)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Copy</strong></td>
<td>copy the selected item and waveform</td>
</tr>
<tr>
<td><strong>Paste</strong></td>
<td>paste the previously cut or copied item above the currently selected item</td>
</tr>
<tr>
<td><strong>Delete</strong></td>
<td>delete the selected item and its waveform</td>
</tr>
<tr>
<td><strong>Select All</strong></td>
<td>select, or unselect, all item names in the name pane</td>
</tr>
<tr>
<td><strong>Unselect All</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Combine</strong></td>
<td>combine the selected fields into a user-defined bus</td>
</tr>
<tr>
<td><strong>Signal Breakpoints</strong></td>
<td>add, edit, and delete signal breakpoints; see ”Setting signal breakpoints” (UM-192)</td>
</tr>
<tr>
<td><strong>Sort</strong></td>
<td>sort the top-level items in the name pane; sort with full path name or viewed name; use ascending or descending order</td>
</tr>
<tr>
<td><strong>Find</strong></td>
<td>find the specified item label within the pathname pane or the specified value within the value pane</td>
</tr>
<tr>
<td><strong>Search</strong></td>
<td>search the waveform display for a specified value, or the next transition for the selected signal; see: ”Searching for item values in the Wave window” (UM-231)</td>
</tr>
<tr>
<td><strong>Justify Values</strong></td>
<td>justify values to the left or right margins of the window pane</td>
</tr>
<tr>
<td><strong>Display Properties</strong></td>
<td>set display properties for signal path length, cursor snap distance, row margin, and dataset prefixes</td>
</tr>
</tbody>
</table>
### Signal Properties
- Set label, height, color, radix, and format for the selected item (use the Format menu selections below to quickly change individual properties); also set properties related to waveform comparisons.

### Cursor menu
- **Add Cursor**: add a cursor to the center of the waveform window.
- **Delete Cursor**: delete the selected cursor from the window.
- **Goto**: choose a cursor to go to from a list of current cursors.

### Zoom menu
- **Zoom <selection>**
  - Selection: Full, In, Out, Last, Area with mouse button 1, or Range to change the waveform display range.

### Compare menu
- **Start Comparison**: start a new comparison.
- **Comparison Wizard**: receive step-by-step assistance while creating a waveform comparison.
- **Run Comparison**: compute differences from time zero until the end of the simulation.
- **End Comparison**: stop difference computation and close the currently open comparison.
- **Add**: provides three options:
  - Compare by Signal - specify signals for comparison
  - Compare by Region - designate a reference region for a comparison
  - Clocks - define clocks to be used in a comparison
- **Options**: set options for waveform comparisons.
- **Differences**: provides four options:
  - Clear - clear all differences from the Wave window
  - Show - display differences in a text format in the Main window Transcript
  - Save - save computation differences to a file that can be reloaded later
  - Write Report - save computation differences to a text file
- **Rules**: provides two options:
  - Show - display the rules used to set up the waveform comparison
  - Save - save rules for waveform comparison to a file
<table>
<thead>
<tr>
<th>Reload</th>
<th>load saved differences and rules files</th>
</tr>
</thead>
</table>

**Bookmark menu**

<table>
<thead>
<tr>
<th>Add Bookmark</th>
<th>add a new bookmark that saves a specific zoom and scroll range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit Bookmarks</td>
<td>edit an existing bookmark</td>
</tr>
<tr>
<td>&lt;bookmark_name&gt;</td>
<td>list of currently defined bookmarks</td>
</tr>
</tbody>
</table>

**Format menu**

<table>
<thead>
<tr>
<th>Radix</th>
<th>set the selected item’s radix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>set the waveform format for the selected item – Literal, Logic, Event, Analog</td>
</tr>
<tr>
<td>Color</td>
<td>set the color for the selected item from a color palette</td>
</tr>
<tr>
<td>Height</td>
<td>set the waveform height in pixels for the selected item</td>
</tr>
</tbody>
</table>

**Window menu**

<table>
<thead>
<tr>
<th>Initial Layout</th>
<th>restore all windows to the size and placement of the initial full-screen layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cascade</td>
<td>cascade all open windows</td>
</tr>
<tr>
<td>Tile Horizontally</td>
<td>tile all open windows horizontally</td>
</tr>
<tr>
<td>Tile Vertically</td>
<td>tile all open windows vertically</td>
</tr>
<tr>
<td>Icon Children</td>
<td>icon all but the Main window</td>
</tr>
<tr>
<td>Icon All</td>
<td>icon all windows</td>
</tr>
<tr>
<td>Deicon All</td>
<td>deicon all windows</td>
</tr>
<tr>
<td>Customize</td>
<td>use the The Button Adder (UM-263) to define and add a button to either the tool or status bar of the specified window</td>
</tr>
<tr>
<td>&lt;window_name&gt;</td>
<td>list of the currently open windows; select a window name to switch to, or show that window if it is hidden; when the source window is available, the source file name is also indicated; open additional windows from the &quot;View menu&quot; (UM-156) in the Main window, or use the view command (CR-250)</td>
</tr>
</tbody>
</table>
The Wave window toolbar

The Wave window toolbar gives you quick access to these ModelSim commands and functions.

Wave window toolbar buttons

<table>
<thead>
<tr>
<th>Button</th>
<th>Menu equivalent</th>
<th>Other options</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Load Wave Format" /></td>
<td>File &gt; Load Format</td>
<td>do wave.do</td>
</tr>
<tr>
<td><img src="image" alt="Save Wave Format" /></td>
<td>File &gt; Save Format</td>
<td>see do command (CR-127)</td>
</tr>
<tr>
<td><img src="image" alt="Print Waveform" /></td>
<td>File &gt; Print</td>
<td>none</td>
</tr>
<tr>
<td><img src="image" alt="Print Waveform" /></td>
<td>File &gt; PrintPostscript</td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="Cut" /></td>
<td>Edit &gt; Cut</td>
<td>right mouse in pathname pane &gt; Cut</td>
</tr>
<tr>
<td><img src="image" alt="Copy" /></td>
<td>Edit &gt; Copy</td>
<td>right mouse in pathname pane &gt; Copy</td>
</tr>
<tr>
<td>Wave window toolbar buttons</td>
<td>Menu equivalent</td>
<td>Other options</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------</td>
<td>---------------</td>
</tr>
<tr>
<td><strong>Paste</strong></td>
<td>Edit &gt; Paste</td>
<td>right mouse in pathname pane &gt; Paste</td>
</tr>
<tr>
<td><strong>Add Cursor</strong></td>
<td>Cursor &gt; Add Cursor</td>
<td>none</td>
</tr>
<tr>
<td><strong>Delete Cursor</strong></td>
<td>Cursor &gt; Delete Cursor</td>
<td>none</td>
</tr>
<tr>
<td><strong>Find Previous Transition</strong></td>
<td>Edit &gt; Search</td>
<td>keyboard: Shift + Tab (Search Reverse)</td>
</tr>
<tr>
<td></td>
<td>(Search Reverse)</td>
<td>left &lt;arguments&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>see left command (CR-152)</td>
</tr>
<tr>
<td><strong>Find Next Transition</strong></td>
<td>Edit &gt; Search</td>
<td>keyboard: Tab (Search Forward)</td>
</tr>
<tr>
<td></td>
<td>(Search Forward)</td>
<td>right &lt;arguments&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>see right command (CR-197)</td>
</tr>
<tr>
<td><strong>Zoom in 2x</strong></td>
<td>Zoom &gt; Zoom In</td>
<td>keyboard: i I or +</td>
</tr>
<tr>
<td></td>
<td></td>
<td>right mouse in wave pane &gt; Zoom In</td>
</tr>
<tr>
<td><strong>Zoom out 2x</strong></td>
<td>Zoom &gt; Zoom Out</td>
<td>keyboard: o O or -</td>
</tr>
<tr>
<td></td>
<td></td>
<td>right mouse in wave pane &gt; Zoom Out</td>
</tr>
<tr>
<td><strong>Zoom area with mouse button 1</strong></td>
<td>Zoom &gt; Zoom Range</td>
<td>keyboard: r or R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>right mouse in wave pane &gt; Zoom Area</td>
</tr>
<tr>
<td><strong>Zoom Full</strong></td>
<td>Zoom &gt; Zoom Full</td>
<td>keyboard: f or F</td>
</tr>
<tr>
<td></td>
<td></td>
<td>right mouse in wave pane &gt; Zoom Full</td>
</tr>
<tr>
<td><strong>Interrupt Wave Drawing</strong></td>
<td>none</td>
<td>.wave.tree interrupt</td>
</tr>
</tbody>
</table>
### Wave window toolbar buttons

<table>
<thead>
<tr>
<th>Button</th>
<th>Menu equivalent</th>
<th>Other options</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Restart</strong></td>
<td>Main menu: Run &gt; Restart</td>
<td>restart &lt;arguments&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>see: restart (CR-193)</td>
</tr>
<tr>
<td><strong>Run</strong></td>
<td>Main menu: Run &gt; Run &lt;default_length&gt;</td>
<td>use the run command at the VSIM prompt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>see: run (CR-199)</td>
</tr>
<tr>
<td><strong>Continue Run</strong></td>
<td>Main menu: Run &gt; Continue</td>
<td>use the run -continue command at the VSIM prompt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>see: run (CR-199)</td>
</tr>
<tr>
<td><strong>Run -All</strong></td>
<td>Main menu: Run &gt; Run -All</td>
<td>use run -all command at the VSIM prompt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>see: run (CR-199), also see &quot;Assertion settings tab&quot; (UM-260)</td>
</tr>
<tr>
<td><strong>Break</strong></td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td><strong>Find First Difference</strong></td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td><strong>Find Previous Difference</strong></td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td><strong>Find Next Difference</strong></td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td><strong>Find Last Difference</strong></td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>
Using Dividers

Dividing lines can be placed in the pathname and values window panes by selecting **File > New Divider** (Wave window). Dividers serve as a visual aid to signal debugging, allowing you to separate signals and waveforms for easier viewing.

Dividing lines can be assigned any name, or no name at all. The default name is "New Divider." In the illustration below, two datasets have been separated with a Divider called "Gold." Notice that the waveforms in the waveform window pane have been separated by the divider as well.

After you have added a divider, you can move it, change its properties (name and size), or delete it.

**To move a divider** — Click and drag the divider to the location you want

**To change a divider's name and size** — Click the divider with the right (Windows) or third (UNIX) mouse button and select Divider Properties from the pop-up menu

**To delete a divider** — Select the divider and either press the <Delete> key on your keyboard or select Delete from the pop-up menu
Splitting Wave window panes

The pathnames, values and waveforms window panes of the Wave window display can be split to accommodate signals from one or more datasets. Selecting **File > New Window Pane** (Wave window) creates a space below the selected waveset and makes the new window pane the selected pane. (The selected wave window pane is indicated by a white bar along the left margin of the pane.)

In the illustration below, the Wave window is split, showing the current active simulation with the prefix "sim," and a second view-mode dataset, with the prefix "gold."

For more information on viewing multiple simulations, see *Chapter 7 - WLF files (datasets) and virtuals.*
Combining items in the Wave window

You can combine signals in the Wave window into busses. A bus is a collection of signals concatenated in a specific order to create a new virtual signal with a specific value. To create a bus, select one or more signals in the Wave window and then choose Edit > Combine. 

The Combine Selected Signals dialog box includes these options:

- **Combine Into**
  
  Only the Bus option is valid at this time. Groups are not currently implemented.

- **Order of Indexes**
  
  Specifies in which order the selected signals are indexed in the bus. If set to Ascending, the first signal selected in the Wave window will be assigned an index of 0. If set to Descending, the first signal selected will be assigned the highest index number.

- **Remove selected signals after combining**
  
  Specifies whether you want to remove the selected signals from the Wave window once the bus is created.

In the illustration below, three signals have been combined to form a new bus called BUS1. Note that the component signals are listed in the order in which they were selected in the Wave window. Also note that the bus’ value is made up of the values of its component signals arranged in a specific order. Virtual objects are indicated by an orange diamond.
Other virtual items in the Wave window

See "Virtual Objects (User-defined buses, and more)" (UM-138) for information about other virtual items viewable in the Wave window.

Editing and formatting HDL items in the Wave window

Once you have the HDL items you want in the Wave window, you can edit and format the list in the pathname and values panes to create the view you find most useful. (See also, "Setting Wave window display properties" (UM-229).)

To edit an item:

Select the item’s label in the pathname pane or its waveform in the waveform pane. Move, copy, or remove the item by selecting commands from the Wave window Edit menu (UM-215).

You can also click+drag to move items within the pathnames and values panes:

- to select several items:
  control+click to add or subtract from the selected group
- to move the selected items:
  re-click and hold on one of the selected items, then drag to the new location
To format an item:

Select the item’s label in the pathname pane or its waveform in the waveform pane, then select Edit > Signal Properties (Wave window). The resulting Wave Signal Properties dialog box has three tabs: View, Format, and Compare.

The View tab includes these options:

- **Display Name**
  Specifies a new name (in the pathname pane) for the selected signal.

- **Radix**
  Specifies the Radix of the selected signal(s). Setting this to default causes the signal’s radix to change whenever the default is modified using the `radix` command (CR-189). Item values are not translated if you select Symbolic.

- **Wave Color**
  Specifies the waveform color. Select a new color from the color palette, or enter an X-Windows color name.

- **Name Color**
  Specifies the signal name’s color. Select a new color from the color palette, or enter an X-Windows color name.
The **Format** tab includes these options:

- **Format: Literal**
  Displays the waveform as a box containing the item value (if the value fits the space available). This is the only format that can be used to list a record.

- **Format: Logic**
  Displays values as U, X, 0, 1, Z, W, L, H, or -.

- **Format: Event**
  Marks each transition during the simulation run.
• **Format: Analog [Step | Interpolated | Backstep]**

  All signals in the following illustration are the same /top/clk signal. Starting with "analog step", the /top/clk signal has been relabeled to illustrate each different wave format.

  **Analog Step**
  Displays a waveform in step style.

  **Analog Interpolated**
  Displays the waveform in interpolated style.

  **Analog Backstep**
  Displays the waveform in backstep style. Often used for power calculations.

  **Offset and Scale**
  Allows you to adjust the scale of the item as it is seen on the display. Offset is the number of pixels offset from zero. The scale factor reduces (if less than 1) or increases (if greater than 1) the number of pixels displayed.

  Only the following types are supported in Analog format:

  **VHDL types:**
  - All vectors - std logic vectors, bit vectors, and vectors derived from these types
  - Scalar integers
  - Scalar reals
  - Scalar time

  **Verilog types:**
  - All vectors
  - Scalar real
  - Scalar integers
- **Height**
  Allows you to specify the height (in pixels) of the waveform.

The **Compare** tab includes the same options as those in the Add Signal Options dialog box (see Adding Signals, Regions and/or Clocks (UM-301)).
Setting Wave window display properties

You can define display properties of the Wave window by selecting Edit > Display Properties (Wave window). To save these settings permanently, select Options > Save Preferences (Main window).

The Wave Window Properties dialog box includes the following options:

- **Display Signal Path**
  Sets the display to show anything from the full pathname of each signal (e.g., sim:/top/clk) to only its leaf element (e.g., sim:clk). A non-zero number indicates the number of path elements to be displayed. The default is Full Path.

- **Justify Value**
  Specifies whether the signal values will be justified to the left margin or the right margin in the values window pane.

- **Snap Distance**
  Specifies the distance the cursor needs to be placed from an item edge to jump to that edge (a 0 specification turns off the snap).

- **Row Margin**
  Specifies the distance in pixels between top-level signals.

- **Child Row Margin**
  Specifies the distance in pixels between child signals.
• **Waveform Popup**
  Toggles on/off the popup that displays when you rest your mouse pointer on a signal or comparison object.

• **Dataset Prefix**
  Specifies how signals from different datasets are displayed.
  
  *Always Show Dataset Prefixes*
  All dataset prefixes will be displayed along with the dataset prefix of the current simulation ("sim").

  *Show Dataset Prefixes if 2 or more*
  Displays all dataset prefixes if 2 or more datasets are displayed. "sim" is the default prefix for the current simulation.

  *Never Show No Dataset Prefixes*
  No dataset prefixes will be displayed. This selection is useful if you are running only a single simulation.

### Sorting a group of HDL items

Select **Edit > Sort** to sort the items in the pathname and values panes.

### Setting signal breakpoints

You can set signal breakpoints (a.k.a., when breakpoints; see the **when** command (CR-298) or "Setting signal breakpoints" (UM-192) for more details) using a pop-up menu. Start by selecting a signal and then clicking your second (Windows) or third (UNIX) mouse button. Select **Signal Breakpoints** from the pop-up menu and you’ll see six items:

• **Add**
  Creates a signal breakpoint on the selected signal

• **Edit Breakpoints**
  Opens the Edit When dialog. See "Setting signal breakpoints" (UM-192) for more information.

• **Edit All Breakpoints**
  Opens the Breakpoints dialog. See "Setting file-line breakpoints" (UM-199) for more information.

• **Remove Signal**
  Removes the signal breakpoint from the selected signal

• **Remove All Signals**
  Removes all signal breakpoints

• **Show All**
  Shows a list of all signal breakpoints

When a breakpoint is hit, a message appears in the transcript window about which signal caused the breakpoint. Breakpoints created by the **when** command (CR-298) are not affected by the **Remove All Signals** menu pick, nor are they reported via **Show All**.
Finding items by name or value in the Wave window

The Find dialog box allows you to search for text strings in the Wave window. Select **Edit > Find (Wave window)** to bring up the Find dialog box.

Choose either the Name or Value field to search and enter the value to search for in the Find field. **Find** the item by searching **Down** or **Up** through the Wave window display. **Auto Wrap** continues the search at the top of the window.

The find operation works only within the active pane.

Searching for item values in the Wave window

Select an item in the Wave window and then select **Edit > Search** to bring up the Wave Signal Search dialog box.
The Wave Signal Search dialog box includes these options:

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

- **Search Type: Any Transition**
  Searches for any transition in the selected signal(s).

- **Search Type: Rising Edge**
  Searches for rising edges in the selected signal(s).

- **Search Type: Falling Edge**
  Searches for falling edges in the selected signal(s).

- **Search Type: Search for Signal Value**
  Searches for the value specified in the Value field; the value should be formatted using VHDL or Verilog numbering conventions; see "Numbering conventions" (CR-13).

  ▶ Note: If your signal values are displayed in binary radix, see "Searching for binary signal values in the GUI" (CR-22) for details on how signal values are mapped between a binary radix and std_logic.

- **Search Type: Search for Expression**
  Searches for the expression specified in the Expression field evaluating to a boolean true. Activates the Builder button so you can use "The GUI Expression Builder" (UM-269) if desired.

  The expression can involve more than one signal but is limited to signals logged in the Wave window. Expressions can include constants, variables, and DO files. If no expression is specified, the search will give an error. See "Expression syntax" (CR-23) for more information.

- **Search Options: Match Count**
  You can search for the n-th transition or the n-th match on value; Match Count indicates the number of transitions or matches to search for.

The Search Results are indicated at the bottom of the dialog box.
Using time cursors in the Wave window

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

<table>
<thead>
<tr>
<th>Add Cursor</th>
<th>Delete Cursor</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="Figure" alt="Add Cursor" /></td>
<td><img src="Figure" alt="Delete Cursor" /></td>
</tr>
</tbody>
</table>

- **Add Cursor**: add a cursor to the center of the waveform window
- **Delete Cursor**: delete the selected cursor from the window
Finding a cursor

The cursor value (on the Goto list) corresponds to the simulation time of that cursor. Choose a specific cursor view by selecting Cursor > Goto. You can also select cursors by clicking a value in the cursor-value pane.

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

Making cursor measurements

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

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

The cursors are designed to snap to the closest wave edge to the left on the waveform that the mouse pointer is positioned over. You can control the snap distance via the Edit > Display Properties menu selection.

You can position a cursor without snapping by dragging in the area below the waveforms.

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

<table>
<thead>
<tr>
<th>Find Previous Transition</th>
<th>Find Next Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>locate the previous signal value change for the selected signal</td>
<td>locate the next signal value change for the selected signal</td>
</tr>
</tbody>
</table>

Zooming - changing the waveform display range

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

Using the Zoom menu

You can use the Wave window menu bar, or call up the context menu by clicking the right mouse button in the waveform pane.
The Zoom menu options include:

- **Zoom Area with Mouse Button 1**
  Use mouse button 1 to create a zoom area. Position the mouse cursor to the left side of the desired zoom interval, press mouse button 1 and drag to the right. Release when the box has expanded to the right side of the desired zoom interval.

- **Zoom In**
  Zooms in by a factor of two, increasing the resolution and decreasing the visible range horizontally.

- **Zoom Out**
  Zooms out by a factor of two, decreasing the resolution and increasing the visible range horizontally.

- **Zoom Full**
  Redraws the display to show the entire simulation from time 0 to the current simulation time.

- **Zoom Last**
  Restores the display to where it was before the last zoom operation.

- **Zoom Range**
  Brings up a dialog box that allows you to enter the beginning and ending times for a range of time units to be displayed.

**Zooming with toolbar buttons**

These zoom buttons are available on the toolbar:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Zoom In 2x" /></td>
<td>zoom in by a factor of two from the current view</td>
</tr>
<tr>
<td><img src="image2" alt="Zoom area" /></td>
<td>use the cursor to outline a zoom area</td>
</tr>
<tr>
<td><img src="image3" alt="Zoom out 2x" /></td>
<td>zoom out by a factor of two from current view</td>
</tr>
<tr>
<td><img src="image4" alt="Zoom Full" /></td>
<td>zoom out to view the full range of the simulation from time 0 to the current time</td>
</tr>
</tbody>
</table>

**Zooming with the mouse**

To zoom with the mouse, position the mouse cursor to the left side of the desired zoom interval, press the middle mouse button (three-button mouse), or <Ctrl>+left mouse button (two-button mouse), and while continuing to press, drag to the right and then release at the right side of the desired zoom interval.

**Zooming keyboard shortcuts**

See "Wave window mouse and keyboard shortcuts" (UM-238) for a complete list of Wave window keyboard shortcuts.
Saving zoom range and scroll position with bookmarks

Bookmarks allow you to save a particular zoom range and scroll position. This lets you return easily to a specific view later. You save the bookmark with a name, and then access the named bookmark from the Bookmark menu.

Bookmarks are saved in the Wave format file (see "Adding items with a Wave window format file" (UM-213)) and are restored when the format file is read. There is no limit to the number of bookmarks you can save.

Bookmarks can also be created and managed from the command line. See bookmark add wave command (CR-65) for details.

To add a bookmark, select Bookmark > Add Bookmark (Wave window).

The Bookmark Properties dialog includes the following options.

- **Bookmark Label**
  A text label to assign to the bookmark. The label will identify the bookmark on the Bookmark menu.

- **Zoom**
  A starting value and ending value that define the zoom range.

- **Top Index**
  The item that will display at the top of the wave window. For instance, if you specify 15, the Wave window will be scrolled down to show the 15th item in the window.

- **Save zoom range with bookmark**
  When checked the zoom range will be saved in the bookmark.

- **Save scroll location with bookmark**
  When checked the scroll location will be saved in the bookmark.

Once the bookmark is saved, select it by name from the Bookmark menu, and the Wave window will be zoomed and scrolled accordingly.
To edit or delete a bookmark, select **Bookmark > Edit Bookmarks** (Wave window).

The Bookmark Selection dialog includes the following options.

- **Add** (bookmark add wave)
  Add a new bookmark

- **Modify**
  Edit the selected bookmark

- **Delete** (bookmark delete wave)
  Delete the selected bookmark

- **Delete All** (bookmark delete wave)
  Delete all bookmarks

- **Goto** (bookmark goto wave)
  Zoom and scroll the Wave window using the selected bookmark
# Wave window mouse and keyboard shortcuts

The following mouse actions and keystrokes can be used in the Wave window.

<table>
<thead>
<tr>
<th>Mouse action</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;control - left-button - click on a scroll arrow&gt;</td>
<td>scrolls window to very top or bottom (vertical scroll) or far left or right (horizontal scroll)</td>
</tr>
<tr>
<td>&lt;middle mouse-button - click in scroll bar trough&gt; (UNIX) only</td>
<td>scrolls window to position of click</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Keystroke</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>i I or +</td>
<td>zoom in</td>
</tr>
<tr>
<td>o O or -</td>
<td>zoom out</td>
</tr>
<tr>
<td>f or F</td>
<td>zoom full; mouse pointer must be over the the cursor or waveform panes</td>
</tr>
<tr>
<td>l or L</td>
<td>zoom last</td>
</tr>
<tr>
<td>r or R</td>
<td>zoom range</td>
</tr>
<tr>
<td>&lt;arrow up&gt;</td>
<td>scroll waveform display up by selecting the item above the currently selected item</td>
</tr>
<tr>
<td>&lt;arrow down&gt;</td>
<td>scroll waveform display down by selecting the item below the currently selected item</td>
</tr>
<tr>
<td>&lt;arrow left&gt;</td>
<td>scroll waveform display left</td>
</tr>
<tr>
<td>&lt;arrow right&gt;</td>
<td>scroll waveform display right</td>
</tr>
<tr>
<td>&lt;page up&gt;</td>
<td>scroll waveform display up by a page</td>
</tr>
<tr>
<td>&lt;page down&gt;</td>
<td>scroll waveform display down by a page</td>
</tr>
<tr>
<td>&lt;tab&gt;</td>
<td>search forward (right) to the next transition on the selected signal - finds the next edge</td>
</tr>
<tr>
<td>&lt;shift-tab&gt;</td>
<td>search backward (left) to the previous transition on the selected signal - finds the previous edge</td>
</tr>
<tr>
<td>&lt;control-f&gt; Windows &lt;control-s&gt; UNIX</td>
<td>open the find dialog box; searches within the specified field in the pathname pane for text strings</td>
</tr>
</tbody>
</table>
Printing and saving waveforms

**Saving a .eps file and printing under UNIX**

Select **File > Print Postscript** (Wave window) to print all or part of the waveform in the current Wave window in UNIX, or save the waveform as a .eps file on any platform (see also write wave command (CR-313)). Printing and writing preferences are controlled by the dialog box shown below.

The **Write Postscript** dialog box includes these options:

**Printer**

- **Print command**
  Enter a UNIX print command to print the waveform in a UNIX environment.

- **File name**
  Enter a filename for the encapsulated Postscript (.eps) file to be created; or browse to a previously created .eps file and use that filename.

**Signal Selection**

- **All signals**
  Print all signals.

- **Current View**
  Print signals in the current view

- **Selected**
  Print all selected signals
Time Range

- **Full Range**
  Print all specified signals in the full simulation range.

- **Current view**
  Print the specified signals for the viewable time range.

- **Custom**
  Print the specified signals for a user-designated From and To time.

Setup button

See "Printer Page Setup" (UM-242)

**Printing on Windows platforms**

Select File > Print (Wave window) to print all or part of the waveform in the current Wave window, or save the waveform as a printer file (a Postscript file for Postscript printers). Printing and writing preferences are controlled by the dialog box shown below.

Printer

- **Name**
  Choose the printer from the drop-down menu. Set printer properties with the Properties button.

- **Status**
  Indicates the availability of the selected printer.
• **Type**  
  Printer driver name for the selected printer. The driver determines what type of file is output if "Print to file" is selected.

• **Where**  
  The printer port for the selected printer.

• **Comment**  
  The printer comment from the printer properties dialog box.

• **Print to file**  
  Make this selection to print the waveform to a file instead of a printer. The printer driver determines what type of file is created. Postscript printers create a Postscript (.ps) file, non-Postscript printers create a .prn or printer control language file. To create an encapsulated Postscript file (.eps) use the **File > Print Postscript** menu selection.

**Signal Selection**

• **All signals**  
  Print all signals.

• **Current View**  
  Print signals in current view.

• **Selected**  
  Print all selected signals.

**Time Range**

• **Full Range**  
  Print all specified signals in the full simulation range.

• **Current view**  
  Print the specified signals for the viewable time range.

• **Custom**  
  Print the specified signals for a user-designated **From** and **To** time.

**Setup button**

See "**Printer Page Setup**" (UM-242)
**Printer Page Setup**

Clicking the Setup button in the Write Postscript or Print dialog box allows you to define the following options (this is the same dialog that opens via File > Page setup).

- **Paper Size**
  Select your output page size from a number of options; also choose the paper width and height.

- **Margins**
  Specify the page margins; changing the Margin will change the Scale and Page specifications.

- **Label width**
  Specify Auto Adjust to accommodate any length label, or set a fixed label width.

- **Cursors**
  Turn printing of cursors on or off.

- **Grid**
  Turn printing of grid lines on or off.
• **Color**  
Select full color printing, grayscale or black and white.

• **Scaling**  
Specify a **Fixed** output time width in nanoseconds per page – the number of pages output is automatically computed; or, select **Fit to** to define the number of pages to be output based on the paper size and time settings; if set, the time-width per page is automatically computed.

• **Orientation**  
Select the output page orientation, **Portrait** or **Landscape**.
Compiling with the graphic interface

You can use a project or the **Compile HDL Source Files** dialog box to compile VHDL or Verilog designs. For information on compiling in a project, see "Getting started with projects" (UM-24). To open the Compile HDL Source Files dialog, select the Compile button (Main window) or **Design > Compile**.

![Compile HDL Source Files dialog box](image)

The Compile HDL Source Files dialog box opens as shown below.
From the Compile HDL Source Files dialog box you can:

- select source files to compile in any language combination
- specify the target library for the compiled design units
- select among the compiler options for either VHDL or Verilog

Select the Default Options button to change the compiler options, see “Setting default compile options” (UM-246) for details. The same Compiler Options dialog box can also be accessed by selecting Options > Compile (Main window) or by selecting Compile Properties from the context menu in Project tab.

Select the Edit Source button to view or edit a source file via the Compile dialog box. See “Source window” (UM-195) for additional source file editing information.

**Locating source errors during compilation**

If a compiler error occurs during compilation, a red error message is printed in the Main transcript. Double-click on the error message to open the source file in an editable Source window with the error highlighted.

double-click on the error in the Main window and the error is highlighted and ready to edit in the Source window
Setting default compile options

Select Options > Compile (Main window) to bring up the Compiler Options dialog box shown below. OK accepts the changes made and closes the dialog box. Apply makes the changes with the dialog box open so you can test your settings. Cancel closes the dialog box and makes no changes. The options found on each tab of the dialog box are detailed below. Changes made in the Compiler Options dialog box become the default for all future simulations.

VHDL compiler options tab

- **Use 1993 language syntax**
  Specifies the use of VHDL93 during compilation. The 1987 standard is the default. Same as the -93 switch for the vcom command (CR-240). Edit the VHDL93 (UM-395) variable in the modelsim.ini file to set a permanent default.

- **Don’t put debugging info in library**
  Models compiled with this option do not use any of the ModelSim debugging features. Consequently, your user will not be able to see into the model. This also means that you cannot set breakpoints or single step within this code. Don’t compile with this option until you’re done debugging. Same as the -nodebug switch for the vcom command (CR-240). See “Source code security and -nodebug” (UM-420) for more details. Edit the NoDebug (UM-387) variable in the modelsim.ini file to set a permanent default.
• **Use explicit declarations only**
  Used to ignore an error in packages supplied by some other EDA vendors; directs the compiler to resolve ambiguous function overloading in favor of the explicit function definition. Same as the -explicit switch for the vcom command (CR-240). Edit the Explicit (UM-387) variable in the modelsim.ini file to set a permanent default.

  Although it is not intuitively obvious, the = operator is overloaded in the std_logic_1164 package. All enumeration data types in VHDL get an “implicit” definition for the = operator. So while there is no explicit = operator, there is an implicit one. This implicit declaration can be hidden by an explicit declaration of = in the same package (LRM Section 10.3). However, if another version of the = operator is declared in a different package than that containing the enumeration declaration, and both operators become visible through use clauses, neither can be used without explicit naming, for example:

  ARITHMETIC."="(left, right)

  This option allows the explicit = operator to hide the implicit one.

• **Disable loading messages**
  Disables loading messages in the Main window. Same as the -quiet switch for the vcom command (CR-240). Edit the Quiet (UM-387) variable in the modelsim.ini file to set a permanent default.

• **Show source lines with errors**
  Causes the compiler to display the relevant lines of code in the transcript. Same as the -source switch for the vcom command (CR-240). Edit the Show_source (UM-387) variable in the modelsim.ini file to set a permanent default.

**Flag Warnings on:**

• **Unbound Component**
  Flags any component instantiation in the VHDL source code that has no matching entity in a library that is referenced in the source code, either directly or indirectly. Edit the Show_Warning1 (UM-387) variable in the modelsim.ini file to set a permanent default.

• **Process without a WAIT statement**
  Flags any process that does not contain a wait statement or a sensitivity list. Edit the Show_Warning2 (UM-387) variable in the modelsim.ini file to set a permanent default.

• **Null Range**
  Flags any null range, such as 0 down to 4. Edit the Show_Warning3 (UM-387) variable in the modelsim.ini file to set a permanent default.

• **No space in time literal (e.g. 5ns)**
  Flags any time literal that is missing a space between the number and the time unit. Edit the Show_Warning4 (UM-387) variable in the modelsim.ini file to set a permanent default.

• **Multiple drivers on unresolved signals**
  Flags any unresolved signals that have multiple drivers. Edit the Show_Warning5 (UM-387) variable in the modelsim.ini file to set a permanent default.

**Check for:**

• **Synthesis**
  Turns on limited synthesis-rule compliance checking. Edit the CheckSynthesis (UM-386) variable in the modelsim.ini file to set a permanent default.
• **Vital Compliance**  
  Toggle Vital compliance checking. Edit the `NoVitalCheck` (UM-387) variable in the `modelsim.ini` file to set a permanent default.

**Optimize for:**

• **StdLogic1164**  
  Causes the compiler to perform special optimizations for speeding up simulation when the multi-value logic package `std_logic_1164` is used. Unless you have modified the `std_logic_1164` package, this option should always be checked. Edit the `Optimize_1164` (UM-387) variable in the `modelsim.ini` file to set a permanent default.

• **Vital**  
  Toggle acceleration of the Vital packages. Edit the `NoVital` (UM-387) variable in the `modelsim.ini` file to set a permanent default.

**Verilog compiler options tab**

- **Enable run-time hazard checks**  
  Enables the run-time hazard checking code. Same as the `-hazards` switch for the `vlog` command (CR-274). Edit the `Hazard` (UM-388) variable in the `modelsim.ini` file to set a permanent default.
• **Disable debugging data**
Models compiled with this option do not use any of the ModelSim debugging features. Consequently, your user will not be able to see into the model. This also means that you cannot set breakpoints or single step within this code. Don’t compile with this option until you’re done debugging. Same as the `-nodebug` switch for the `vlog` command (CR-274). See "Source code security and `-nodebug`" (UM-420) for more details. Edit the `NoDebug` (UM-387) variable in the `modelsim.ini` file to set a permanent default.

• **Convert Verilog identifiers to upper-case**
Converts regular Verilog identifiers to uppercase. Allows case insensitivity for module names. Same as the `-u` switch for the `vlog` command (CR-274). Edit the `UpCase` (UM-388) variable in the `modelsim.ini` file to set a permanent default.

• **Disable loading messages**
Disables loading messages in the Main window. Same as the `-quiet` switch for the `vlog` command (CR-274). Edit the `Quiet` (UM-387) variable in the `modelsim.ini` file to set a permanent default.

• **Show source lines with errors**
Causes the compiler to display the relevant lines of code in the transcript. Same as the `-source` switch for the `vlog` command (CR-274). Edit the `Show_source` (UM-387) variable in the `modelsim.ini` file to set a permanent default.

**Other Verilog Options:**

• **Library Search**
Specifies the Verilog source library directory to search for undefined modules. Same as the `-y <library_directory>` switch for the `vlog` command (CR-274).

• **Extension**
Specifies the suffix of files in the library directory. Multiple suffixes can be used. Same as the `+libext+<suffix>` switch for the `vlog` command (CR-274).

• **Library File**
Specifies the Verilog source library file to search for undefined modules. Same as the `-v <library_file>` switch for the `vlog` command (CR-274).

• **Include Directory**
Specifies a directory for files included with the `#include filename` compiler directive. Same as the `+incdir+<directory>` switch for the `vlog` command (CR-274).

• **Macro**
Defines a macro to execute during compilation. Same as the compiler directive: `define macro_name macro_text`. Also the same as the `+define+<macro_name> [ =<macro_text> ]` switch for the `vlog` command (CR-274).
Simulating with the graphic interface

You can use a project or the **Load Design** dialog box to simulate a compiled design. For information on simulating in a project, see "Getting started with projects" (UM-24). To open the Load Design dialog, select the **Load Design** button (Main window) or **Design > Load Design**.

Five tabs - **Design**, **VHDL**, **Verilog**, **Libraries**, and **SDF** - allow you to select various simulation options.

You can switch between tabs to modify settings, then begin simulation by selecting the **Load** button. If you select **Cancel**, all selections remain unchanged and you are returned to the Main window; the **Exit** button (only active before simulation) closes ModelSim. The **Save Settings** button allows you to save the preferences on all tabs to a DO (macro) file.

**Compile before you simulate**

To begin simulation you must have compiled design units located in a design library, see "Creating a design library" (UM-53).

► **Note**: Many of the dialog box options discussed in this section include parenthetical elements that correspond to **vsim** (CR-284) command options. For example, **Simulator Resolution** (-time [<multiplier>]<time_unit>).
Design selection tab

![Load Design dialog box](image)

▶ Note: The Exit button closes the Load Design dialog box and quits ModelSim.

The Design tab includes these options:

- **Library**
  Specifies a library to view. Make certain your selection is a valid ModelSim library — the library must be created by ModelSim and it’s directory must include a _info file.

- **Design Unit**
  This hierarchical list allows you to select one top-level entity or configuration to be simulated. All entities, configurations, and modules that exist in the specified library are displayed in the list box. Architectures can be viewed by selecting the "+" box before any name.
• **Simulate** *(<configuration> | <module> | <entity> | [<architecture>]))*
  Specifies the design unit(s) to simulate. You can simulate several Verilog top-level modules or a VHDL top-level design unit in one of three ways:

  1. Type a design unit name (configuration, module, or entity) into the field, separate additional names with a space. Specify library/design units with the following syntax:
     
     

     ![Syntax Example](library_name.]


     ![Design Unit Name](design_unit)

  2. Click on a name in the **Design Unit** list below and click the **Add** button.

  3. Leave this field blank and click on a name in the **Design Unit** list (single unit only).

• **Simulator Resolution**
  
  *(.-time [<multiplier>]<time_unit>)*

  The drop-down menu sets the simulator time units (original default is ns).

  Simulation time units can be expressed as any of the following:

<table>
<thead>
<tr>
<th>Simulation time units</th>
<th>Time Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>1fs, 10fs, or 100fs</td>
<td>femtoseconds</td>
</tr>
<tr>
<td>1ps, 10ps, or 100ps</td>
<td>picoseconds</td>
</tr>
<tr>
<td>1ns, 10ns, or 100ns</td>
<td>nanoseconds</td>
</tr>
<tr>
<td>1us, 10us, or 100us</td>
<td>microseconds</td>
</tr>
<tr>
<td>1ms, 10ms, or 100ms</td>
<td>milliseconds</td>
</tr>
<tr>
<td>1sec, 10sec, or 100sec</td>
<td>seconds</td>
</tr>
</tbody>
</table>

  See also, "Selecting the time resolution" (UM-54).
VHDL settings tab

The **VHDL** tab includes these options:

**Generics**

The **Add** button opens a dialog box (shown below) that allows you to specify the value of generics within the current simulation; generics are then added to the **Generics** list. You can also select a generic on the listing to **Delete** or **Edit**.
From the **Specify a Generic** dialog box you can set the following options.

- **Generic Name** (-g <Name>=<Value>)
  The name of the generic parameter. Type it in as it appears in the VHDL source (case is ignored).

- **Value**
  Specifies a value for all generics in the design with the given name (above) that have not received explicit values in generic maps (such as top-level generics and generics that would otherwise receive their default value). The value must be appropriate for the declared data type of the generic parameter. No spaces are allowed in the specification (except within quotes) when specifying a string value.

- **Override Instance-specific Values** (-G <Name>=<Value>)
  Select to override generics that received explicit values in generic maps. The name and value are specified as above. The use of this switch is indicated in the **Override Instance** column of the **Generics** list.

The **OK** button adds the generic to the **Generics** listing; **Cancel** dismisses the dialog box without changes.

**VITAL**

- **Disable Timing Checks** (+notimingchecks)
  Disables timing checks generated by VITAL models.

- **Use Vital 2.2b SDF Mapping** (-vital2.2b)
  Selects SDF mapping for VITAL 2.2b (default is Vital95).

- **Disable Glitch Generation** (-noglitch)
  Disables VITAL glitch generation.

**TEXTIO files**

- **STD_INPUT** (-std_input <filename>)
  Specifies the file to use for the VHDL textio STD_INPUT file. Use the **Browse** button to locate a file within your directories.

- **STD_OUTPUT** (-std_output <filename>)
  Specifies the file to use for the VHDL textio STD_OUTPUT file. Use the **Browse** button to locate a file within your directories.
Verilog settings tab

The **Verilog** tab includes these options:

- **Delay Selection** (+mindelays | +typdelays | +maxdelays)
  Use the drop-down menu to select timing for min:typ:max expressions.

**Pulse Options**

- **Disable pulse error and warning messages** (+no_pulse_msg)
  Disables path pulse error warning messages.

- **Rejection Limit** (+pulse_r/<percent>)
  Sets the module path pulse rejection limit as a percentage of the path delay.

- **Error Limit** (+pulse_e/<percent>)
  Sets the module path pulse error limit as a percentage of the path delay.
Other Options

- **Enable Hazard Checking** (-hazards)
  Enables hazard checking in Verilog modules.

- **Disable Timing Checks in Specify Blocks** (+notimingchecks)
  Disables the timing check system tasks ($setup, $hold,...) in specify blocks.

- **User Defined Arguments** (+<plusarg>)
  Arguments are preceded with “+”, making them accessible through the Verilog PLI routine `mc_scan_plusargs`. The values specified in this field must have a "+" preceding them or ModelSim may incorrectly parse them.

Libraries settings tab

The **Libraries** tab includes these options:

- **Search Libraries** (-L)
  Specifies the library to search for design units instantiated from Verilog.

- **Search Libraries First** (-Lf)
  Same as Search Libraries but these libraries are searched before ‘uselib.'
SDF settings tab

The **SDF** (Standard Delay Format) tab includes these options:

**SDF Files**

The **Add** button opens a dialog box that allows you to specify the SDF files to load for the current simulation; files are then added to the **Region/File** list. You may also select a file on the listing to **Delete** or **Edit** (opens the dialog box below).
From the Specify an SDF File dialog box you can set the following options.

- **SDF file** ([<region>] = <sdf_filename>)
  Specifies the SDF file to use for annotation. Use the Browse button to locate a file within your directories.

- **Apply to region** ([<region>] = <sdf_filename>)
  Specifies the design region to use with the selected SDF options.

- **Delay Selection** (-sdfmin | -sdftyp | -sdfmax)
  The drop-down menu selects delay timing (min, typ or max) to be used from the specified SDF file. See also, “Specifying SDF files for simulation” (UM-320).

  The OK button places the specified SDF file and delay on the Region/File list; Cancel dismisses the dialog box without changes.

### SDF options

- **Disable SDF warnings** (-sdfnowarn)
  Select to disable warnings from the SDF reader.

- **Reduce SDF errors to warnings** (-sdfnoerror)
  Change SDF errors to warnings so the simulation can continue.

- **Multi-Source Delay** (-multisource_delay <sdf_option>)
  Select max, min or latest delay. Controls how multiple PORT or INTERCONNECT constructs that terminate at the same port are handled. By default, the Module Input Port Delay (MIPD) is set to the max value encountered in the SDF file. Alternatively, you can choose the min or latest of the values.
Setting default simulation options

Select Options > Simulation... (Main window) to bring up the Simulation Options dialog box shown below. Options you can set for the current simulation include: default radix, default force type, default run length, iteration limit, warning suppression, break on assertion specifications, and WLF file configuration. OK accepts the changes made and closes the dialog box. Apply makes the changes with the dialog box open so you can test your settings. Cancel closes the dialog box and makes no changes. The options found on each tab are detailed below.

▶ Note: Changes made in the Simulation Options dialog box are the default for the current simulation only. Options can be saved as the default for future simulations by editing the simulator control variables in the modelsim.ini file; the variables to edit are noted in the text below. You can use Notepad (see notepad command (CR-164)) to edit the variables in modelsim.ini if you wish. See also, “Projects and system initialization” (UM-21) for more information.

**Default settings tab**

The Defaults tab includes these options:

- **Default Radix**
  Sets the default radix for the current simulation run. You can also use the radix (CR-189) command to set the same temporary default. A permanent default can be set by editing the DefaultRadix (UM-389) variable in the modelsim.ini file. The chosen radix is used for all commands (force (CR-144), examine (CR-138), change (CR-73) are examples) and for displayed values in the Signals, Variables, Dataflow, List, and Wave windows.
- **Suppress Warnings**
  Selecting **From Synopsys Packages** suppresses warnings generated within the accelerated Synopsys std_arith packages. Edit the **StdArithNoWarnings** (UM-390) variable in the modelsim.ini file to set a permanent default.

  Selecting **From IEEE Numeric Std Packages** suppresses warnings generated within the accelerated numeric_std and numeric_bit packages. Edit the **NumericStdNoWarnings** (UM-390) variable in the modelsim.ini file to set a permanent default.

- **Default Run**
  Sets the default run length for the current simulation. Edit the **RunLength** (UM-390) variable in the modelsim.ini file to set a permanent default.

- **Iteration Limit**
  Sets a limit on the number of deltas within the same simulation time unit to prevent infinite looping. Edit the **IterationLimit** (UM-389) variable in the modelsim.ini file to set a permanent iteration limit default.

- **Default Force Type**
  Selects the default force type for the current simulation. Edit the **DefaultForceKind** (UM-389) variable in the modelsim.ini file to set a permanent default.

**Assertion settings tab**

The **Assertions** tab includes these options:

- **Break on Assertion**
  Selects the assertion severity that will stop simulation. Edit the **BreakOnAssertion** (UM-388) variable in the modelsim.ini file to set a permanent default.
• **Ignore Assertions For**
Selects the assertion type to ignore for the current simulation. Multiple selections are possible. Edit the IgnoreFailure, IgnoreError, IgnoreWarning, and IgnoreNote (UM-389) variables in the modelsim.ini file to set permanent defaults.

When an assertion type is ignored, no message will be printed, nor will the simulation halt (even if break on assertion is set for that type).

*Note:* Assertions that appear within an instantiation or configuration port map clause conversion function will not stop the simulation regardless of the severity level of the assertion.

**WLF settings tab**

The **WLF Files** tab includes these options:

• **WLF File Size Limit**
Limits the WLF file by size (as closely as possible) to the specified number of megabytes. If both size and time limits are specified, the most restrictive is used. Setting it to 0 results in no limit. Edit the WLFSizeLimit (UM-391) variable in the modelsim.ini file to set a permanent default.

• **WLF File Time Limit**
Limits the WLF file by size (as closely as possible) to the specified amount of time. If both time and size limits are specified, the most restrictive is used. Setting it to 0 results in no limit. Edit the WLFTimeLimit (UM-391) variable in the modelsim.ini file to set a permanent default.
• **Compress WLF data**
  Compresses WLF files to reduce their size. You would typically only disable compression for troubleshooting purposes. Edit the `WLFCompress` (UM-391) variable in the `modelsim.ini` file to set a permanent default.

• **Delete WLF file on exit**
  Specifies whether the WLF file should be deleted when the simulation ends. Edit the `WLFDeleteOnQuit` (UM-391) variable in the `modelsim.ini` file to set a permanent default.

• **Design Hierarchy**
  Specifies whether to save all design hierarchy in the WLF file or only regions containing logged signals. Edit the `WLFSaveAllRegions` (UM-391) variable in the `modelsim.ini` file to set a permanent default.
Several tools are available from ModelSim menus. The menu selections to locate the tools are below the tool names. Follow the links for more information on each tool.

- "The Button Adder" (UM-263)
  Window > Customize (any window)
  Allows you to add a temporary function button or toolbar to any window.

- "The Macro Helper" (UM-264)
  Macro > Macro Helper (Main window)
  Creates macros by recording mouse movements and key strokes. UNIX only (excluding Linux).

- "The Tcl Debugger" (UM-265)
  Macro > Tcl Debugger (Main window)
  Helps you debug your Tcl procedures.

- "The GUI Expression Builder" (UM-269)
  Edit > Search > Search for Expression > Builder (List or Wave window)
  Helps you build logical expressions for use in Wave and List window searches and several simulator commands. For expression format syntax see "GUI_expression_format" (CR-19).

The Button Adder

The ModelSim Button Adder creates a single button, or a combined button and toolbar in any currently opened ModelSim window. The button exists until you close the window. (See "Buttons the easy way" (UM-273).)

Note: When a button is created with the Button Adder, the commands that created the button are echoed in the transcript. The transcript can then be saved and used as a DO file, allowing you to reuse the commands to recreate the button from a startup DO file. See "Using a startup file" (UM-394) for additional information.

Invoke the Button Adder from any ModelSim window menu: Window > Customize.

You have the following options for adding a button:

- **Window Name** is the name of the window to which you wish to add the button.
- **Button Name** is the button’s label.
- **Function** can be any command or macro you might execute from the ModelSim command line. For example, you might want to add a Run or Step button to the Wave window.

Locate the button within the window with these selections:

- **Toolbar** places the button on a new toolbar.
The Macro Helper

This tool is available for UNIX only (excluding Linux).

The purpose of the Macro Helper is to aid macro creation by recording a simple series of mouse movements and key strokes. The resulting file can be called from a more complex macro by using the **play** (CR-171) command. Actions recorded by the Macro Helper can only take place within the ModelSim GUI (window sizing and repositioning are not recorded because they are handled by your operating system’s window manager). In addition, the **run** (CR-199) commands cannot be recorded with the Macro Helper but can be invoked as part of a complex macro.

Select **Macro > Macro Helper** (Main window) to access the Macro Helper.

- **Record a macro**
  by typing a new macro file name into the field provided, then press **Record**. Use the **Pause** and **Stop** buttons as shown in the table below.

- **Play a macro**
  by entering the file name of a Macro Helper file into the field and pressing **Play**.

Files created by the Macro Helper can be viewed with the **notepad** (CR-164).

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record/Stop</td>
<td>Record begins recording and toggles to Stop once a recording begins</td>
</tr>
<tr>
<td>Insert Pause</td>
<td>inserts a .5 second pause into the macro file; press the button more than once to add more pause time; the pause time can subsequently be edited in the macro file</td>
</tr>
<tr>
<td>Play</td>
<td>plays the Macro Helper file specified in the file name field</td>
</tr>
</tbody>
</table>

See the **macro_option** command (CR-158) for playback speed, delay and debugging options for completed macro files.
The Tcl Debugger

We would like to thank Gregor Schmid for making TDebug available for use in the public domain.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of FITNESS FOR A PARTICULAR PURPOSE.

Starting the debugger

Select Macro > Tcl Debugger (Main window) to run the debugger. Make sure you use the ModelSim and TDebug menu selections to invoke and close the debugger. If you would like more information on the configuration of TDebug see Help > Technotes > tdebug.

The following text is an edited summary of the README file distributed with TDebug.

How it works

TDebug works by parsing and redefining Tcl/Tk-procedures, inserting calls to ‘td_eval’ at certain points, which takes care of the display, stepping, breakpoints, variables etc. The advantages are that TDebug knows which statement in what procedure is currently being executed and can give visual feedback by highlighting it. All currently accessible variables and their values are displayed as well. Code can be evaluated in the context of the current procedure. Breakpoints can be set and deleted with the mouse.

Unfortunately there are drawbacks to this approach. Preparation of large procedures is slow and due to Tcl’s dynamic nature there is no guarantee that a procedure can be prepared at all. This problem has been alleviated somewhat with the introduction of partial preparation of procedures. There is still no possibility to get at code running in the global context.

The Chooser

Select Macro > Tcl Debugger (Main window) to open the TDebug chooser.

The TDebug chooser has three parts. At the top the current interpreter, vsim.op_, is shown. In the main section there are two list boxes. All currently defined procedures are shown in the left list box. By clicking the left mouse button on a procedure name, the procedure gets prepared for debugging and its name is moved to the right list box. Clicking a name in the right list box returns a procedure to its normal state. Press the right mouse button on a procedure in either list box to get its program code displayed in the main debugger window.

The three buttons at the bottom let you force a Rescan of the available procedures, Popup the debugger window or Exit TDebug. Exiting from TDebug doesn't
terminate ModelSim, it merely detaches from `vsim.op_`, restoring all prepared procedures to their unmodified state.

**The Debugger**

Select the **Popup** button in the Chooser to open the debugger window.

The debugger window is divided into the main region with the name of the current procedure (Proc), a listing in which the expression just executed is highlighted, the **Result** of this execution and the currently available **Variables** and their values, an entry to **Eval** expressions in the context of the current procedure and some button controls for the state of the debugger.

A procedure listing displayed in the main region will have a darker background on all lines that have been prepared. You can prepare or restore additional lines by selecting a region (<Button-1>, standard selection) and choosing **Selection > Prepare Proc** or **Selection > Restore Proc** from the debugger menu (or by pressing `^P` or `^R`).

When using ‘Prepare’ and ‘Restore’, try to be smart about what you intend to do. If you select just a single word (plus some optional white space) it will be interpreted as the name of a procedure to prepare or restore. Otherwise, if the selection is owned by the listing, the corresponding lines will be used.

Be careful with partial prepare or restore! If you prepare random lines inside a ‘switch’ or ‘bind’ expression, you may get surprising results on execution, because the parser doesn’t know about the surrounding expression and can’t try to prevent problems.
There are seven possible debugger states, one for each button and an ‘idle’ or ‘waiting’ state when no button is active. The button-activated states are:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop</td>
<td>stop after next expression, used to get out of slow/fast/nonstop mode</td>
</tr>
<tr>
<td>Next</td>
<td>execute one expression, then revert to idle</td>
</tr>
<tr>
<td>Slow</td>
<td>execute until end of procedure, stopping at breakpoints or when</td>
</tr>
<tr>
<td></td>
<td>the state changes to stop; after each execution, stop for ‘delay’ milliseconds; the delay can be changed with the ‘+’ and ‘-’ buttons</td>
</tr>
<tr>
<td>Fast</td>
<td>execute until end of procedure, stopping at breakpoints</td>
</tr>
<tr>
<td>Nonstop</td>
<td>execute until end of procedure without stopping at breakpoints or</td>
</tr>
<tr>
<td></td>
<td>updating the display</td>
</tr>
<tr>
<td>Break</td>
<td>terminate execution of current procedure</td>
</tr>
</tbody>
</table>

Closing the debugger doesn’t quit it, it only does ‘wm withdraw’. The debugger window will pop up the next time a prepared procedure is called. Make sure you close the debugger with **Debugger > Close**.

**Breakpoints**

To set/unset a breakpoint, double-click inside the listing. The breakpoint will be set at the innermost available expression that contains the position of the click. There’s no support for conditional or counted breakpoints.

The **Eval** entry supports a simple history mechanism available via the `<Up_arrow>` and `<Down_arrow>` keys. If you evaluate a command while stepping through a procedure, the command will be evaluated in the context of the procedure; otherwise it will be evaluated at the global level. The result will be displayed in the result field. This entry is useful for a lot of things, but especially to get access to variables outside the current scope.
Try entering the line ‘global td_priv’ and watch the Variables box (with global and array variables enabled of course).

**Configuration**

You can customize TDebug by setting up a file named .tdebugrc in your home directory. See the TDebug README at Help > Technotes > tdebug for more information on the configuration of TDebug.

**TclPro Debugger**

The Macro menu in the Main window contains a selection for the TclPro Debugger from Scriptics Corporation. This debugger can be acquired from Scriptics. Once acquired, do the following steps to use the TclPro Debugger:

1. Launch TclPro Debugger
2. Launch ModelSim
3. Select Macro > TclPro Debugger (Main window)

This will connect ModelSim to the Scriptics TclPro Debugger.
The GUI Expression Builder

The GUI Expression Builder is a feature of the Wave and List Signal Search dialog boxes, and the List trigger properties dialog box. It aids in building a search expression that follows the "GUI_expression_format" (CR-19).

To locate the Builder:

- select Edit > Search (List or Wave window)
- select the Search for Expression option in the resulting dialog box
- select the Builder button

The Expression Builder dialog box provides an array of buttons that help you build a GUI expression. For instance, rather than typing in a signal name, you can select the signal in the associated Wave or List window and press Insert Reference Signal in the Expression Builder. The result will be the full signal name added to the expression field. All Expression Builder buttons correspond to the "Expression syntax" (CR-23).
To search for when a signal reaches a particular value

Select the signal in the Wave window and click Insert Reference Signal and ==. Then, click the value buttons or type a value.

To evaluate only on clock edges

Click the && button to AND this condition with the rest of the expression. Then select the clock in the Wave window and click Insert Reference Signal and rising. You can also select the falling edge or both edges.

Operators

Other buttons will add operators of various kinds (see "Expression syntax" (CR-23)), or you can type them in.

See "Setting up a List trigger with Expression Builder" (UM-434) for an additional Expression builder example.
### Graphic interface commands

The following commands provide control and feedback during simulation as well as the ability to edit, and add menus and buttons to the interface. Only brief descriptions are provided here; for more information and command syntax see the *ModelSim Command Reference*.

<table>
<thead>
<tr>
<th>Window control and feedback commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>batch_mode</strong> (CR-63)</td>
<td>returns a 1 if ModelSim is operating in batch mode, otherwise returns a 0; it is typically used as a condition in an if statement</td>
</tr>
<tr>
<td><strong>configure</strong> (CR-109)</td>
<td>invokes the List or Wave widget configure command for the current default List or Wave window</td>
</tr>
<tr>
<td><strong>down</strong> (CR-128)</td>
<td>moves the active marker in the List window down to the next transition on the selected signal that matches the specifications</td>
</tr>
<tr>
<td><strong>getactivecursortime</strong> (CR-147)</td>
<td>gets the time of the active cursor in the Wave window</td>
</tr>
<tr>
<td><strong>getactivemarkertime</strong> (CR-148)</td>
<td>gets the time of the active marker in the List window</td>
</tr>
<tr>
<td><strong>left</strong> (CR-152)</td>
<td>searches left through the specified Wave window for signal transitions or values</td>
</tr>
<tr>
<td><strong>notepad</strong> (CR-164)</td>
<td>a simple text editor; used to view and edit ASCII files or create new files</td>
</tr>
<tr>
<td><strong>play</strong> (CR-171)</td>
<td><strong>UNIX only (excluding Linux)</strong> - replays a sequence of keyboard and mouse actions that were previously saved to a file with the <strong>record</strong> command (CR-190)</td>
</tr>
<tr>
<td><strong>property list</strong> (CR-184)</td>
<td>changes properties of an HDL item in the List window display</td>
</tr>
<tr>
<td><strong>property wave</strong> (CR-185)</td>
<td>changes properties of an HDL item in the waveform or signal name display in the Wave window</td>
</tr>
<tr>
<td><strong>record</strong> (CR-190)</td>
<td><strong>UNIX only (excluding Linux)</strong> - starts recording a replayable trace of all keyboard and mouse actions</td>
</tr>
<tr>
<td><strong>right</strong> (CR-197)</td>
<td>searches right through the specified Wave window for signal transitions or values</td>
</tr>
<tr>
<td><strong>search</strong> (CR-201)</td>
<td>searches the specified window for one or more items matching the specified pattern(s)</td>
</tr>
<tr>
<td><strong>seetime</strong> (CR-205)</td>
<td>scrolls the List or Wave window to make the specified time visible</td>
</tr>
<tr>
<td><strong>transcribe</strong> (CR-216)</td>
<td>displays a command in the Main window, then executes the command</td>
</tr>
<tr>
<td><strong>up</strong> (CR-219)</td>
<td>moves the active marker in the List window up to the next transition on the selected signal that matches the specifications</td>
</tr>
<tr>
<td><strong>write preferences</strong> (CR-308)</td>
<td>saves the current GUI preference settings to a Tcl preference file</td>
</tr>
<tr>
<td>Window menu and button commands</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>add button (CR-47)</td>
<td>adds a user-defined button to the Main window button bar</td>
</tr>
<tr>
<td>add_menu (CR-52)</td>
<td>adds a menu to the menu bar of the specified window</td>
</tr>
<tr>
<td>add_menucb (CR-54)</td>
<td>creates a checkbox within the specified menu of the specified window</td>
</tr>
<tr>
<td>add_menuitem (CR-55)</td>
<td>creates a menu item within the specified menu of the specified window</td>
</tr>
<tr>
<td>add_separator (CR-56)</td>
<td>adds a separator as the next item in the specified menu path in the specified window</td>
</tr>
<tr>
<td>add_submenu (CR-57)</td>
<td>creates a cascading submenu within the specified menu_path of the specified window</td>
</tr>
<tr>
<td>change_menu_cmd (CR-74)</td>
<td>changes the command to be executed for a specified menu item label, in the specified menu, in the specified window</td>
</tr>
<tr>
<td>disable_menu (CR-125)</td>
<td>disables the specified menu within the specified window; useful if you want to restrict access to a group of ModelSim features</td>
</tr>
<tr>
<td>disable_menuitem (CR-126)</td>
<td>disables a specified menu item within the specified menu_path of the specified window; useful if you want to restrict access to a specific ModelSim feature</td>
</tr>
<tr>
<td>enable_menu (CR-135)</td>
<td>enables a previously-disabled menu</td>
</tr>
<tr>
<td>enable_menuitem (CR-136)</td>
<td>enables a previously-disabled menu item</td>
</tr>
</tbody>
</table>
Customizing the interface

Try customizing ModelSim’s interface yourself; use the command examples for `add button` (CR-47) and `add_menu` (CR-52) to add a button to the Main window, and a new menu to the Signals window (UM-187). Results of the button and menu commands are shown below

**Buttons the easy way**

“The Button Adder” (UM-263) tool makes adding buttons easy. Select Window > Customize in any window to access the Button Adder. Buttons you create are not permanent; they exist only during the current session. To reuse a button, save the Main transcript (File > Save Transcript As) after the button is created. Edit the file to contain only button-creation commands, then pass the filename as an argument to the `do` command (CR-127) to recreate the button.

- The `pwd` button was added to the Main window with the `add button` command (CR-47). Buttons can be added to the status bar as well.
- The Mine menu was added to the Signals window with the `add_menu` command (CR-52).
- The Do My Own Thing menu item was added with the `add_menuitem` command (CR-55).
- The menu separator was added with the `add_separator` command (CR-56).
- The ChangeCase and Vars submenus were added with the `add_submenu` command (CR-57).
- You can also add a menu checkbox (like those in this menu tearoff) with the `add_menucb` command (CR-54).
You can use the Performance Analyzer to easily identify areas in your simulation where performance can be improved. The Performance Analyzer can be used at all levels of design simulation – Functional, RTL, and Gate Level – and has the potential to save hours of regression test time. In addition, ASIC and FPGA design flows benefit from the use of this tool.
Introducing Performance Analysis

The Performance Analyzer provides an interactive graphical representation of where ModelSim is spending its time while running your design. This feature enables you to quickly determine what is impacting the design environment’s simulation performance. Those familiar with the design and validation environment will be able to find first-level improvements in a matter of minutes.

For example, the Performance Analyzer might show some or all of the following:

- A non-accelerated VITAL library cell is impacting simulation run time
- A process is consuming more time than necessary because of non-required items in its sensitivity list
- A testbench process is active even though it is not needed
- A random number process is consuming simulation resources when in a testbench that is running in non-random mode

With this information, you can make changes to the VHDL or Verilog source code that will speed up the simulation.

A Statistical Sampling Profiler

The Performance Analyzer feature in ModelSim is a statistical sampling profiler. It periodically "wakes up" and samples the current simulation at a user-determined rate, and records what is executing in the simulation during the sample period. The advantage of statistical analysis is that an entire simulation may not have to be run to get good information from the Performance Analyzer. A few thousand samples, for example, can be accumulated before pausing the simulation to see where simulation time is being spent.

The Performance Analyzer reports only on the samples that it can attribute to user code. For example, if you used the -nodebug argument to vsim, it could not report sample results.
During sampling, the Samples field in the footer of the Main window displays the number of profiling samples collected, and each sample becomes one data point in the simulation profile.

### Getting Started

Performance analysis occurs during the ModelSim run command and is displayed graphically as a profile of simulator performance. To enable the Performance Analyzer, use the `profile on` command at the VSIM prompt. After this command is executed, all subsequent `run` commands will have profiling statistics gathered for them. With the Performance Analyzer enabled and a `run` command initiated, the simulator will provide a message indicating that profiling has started.

The Performance Analyzer is turned off by issuing the `profile off` command at the VSIM prompt. Any ModelSim `run` commands that follow will not be profiled.

Profiling results are cumulative. Therefore, each `run` command performed with profiling ON will add new information to the data being gathered. To clear this data, issue the `profile clear` command at the VSIM prompt.

### Interpreting the data

The Performance Analyzer is most helpful in those situations where a high percentage of simulation time is being spent in a particular module. For example, say the Performance Analyzer shows that the simulation is spending 60% of its time in module X. This information can be used to find where module X was implemented poorly and to implement a change that runs several times faster.

More commonly, the Performance Analyzer will tell you that 30% of simulation time was spent in model X, 25% in model Y, and 20% in model Z. In such situations, careful
examination and improvement of each model may result in a significant overall speed improvement.

There are times, however, when the Performance Analyzer tells you nothing better than that the simulation has executed in several hundred different models and has spent less than 1% of its time in any one of them. In such situations, the Performance Analyzer provides little helpful information and simulation improvement must come from a higher level examination of how the design can be changed or optimized.

Viewing Performance Analyzer Results

The Performance Analyzer provides two views of the collected data – a **hierarchical** and a **ranked** view. The hierarchical view is accessed by clicking **View > Other > Hierarchical Profile** (Main window). The ranked view is accessed by selecting **View > Other > Ranked Profile**.
The Hierarchical view can also be invoked by entering `view_profile` at the VSIM prompt. In the Hierarchical Profile window, you can expand and collapse various levels to hide data that is not useful and/or is cluttering the data display. Click on a the ‘−’ box to collapse all levels beneath the entry. Click on the ‘+’ box to expand an entry. By default, all levels are fully expanded. In the hierarchical view below, two lines (retrieve.vhd:35 and store.vhd:43) are taking the majority of the simulation time.

![Hierarchical Profile Table]

<table>
<thead>
<tr>
<th>Name</th>
<th>Under(%)</th>
<th>In(%)</th>
<th>%Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>retrieve.vhd:35</td>
<td>44</td>
<td>10</td>
<td>...</td>
</tr>
<tr>
<td>ieee_src/mti_std_logic_unsigned.vhd:429</td>
<td>34</td>
<td>34</td>
<td>77</td>
</tr>
<tr>
<td>store.vhd:43</td>
<td>43</td>
<td>9</td>
<td>...</td>
</tr>
<tr>
<td>ieee_src/mti_std_logic_unsigned.vhd:429</td>
<td>33</td>
<td>33</td>
<td>78</td>
</tr>
<tr>
<td>control.vhd:87</td>
<td>2</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>ieee_src/mti_std_logic_unsigned.vhd:276</td>
<td>1</td>
<td>1</td>
<td>56</td>
</tr>
<tr>
<td>retrieve.vhd:38</td>
<td>1</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>testing.vhd:99</td>
<td>1</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>control.vhd:98</td>
<td>1</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>ieee_src/mti_std_logic_unsigned.vhd:424</td>
<td>1</td>
<td>1</td>
<td>76</td>
</tr>
<tr>
<td>store.vhd:46</td>
<td>1</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>testing.vhd:97</td>
<td>1</td>
<td>1</td>
<td>...</td>
</tr>
</tbody>
</table>
The Ranked view can be invoked by entering **view_profile_ranked**. The modules and code lines are ranked in order of the amount of simulation time used. The two lines that are taking up most of the simulation time – retrieve.vhd:35 and store.vhd:43 – appear at the top of the list under the VHDL module that contains them.

<table>
<thead>
<tr>
<th>Name</th>
<th>Under(%)</th>
<th>In(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ieee_src/mti_std_logic_unsigned.vhd:429</td>
<td>67</td>
<td>67</td>
</tr>
<tr>
<td>retrieve.vhd:35</td>
<td>44</td>
<td>10</td>
</tr>
<tr>
<td>store.vhd:43</td>
<td>43</td>
<td>9</td>
</tr>
<tr>
<td>ieee_src/mti_std_logic_unsigned.vhd:276</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ieee_src/mti_std_logic_unsigned.vhd:424</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>retrieve.vhd:38</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>testing.vhd:99</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>store.vhd:46</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>testing.vhd:97</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Interpreting the Name Field**

The Name, Under(%) and In(%) fields appear in both the ranked and hierarchical views. These fields are interpreted identically in both views. Typically a Name consists of an HDL file and line number pair. Most useful names consist of a line of VHDL or Verilog source code. If you use a PLI/VPI or FLI routine, then the name of the C function that implements that routine can also appear in the name field.

vsim is a stripped executable file, so that any functions inside of it will be credited to the line of code that uses the function.

The hierarchical view opens with all levels displayed. You can collapse the hierarchical view by clicking the boxes next to the high-level names. At this time, the hierarchical view will not remember which levels are opened or closed when data is reloaded. By default, hierarchical levels are opened every time data is reloaded.

**Interpreting the Under(%) and In(%) Fields**

The In(%) and Under(%) columns describe the percentage of the total simulation time spent in and under a function listed in the Name field.

The distinction between In(%) and Under(%) is subtle but important. For the retrieve.vhd:35 entry in the hierarchical and ranked views above, Under(%) is 44 and In(%) is 10. "Under(%)" means that this particular line and all support routines it needed took 44% of total simulation time. "In(%)" means that 10% of the total simulation time was actually spent executing this line of VHDL code.
In the body of the Hierarchical Profile or Ranked Profile windows, you can double-click on any VHDL/Verilog file and line-number pair to bring up that file in the Source Window with the selected line highlighted. In the diagram below, *retrieve.vhd:35* was selected in the Hierarchical Profile and, consequently, is highlighted in the Source window.

The actual line of VHDL code for *retrieve.vhd:35* is:

```
IF (i=ramadrs((counter_size-1)downto 0))THEN
```

**Differences in the Ranked and Hierarchical Views**

The hierarchical view differs from the ranked view in two important respects.

- Entries in the Name column of the hierarchical view are indented in order to show which functions or routines call which others.

- A `%Parent` column in the hierarchical view allows you to see what percentage of a parent routine’s simulation time is used in which subroutines.

Indentation in the Name column of the Hierarchical Profile window indicates which line is calling a function. For example, in the hierarchical view above, the line *store.vhd:43* calls *ieee_src/mti_std_logic_unsigned.vhd:429*.

The hierarchical view presents data in a call-graph style format that provides more context than the ranked view about where simulation time is spent. For example, your models may contain several instances of a utility function that compute the maximum of 3-delay values. A ranked view might reveal that the simulation spent 60% of its time in this utility function, but would not tell you which routine or routines were making the most use of it. The hierarchical view will reveal which line is calling the function most frequently. Using this information, you might decide that instead of calling the function every time to compute the maximum of the 3-delays, this spot in your VHDL code can be used to compute it just once. You can then store the maximum delay value in a local variable.

The `%Parent` column provides the percent of simulation time a given entry used of its parent’s total simulation time. From this column, you can calculate the percentage of total simulation time taken up by any function. For example, if a particular parent entry used
10% of the total simulation time, and it called a routine that used 80% of its simulation time, then the percentage of total simulation time spent in that routine would be 80% of 10%, or 8%.

In addition to these differences, the ranked view displays any particular function only once, regardless of where it was used. In the hierarchical view, the function can appear multiple times – each time in the context of where it was used.

**Ranked/Hierarchical Profile Window Features**

The Ranked and Hierarchical Profile windows have a number of features that can manipulate the data displayed. Most of these features are contained in a toolbar in the header of the window, which displays an icon for each feature. Placing the mouse over an icon causes its function to be displayed.

The **Find Entry** icon provides access to a search function that can be used to search for a given string in the window. Type text in the entry box and then press Return or click the binocular icon.

The **Under%** filter allows you to specify a cutoff percentage for displaying the data. By default, every entry in the profiling data that has spent at least 1% of the simulation time under that entry will be displayed.

The **hierCutoff** and **rankCutoff** variables provide a similar function. See "Performance Analyzer preference variables" (UM-...)

The **Update Data** icon causes the data to be reloaded from the simulator. If you change the cutoff percentage or do an additional simulation run the Ranked and Hierarchical Profile windows are not automatically updated. You should click on this button to update the data being displayed in these windows.

The **Save to File** icon allows the data to be saved to disk. You will be prompted for the output file name. The **profile report** command (CR-181) provides another way to save profile data.
The report option

You can also use the **profile report** command (CR-181) to save the Performance Analyzer results.

```plaintext
profile report [<option>]
```

The arguments to the command are [-hierarchical | -ranked] [-file<filename>] [-cutoff <percentage>]. For example, the command

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

will produce a profile report in a text file called *hier.rpt*, as shown here.

![Hierarchical profile report](attachment:profile_report.png)
Performance Analyzer preference variables

Various Tcl variables control how the Hierarchical Profile and Ranked Profile windows are displayed. You can set these preference variables by selecting Options > Edit Preferences > By Name > Profile (Main window). Use the Apply button to view temporary changes, or Save the changes to a local modelsim.tcl file. Once saved, the preferences will be the default for subsequent simulations invoked from the same directory. See http://www.model.com/resources/pref_variables/frameset.htm for more information on the individual variables.

Performance Analyzer commands

The table below provides a brief description of the profile commands; follow the links for complete command syntax.

See the ModelSim Command Reference for complete command details.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>profile clear</strong> (CR-176)</td>
<td>clears any data that has been gathered during previous run commands; after this command is executed, all profiling data will be reset</td>
</tr>
<tr>
<td><strong>profile interval</strong> (CR-177)</td>
<td>selects the frequency with which the profiler collects samples during a run command</td>
</tr>
<tr>
<td><strong>profile off</strong> (CR-178)</td>
<td>disables runtime profiling</td>
</tr>
<tr>
<td><strong>profile on</strong> (CR-179)</td>
<td>enables runtime analysis of where your simulation is spending its time</td>
</tr>
<tr>
<td><strong>profile option</strong> (CR-180)</td>
<td>changes various profiling options</td>
</tr>
<tr>
<td><strong>profile report</strong> (CR-181)</td>
<td>produces a textual output of the profiling statistics that have been gathered up to this point</td>
</tr>
</tbody>
</table>
Code Coverage gives you graphical and report file feedback on how your source code is being executed. This integrated feature provides three important benefits to the ModelSim user:

1. Because it’s integrated into the ModelSim engine, it is totally non-intrusive – it doesn’t require instrumented HDL code as do third-party code coverage products.

2. It has very little impact on simulation performance (typically less than 5%).

3. There is no need to recompile to obtain code coverage statistics. ModelSim version 5.3 and later libraries fully support this feature.
Enabling Code Coverage

To enable code coverage, begin simulation with the -coverage option to the vsim command (CR-284). With coverage enabled, ModelSim counts how many times each executable line is executed during simulation (number of "hits"). The information is then displayed in the coverage_source and coverage_summary windows. Or, you can save the information in several different text reports (see below for details).

- **Note**: To view the maximum number of lines while doing code coverage, use the -O0 (capital O zero) argument when you compile your design files. This argument minimizes compiler optimizations.

The coverage_summary window

The coverage_summary window provides a graphical view of code coverage. To display the coverage_summary window, select View > Other > Source Coverage (Main window) or enter view_coverage at the VSIM prompt.
The coverage_summary window is split into two panes: the top pane displays **Summary information** (UM-287) on a per file basis; the bottom pane displays lines misses on the **Misses tab** (UM-287) and file or line exclusions on the **Exclusions tab** (UM-287).

The coverage_summary window is linked to **The coverage_source window** (UM-290). When you select a file in the top pane, that file displays in the coverage_source window. Likewise, if you select a line number in the bottom pane, that line is scrolled to in the coverage_source window. In addition, any exclusions you make in the coverage_summary window automatically show up in the coverage_source window and vice versa.

### Summary information

The top pane of the coverage_summary window shows all of the design files that have executable lines of code. The columns of information include:

- The Pathname column shows the path and file name.
- The Lines column contains the number of executable lines in the file.
- The Hits column indicates the number of executable lines that have been executed in the current simulation.
- The Percentage column is the current ratio of Hits to Lines. There is also a bar chart that graphically displays this percentage. If the coverage percentage is below 90%, the bar chart is displayed in red (you can change the percentage by editing the PrefCoverage(cutoff) preference variable).

By default, the summary information is sorted by Pathname. You can sort by another column by clicking on the column heading (i.e., Lines, Hits, %).

A totals row at the bottom of the summary information shows coverage statistics for all of the files combined.

### Misses tab

The Misses tab lists lines from the current file with no hits. Select a file in the top pane of the coverage_summary window to see that file’s missed lines.

This tab also lets you select lines to exclude. Select the line(s) you want to exclude, click your right mouse button, and select **Exclude Selected Lines**. The lines you exclude will be shown in the Exclusions tab and also marked with a green "X" in **The coverage_source window** (UM-290).

### Exclusions tab

The Exclusions tab lists all file and line exclusion filters for the current simulation. This includes line or file exclusions made in the Misses tab or in the coverage_source window.

The Exclusions tab offers several commands via a context menu. Click anywhere within the tab with your right mouse button to get the following context menu:
The menu has the following options:

• **Include Entire Selected Files**
  Adds selected lines or files back into the coverage statistics. If you have multiple lines excluded in one file, it will add back all of them. To add back individual lines, use the coverage_source window.

• **Revert To Initial Filter**
  Returns filtering to the default exclusion filter file

• **Clear Out Current Filter**
  Clears active exclusion filters

• **Load a New Filter**
  Opens a different exclusion filter file

• **Disable/Enable Filtering**
  Disables/enables filtering. Acts as a toggle. Allows you to temporarily turn off filtering to see raw code coverage statistics.

• **Cancel**
  Closes the context menu

### The coverage_summary window menu bar

The coverage_summary window has three menus: File, Coverage, and Report. Brief descriptions of each command are given below.

#### File menu

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open &gt; Coverage &gt; Merge Coverage</td>
<td>Merges saved reports into the current analysis. See &quot;Merging coverage report files&quot; (UM-292) for more details</td>
</tr>
<tr>
<td>Open &gt; Coverage &gt; Apply a Previous Coverage</td>
<td>Clears the current coverage statistics and loads a previously saved coverage report</td>
</tr>
<tr>
<td>Open &gt; Load a New Filter</td>
<td>Loads an exclusion filter file. See &quot;Exclusion filter files&quot; (UM-293) for more details</td>
</tr>
<tr>
<td>Save &gt; Line Coverage</td>
<td>Saves a textual report of the source file summary data and details for each executable line in the file</td>
</tr>
<tr>
<td>Save &gt; Current Filter</td>
<td>Saves the current exclusion filter to a file that can be reloaded later. See &quot;Exclusion filter files&quot; (UM-293) for more details</td>
</tr>
<tr>
<td>Close</td>
<td>Closes the view_coverage window</td>
</tr>
</tbody>
</table>
The coverage_summary window

**Coverage menu**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Current Coverage</td>
<td>Clears the current coverage statistics</td>
</tr>
<tr>
<td>Revert To Initial Filter</td>
<td>Returns filtering to the default exclusion filter file</td>
</tr>
<tr>
<td>Clear out Current Filter</td>
<td>Clears active exclusion filters</td>
</tr>
</tbody>
</table>

**Report menu**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save Summary Coverage</td>
<td>Saves a textual report of the summary lines, hits, and percentages for each source file being analyzed</td>
</tr>
<tr>
<td>Save Line Coverage</td>
<td>Saves a textual report of the source file summary data and details for each executable line in the file</td>
</tr>
<tr>
<td>Save Excluded Lines</td>
<td>Saves a textual report of the lines and files that are currently being excluded from the coverage statistics</td>
</tr>
<tr>
<td>Save Zeroed Lines</td>
<td>Saves a textual report like the Line Coverage report but only includes those lines that have zero coverage</td>
</tr>
<tr>
<td>Save Totals</td>
<td>Saves a one line text report of the total files, lines, hits and overall percentage for the current analysis</td>
</tr>
<tr>
<td>Save As</td>
<td>Lets you choose from the above reports in one dialog</td>
</tr>
</tbody>
</table>
The coverage_source window

You can open the coverage_source window by selecting a file in the pathname column of "The coverage_summary window" (UM-286). The coverage_source window is an enhanced version of the standard Source window (UM-195). When code coverage is enabled, an additional column appears on the left side of the window. This column identifies how many times each executable line of code has been executed during simulation (lines that are not executed are highlighted with a red zero); and it marks with a green "X" lines that have been excluded from the code coverage statistics.

You can skip to "missed lines" using the Edit > Previous Coverage Miss and Edit > Next Coverage Miss commands, or by pressing <Shift> - <Tab> (previous miss) or Tab (next miss).
Excluding lines and files

There may be certain lines or files that you do not want to include in the code coverage statistics. In the coverage_source window, click your right mouse button in the far-left column (the one with the hit counts) to display the following context menu:

The menu has the following options:

- **Exclude Coverage Line #**
  Excludes the specified line number from the code coverage statistics.

- **Exclude Entire File**
  Excludes the entire file from the code coverage statistics.

- **Do Not Exclude Coverage Line #**
  Adds the specified line number back into the code coverage statistics.

- **Do Not Exclude Entire File**
  Adds the file back into the code coverage statistics.

Any exclusions you make in the coverage_source window will show up in the Excluded tab of The coverage_summary window (UM-286).
Merging coverage report files

You can merge the results from two or more analyses. Select File > Open > Coverage > Merge Coverage from the coverage_summary window.

The Merge Coverage Reports dialog has the following options:

- **Coverage File Name to Read From**
  Specify one or more saved coverage reports that you want to merge into the current analysis

- **Clear out accumulated coverage data**
  When checked, clears coverage statistics from the current analysis before merging in saved coverage reports

- **Keep coverage data for files not in the current design**
  When checked, includes coverage data from all files you are merging in, even if they are not part of the current design. If you then select one of those included files in the coverage_source window, it will pop-up an Open Source dialog so you can point to the location of the file.
Exclusion filter files

Exclusion filter files specify files and line numbers that you wish to exclude from the coverage statistics. You can create the filter file in any text editor or save the current filter in the coverage_source window by selecting File > Save > Current Filter. To load the filter during a future analysis, select File > Open > Load a New Filter.

Syntax

```
<filename> [[<range> ...] [<line#> ...]] | all
```

Arguments

<filename>
The name of the file you want to exclude. Required. The filter file may include an unlimited number of filename entries, each on its own line.

<range>, ...
A range of line numbers you want to exclude. Optional. Enter the range in "# - #" format. For example, 32 - 35. You can specify multiple ranges separated by spaces.

<lline#>, ...
A line number that you want to exclude. Optional. You can specify multiple line numbers separated by spaces.

all
Specifies that all lines in the file should be excluded. Required if a range or line number is not specified.

Example

```
control.vhd 72 - 76 84 93
testring.vhd all
```

Default filter file

The Tcl preference variable `PrefCoverage(pref_InitFilterFrom)` specifies a default filter file and path to read when a design is loaded with the -coverage switch. By default this variable is set to "Exclude.cov". See "Code Coverage preference variables" (UM-294) for details on changing this variable.
Code Coverage preference variables

Various Tcl variables control how the coverage data is displayed. You can set these preference variables by selecting Options > Edit Preferences > By Name > Coverage (Main window). Use the Apply button to view temporary changes, or Save the changes to a local modelsim.tcl file. Once saved, the preferences will be the default for subsequent simulations invoked from the same directory. See http://www.model.com/resources/pref_variables/frameset.htm for more information on the individual variables.

Code Coverage commands

The commands below are available once Code Coverage is active. Enable code coverage with the -coverage option to the vsim command (CR-284).

The table below provides a brief description of the coverage commands; follow the links for complete command syntax.

See the ModelSim Command Reference for complete command details.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coverage clear  (CR-112)</td>
<td>clears all coverage data obtained during previous run commands</td>
</tr>
<tr>
<td>coverage reload (CR-113)</td>
<td>merges coverage statistics with the output of a previous coverage report command</td>
</tr>
<tr>
<td>coverage report (CR-114)</td>
<td>used to produce a textual output of the coverage statistics that have been gathered up to this point</td>
</tr>
</tbody>
</table>
# 11 - Waveform Comparison

## Chapter contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>UM-296</td>
</tr>
<tr>
<td>Two Modes of Comparison</td>
<td>UM-297</td>
</tr>
<tr>
<td>Comparing Hierarchical and Flattened Designs</td>
<td>UM-297</td>
</tr>
<tr>
<td>Graphical Interface to Waveform Comparison</td>
<td>UM-299</td>
</tr>
<tr>
<td>Opening Dataset Comparison</td>
<td>UM-299</td>
</tr>
<tr>
<td>Adding Signals, Regions and/or Clocks</td>
<td>UM-301</td>
</tr>
<tr>
<td>Setting Compare Options</td>
<td>UM-308</td>
</tr>
<tr>
<td>Wave window display</td>
<td>UM-310</td>
</tr>
<tr>
<td>Printing compare differences</td>
<td>UM-315</td>
</tr>
<tr>
<td>List window display</td>
<td>UM-316</td>
</tr>
<tr>
<td>Waveform Comparison preference variables</td>
<td>UM-317</td>
</tr>
<tr>
<td>Waveform Comparison commands</td>
<td>UM-317</td>
</tr>
</tbody>
</table>

ModelSim SE User's Manual
Introduction

The ModelSim Waveform Comparison feature allows you to compare the current live simulation against a reference dataset (.wlf file), compare two datasets, or compare different parts of the current live simulation. You can view the results of these comparisons in the Wave and List windows and generate a text file of the results in the Main window.

With the Waveform Comparison feature you can:

- specify the signals or regions to be compared,
- define tolerances for timing differences,
- set a start time and end time for the comparison,
- limit the comparison to a specific number of timing differences, and
- step through a succession of timing differences via buttons in the Wave window.

By default, Waveform Comparison computes the timing differences between test signals and reference signals from time zero to the end of the shortest dataset, or to the end of the current live simulation. But you can also specify an optional start time and end time, or you can limit the comparison to a specific number of encountered timing differences. In addition, you can exclude windows of time with -when conditions in either the clock definitions or in the compare add command (CR-84). The display will indicate intervals of time during which no attempt was made to compute differences.

All waveform differences encountered in the waveform comparison are summarized and listed in the transcript area of the Main window. Waveform differences are also displayed in the Wave and List windows (see Wave window display (UM-310) and List window display (UM-316)). Icons in the toolbar of the Wave window allow you to step forward and backward through successive differences. Or, you can use the Tab and Shift-Tab keys on your keyboard to move to the next or previous difference of a selected signal.

You can also write a list of the differences to a file using the compare info command (CR-94).
Two Modes of Comparison

The Waveform Comparison feature provides two modes of comparison: continuous and clocked.

**Continuous Compare**

In the continuous mode, a test signal (or a group of test signals within a region) is compared to a reference signal (or a group of reference signals within a region) at each transition of the reference. Timing differences between the test and reference signals are highlighted with rectangular red difference markers in the Wave window and yellow markers in the List window.

The continuous compare mode allows you to specify two edge tolerances for timing differences. The leading edge tolerance specifies how much earlier the test signal edge may occur before the reference signal edge. The trailing edge tolerance specifies how much later the test signal edge may occur after the reference signal edge. The default value for both tolerances is zero. In addition, these tolerances may be specified differently for each signal compared.

**Clocked Compare**

In the clocked mode, also called strobed comparison, one or more clocks are defined. A test signal is then compared to a reference signal and both are sampled relative to the defined clock. The clock can be defined as the rising or falling edge (or either edge) of a particular signal plus a user-specified delay. The design need not have any events occurring at the specified clock time.

Differences between the test signal(s) and clock are highlighted with red diamonds in the Wave window.
Comparing Hierarchical and Flattened Designs

If you are comparing a hierarchical RTL design simulation against a flattened synthesized design simulation, you may have different hierarchies, different signal names, and the buses may be broken down into one-bit signals in the gate-level design. All of these differences can be handled by ModelSim’s Waveform Comparison feature.

- If the test design is hierarchical but the hierarchy is different from the hierarchy of the reference design, you can use the `compare add` command (CR-84) to specify which region path in the test design corresponds to that in the reference design.

- If the test design is flattened and test signal names are different from reference signal names, the `compare add` command (CR-84) allows you to specify which signal in the test design will be compared to which signal in the reference design.

- If, in addition, buses have been dismantled, or "bit-blasted", you can use the `-rebuild` option of the `compare add` command (CR-84) to automatically rebuild the bus in the test design. This will allow you to look at the differences as one bus versus another.

If signals in the RTL test design are different in type from the synthesized signals in the reference design – registers versus nets, for example – the Waveform Comparison feature will automatically do the type conversion for you. If the type differences are too extreme (say integer versus real), Waveform Comparison will let you know.
Graphical Interface to Waveform Comparison

Waveform Comparison is initiated from either the Main or Wave window by selecting Compare > Start Comparison.

Opening Dataset Comparison

The Start Comparison dialog box allows you to define the Reference and Test datasets.

Reference Dataset

The Reference Dataset is the .wlf file that the test dataset will be compared to. It can be a saved dataset, the current simulation dataset, or any part of the current simulation dataset.

Test Dataset

The Test Dataset is the .wlf file that will be compared against the Reference Dataset. Like the Reference Dataset, it can be a saved dataset, the current simulation dataset, or any part of the current simulation dataset.

- **Use Current Simulation**
  Selects the current simulation to be used as the Test Dataset. Provides for an optional update on the comparison after each simulation run.

- **Specify Dataset**
  Allows you to select any saved .wlf file to be used as the Test Dataset.

You can specify either dataset by typing in a dataset name, by selecting a dataset from a drop-down history of past dataset selections, or by clicking either of the Browse buttons.
Both Browse buttons take you to the Select Dataset File dialog where you can browse for the dataset you want.

Once the Reference and Test Datasets have been specified, clicking "OK" in the Compare Dataset dialog box will place a Compare tab in the project pane of the Main window. After adding the signals, regions and/or clocks you want to use in the comparison (see "Adding Signals, Regions and/or Clocks" (UM-301)) you'll be able to drag compare objects from this project tab into the Wave and List windows.
Adding Signals, Regions and/or Clocks

To designate the signals, regions and/or clocks to be used in the comparison, click Compare > Add in the Main or Wave window, then make a selection (Compare by Signal (UM-301), Compare by Region (UM-305), Clocks) from the popup menu.

Compare by Signal
Clicking Compare > Add > Compare by Signal in the Wave window opens the structure_browser window, where you can specify signals to be used in the comparison.

You can also set signal options by clicking the Options button, which opens the Add Signal Options dialog box.
• **Add Signal Options**

The Add Signal Options dialog allows you to select the Waveform Comparison method to be used – Clocked (Strobed) or Continuous – and to specify a `when` expression that must evaluate to “true” or 1 at the signal edge for the clock to become effective. A `when` expression can be built using "The GUI Expression Builder" (UM-269), which is accessed by clicking the Builder button.

### Clocked Comparison

If the Clocked Comparison method is chosen, you can select a clock from the drop-down history of past clock selections or click the Clocks button to add a new clock.

Clicking the Clocks button opens the Comparison Clocks dialog box where you can add, modify or delete signals.
The Add button opens the Add Clock dialog, where you can define a clock signal name, a delay signal offset, the signal upon which the clock will be based, and whether the compare strobe edge will be the rising or falling edge or both. You can also use "The GUI Expression Builder" (UM-269) to specify a when expression that must evaluate to "true" or 1 at the signal edge for the clock to become effective.

Clicking the Modify button in the Comparison Clocks dialog opens the Modify Clock dialog. This dialog provides the same functionality as the Add Clock dialog.
Continuous Comparison

With the Continuous Comparison method you can set leading and trailing edge tolerances. The leading edge tolerance specifies how much earlier the test signal edge may occur before the reference signal edge. The trailing edge tolerance specifies how much later the test signal edge may occur after the reference signal edge. The default value for both tolerances is zero. In addition, these tolerances may be specified differently for each signal compared.

With Continuous Comparison, you can also use "The GUI Expression Builder" (UM-269) to specify a when expression that must evaluate to "true" or 1 at the signal edge for the clock to become effective.
**Compare by Region**

Clicking **Compare > Add > Compare by Region** in the Wave window opens the Add Comparison by Region window, where you can specify signals to be used in the comparison.

![Add Comparison by Region Window](image)

**Region Data Tab**

- **Reference Region**
  Allows you to specify the reference region that will be used in the comparison.

- **Test Region**
  Allows you to specify a test region that might have a different name from that of the reference region.

- **Compare Signals of Type**
  Allows you to specify that All Types of signals will be used in the comparison or only Selected Types (In, Out, InOut, Internal, or Port).

- **Recursive Search**
  Specifies whether to search for signals in the hierarchy below the selected region.
Comparison Method Tab

Allows you to select clocked or continuous comparison, and provides the capability to specify a "When" expression.

- Clocked Comparison
  Allows you can select a clock from the drop-down history of past clock selections. Or, you can click the Clocks button to add a new clock.

  Clicking the Clocks button opens the Comparison Clocks dialog box.

  To add a signal, click the Add button to open the Add Clock dialog box, where you can define a clock signal name, a delay signal offset, the signal upon which the clock will be based, and whether the compare strobe edge will be the rising or falling edge or both. You can also use the Expression Builder to Specify
a When Expression that must evaluate to "true" or 1 at the signal edge for the clock to become effective.

- **Continuous Comparison**
  With the Continuous Comparison method you can set leading and trailing edge tolerances. The leading edge tolerance specifies how much earlier the test signal edge may occur before the reference signal edge. The trailing edge tolerance specifies how much later the test signal edge may occur after the reference signal edge. The default value for both tolerances is zero. In addition, these tolerances may be specified differently for each signal compared.

- **Specify When Expression**
  Allows you to use "The GUI Expression Builder" (UM-269) to specify a when expression that must evaluate to "true" or 1 at the signal edge for the clock to become effective.
Setting Compare Options

Selecting Compare > Options in either the Main or Wave windows provides access to the Add Signal Options dialog box. This dialog is divided into two tabs – the General Options tab and the Comparison Method tab.

- **General Options**

  ![Add Signal Options Dialog](image)

  **Comparison Limit Count** — Allows you to limit the waveform comparison to a specific number of total differences and/or a specific number of differences per signal.

  **VHDL Matching** — Allows you to designate which VHDL signal values will match the VHDL X and Z values.

  **Verilog Matching** — Allows you to designate which Verilog signal values will match the Verilog X and Z values. It also allows you to ignore the strength of the Verilog signal and consider only logic values.

  **Save as Default** — Allows you to save all changes as the new default settings for subsequent waveform comparisons.

  **Reset to Default** — Resets all settings to original default values.

  **Automatically add comparisons to the wave window?** — Specifies whether new signal comparison objects are added automatically to the Wave window.
**Comparison Method**

- **Clocked (Strobed) Comparison** — Allows you to select a default reference clock signal via a selection history or a browse button.

- **Continuous Comparison** — Allows you to set leading and trailing edge tolerances for the waveform comparison. The leading edge tolerance specifies how much earlier the test signal edge may occur before the reference signal edge. The trailing edge tolerance specifies how much later the test signal edge may occur after the reference signal edge. The default value for both tolerances is zero.

- **Specify When Expression** — Allows you to specify a *when* expression that must evaluate to “true” or 1 at the signal edge for the clock to become effective. Clicking the Builder button will give you access to “The GUI Expression Builder” (UM-269).

- **Save as Default** — Allows you to save all changes as the new default settings for subsequent waveform comparisons.

- **Reset to Default** — Resets all settings to original default values.
Wave window display
The Wave window provides a graphic display of waveform comparison results. Pathnames of all test signals included in the waveform comparison are denoted by yellow triangles. Test signals that contain timing differences when compared with the reference signals are denoted by a red X over the yellow triangle.

The names of the comparison items take the form <path>/\refSignalName<>testSignalName>. If you compare two signals from different regions, the signal names include the uncommon part of the path.

Timing differences are also indicated by red bars in the vertical and horizontal scroll bars of the waveform display, and by red difference markers on the waveforms themselves. Rectangular difference markers denote continuous differences. Diamond difference markers denote clocked differences. Placing your mouse cursor over any difference marker will initiate a popup display that provides timing details for that difference. You can toggle this popup on and off in the Wave Window Properties dialog (see "Setting Wave window display properties" (UM-229)).

The values column of the Wave window displays the words "match" or "diff" for every test signal, depending on the location of the selected cursor. "Match" indicates that the value of the test signal matches the value of the reference signal at the time of the selected cursor. "Diff" indicates a difference between the test and reference signal values at the selected cursor.
Compare icons
The Wave window includes four waveform comparison icons that enable you to quickly locate the first and last waveform difference and move the cursor in steps to the previous or next difference. If you want to move between differences for the selected signal only, use <tab> (next) or <shift>-<tab> (previous).

Note: A comparison is independent from any window in which you view it. As a result, if you have two Wave windows displayed, each containing different comparison objects, the compare icons will cycle through the differences displayed in both windows.

Compare menu
The Compare menu provides a number of options for controlling waveform comparisons.

- **Start Comparison**
  Opens the Compare Dataset dialog box (page 11-299) where you can enter reference and test dataset names.

- **Comparison Wizard**
  Gives step-by-step assistance while you create a waveform comparison.
• **Run Comparison**
Computes the number of differences from time zero to the end of the simulation run, from time zero until the maximum total number of differences per signal limit is reached, or from time zero until the maximum total number of differences for all signals compared is reached. This information is posted to the Main window transcript and saved to the compare_info.txt file. It is equivalent to the `compare run` command (CR-101).

```
# write results to compare_info.txt
compare start
# Computing waveform differences from time 0 ps to 10 us
# =======================================================
# Max total difference per signal limit of 100 reached on signal compare
#:tst_pseudo/tol_min_exp_data
#
# Comparison reached signal difference limit at time 8080 ns
# Found 438 differences:
VSIM 3:
```

• **End Comparison**
Stops difference computation and closes the currently open comparison.

• **Add**

  **Compare by Signal** — Opens the structure_browser dialog box (page 11-301) and allows you to designate signals for comparison.

  **Compare by Region** — Opens the Add Comparison by Region dialog box (page 11-305) and allows you to designate a reference region for comparison. Also allows you to designate a test region of a different name.

  **Clocks** — Opens the Comparison Clocks dialog box (page 11-302) and allows you to define clocks to be used in the comparison.

• **Options**
Opens the Add Signal Options dialog box (page 11-308), which allows you to define a number of waveform comparison options.
• Differences

Clear — Clears all differences from the Wave window and resets the waveform comparison function. It is equivalent to the compare reset command (CR-100).

Show — Displays the difference in text format in the transcript area of the Main window. It is equivalent to the compare info command (CR-94).

Save — Opens the Specify Differences File dialog box where you can save the differences to a file that can be reloaded later in ModelSim. The default file name is "compare.dif".

Write Report — Saves a report of the differences to a text file that you can view.
Rules

Show — Displays the rules or instructions used to set up the waveform compare. It is equivalent to the compare list command (CR-95).

Save — Opens the Specify Rule File dialog box and allows you to assign a name to the file that will contain all rules for making the comparison. The default file name is "compare.rul."

Reload
Opens the Reload and Redisplay Compare Differences dialog box and allows you to enter or browse for waveform rules and difference file names.

Printing compare differences

You can print the compare differences shown in the Wave window either to a printer or to a Postscript file. See "Printing and saving waveforms" (UM-239) for details.
List window display

Compare objects can be displayed in the List window too. Differences are highlighted with a yellow background. Tabbing on selected columns moves the selection to the next difference (actually difference edge). Shift-tabbing moves the selection backwards.

Right-clicking on a yellow-highlighted difference gives you three options: Diff info, Annotate diff, and Ignore/Noignore diff. With these options you can elect to display difference information, you can ignore selected differences or turn off ignore, and you can annotate individual differences.
Waveform Comparison preference variables

Various Tcl variables control how the compare data is displayed. You can set these preference variables by selecting Options > Edit Preferences > By Name > Coverage (Main window). Use the Apply button to view temporary changes, or Save the changes to a local modelsim.tcl file. Once saved, the preferences will be the default for subsequent simulations invoked from the same directory. See http://www.model.com/resources/pref_variables/frameset.htm for more information on the individual variables.

Waveform Comparison commands

The table below provides a brief description of the compare commands. Follow the links for complete command syntax.

See ModelSim Commands for complete command details.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compare add (CR-84)</td>
<td>defines a comparison between the signals in a specified reference design and the signals in a specified test design</td>
</tr>
<tr>
<td>compare annotate (CR-87)</td>
<td>allows a difference to be flagged as ignore, or an additional text string to be attached</td>
</tr>
<tr>
<td>compare clock (CR-88)</td>
<td>defines a clock for clocked comparison; or, if -delete is specified, deletes a previously-defined clock</td>
</tr>
<tr>
<td>compare delete (CR-92)</td>
<td>deletes a signal or region from the current open comparison.</td>
</tr>
<tr>
<td>compare end (CR-93)</td>
<td>destroys the compare data structures and forgets clock definitions and signals selected for comparison</td>
</tr>
<tr>
<td>compare info (CR-94)</td>
<td>writes out results of the comparison; writes to the transcript unless the -write option is specified</td>
</tr>
<tr>
<td>compare list (CR-95)</td>
<td>shows all the compare region and compare signal commands currently in effect</td>
</tr>
<tr>
<td>compare options (CR-96)</td>
<td>sets values for various compare options on the Tcl parser side; when subsequent commands are called, these values become the defaults</td>
</tr>
<tr>
<td>compare reset (CR-100)</td>
<td>clears the current compare differences, allowing another compare start to be executed</td>
</tr>
<tr>
<td>compare reload (CR-99)</td>
<td>reloads comparison differences to allow viewing without recomputation</td>
</tr>
<tr>
<td>compare run (CR-101)</td>
<td>registers required callbacks and runs the difference computation on the signals selected for comparison; reports the total number of errors found</td>
</tr>
<tr>
<td>compare savediffs (CR-102)</td>
<td>saves the comparison result differences in a form that can be reloaded later</td>
</tr>
<tr>
<td>compare saverules (CR-103)</td>
<td>saves the comparison setup information (or &quot;rules&quot;) to a file that can be re-executed later as a command file; saves compare options and all clock definitions and region and signal selections</td>
</tr>
</tbody>
</table>
### Command Description

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>compare see</strong> command (CR-104)</td>
<td>causes the specified compare difference to be made visible in the specified wave window, using whatever horizontal and vertical scrolling is necessary</td>
</tr>
<tr>
<td><strong>compare start</strong> command (CR-105)</td>
<td>initializes internal data structures for waveform compare</td>
</tr>
<tr>
<td><strong>compare stop</strong> command (CR-107)</td>
<td>used internally by the <strong>compare stop</strong> button to suspend comparison computations in progress</td>
</tr>
<tr>
<td><strong>compare update</strong> command (CR-108)</td>
<td>used internally to update the comparison differences when comparing a live simulation against a .wlf file</td>
</tr>
</tbody>
</table>
This chapter discusses ModelSim’s implementation of SDF (Standard Delay Format) timing annotation. Included are sections on VITAL SDF and Verilog SDF, plus troubleshooting.

Verilog and VHDL VITAL timing data can be annotated from SDF files by using the simulator’s built-in SDF annotator. ASIC and FPGA vendors usually provide tools that create SDF files for use with their cell libraries. Refer to your vendor’s documentation for details on creating SDF files for your library. Many vendors also provide instructions on using their SDF files and libraries with ModelSim.

The SDF specification was originally created for Verilog designs, but it has also been adopted for VHDL VITAL designs. In general, the designer does not need to be familiar with the details of the SDF specification because the cell library provider has already supplied tools that create SDF files that match their libraries.

► **Note:** In order to conserve disk space, ModelSim will read sdf files that were compressed using the standard unix/gnu file compression algorithm. The filename must end with the suffix ".Z" for the decompress to work.
Specifying SDF files for simulation

ModelSim supports SDF versions 1.0 through 3.0. The simulator’s built-in SDF annotator automatically adjusts to the version of the file. Use the following `vsim` (CR-284) command-line options to specify the SDF files, the desired timing values, and their associated design instances:

- `sdfmin [instance]=<filename>`
- `sdftyp [instance]=<filename>`
- `sdfmax [instance]=<filename>`

Any number of SDF files can be applied to any instance in the design by specifying one of the above options for each file. Use `-sdfmin` to select minimum, `-sdftyp` to select typical, and `-sdfmax` to select maximum timing values from the SDF file.

Instance specification

The instance paths in the SDF file are relative to the instance to which the SDF is applied. Usually, this instance is an ASIC or FPGA model instantiated under a testbench. For example, to annotate maximum timing values from the SDF file `myasic.sdf` to an instance `u1` under a top-level named `testbench`, invoke the simulator as follows:

`vsim -sdfmax /testbench/u1=myasic.sdf testbench`

If the instance name is omitted then the SDF file is applied to the top-level. *This is usually incorrect* because in most cases the model is instantiated under a testbench or within a larger system level simulation. In fact, the design can have several models, each having its own SDF file. In this case, specify an SDF file for each instance. For example,

`vsim -sdfmax /system/u1=asic1.sdf -sdfmax /system/u2=asic2.sdf system`
SDF specification with the GUI

As an alternative to the command-line options, you can specify SDF files in the Load Design dialog box under the SDF tab.

You can access this dialog by invoking the simulator without any arguments or by selecting Design > Load Design (Main window). For Verilog designs, you can also specify SDF files by using the $sdf_annotate system task. See "The $sdf_annotate system task" (UM-324) for more details.

Errors and warnings

Errors issued by the SDF annotator while loading the design prevent the simulation from continuing, whereas warnings do not. Use the -sdfnoerror option with vsim (CR-284) to change SDF errors to warnings so that the simulation can continue. Warning messages can be suppressed by using vsim with either the -sdfnowarn or +nosdfwarn options.

Another option is to use the SDF tab from the Load Design dialog box (shown above). Select Disable SDF warnings (-sdfnowarn, or +nosdfwarn) to disable warnings, or select Reduce SDF errors to warnings (-sdfnoerror) to change errors to warnings.

See "Troubleshooting" (UM-331) for more information on errors and warnings, and how to avoid them.
VHDL VITAL SDF

VHDL SDF annotation works on VITAL cells only. The IEEE 1076.4 VITAL ASIC Modeling Specification describes how cells must be written to support SDF annotation. Once again, the designer does not need to know the details of this specification because the library provider has already written the VITAL cells and tools that create compatible SDF files. However, the following summary may help you understand simulator error messages. For additional VITAL specification information, see "Obtaining the VITAL specification and source code" (UM-60).

SDF to VHDL generic matching

An SDF file contains delay and timing constraint data for cell instances in the design. The annotator must locate the cell instances and the placeholders (VHDL generics) for the timing data. Each type of SDF timing construct is mapped to the name of a generic as specified by the VITAL modeling specification. The annotator locates the generic and updates it with the timing value from the SDF file. It is an error if the annotator fails to find the cell instance or the named generic. The following are examples of SDF constructs and their associated generic names:

<table>
<thead>
<tr>
<th>SDF construct</th>
<th>Matching VHDL generic name</th>
</tr>
</thead>
<tbody>
<tr>
<td>(IOPATH a y (3))</td>
<td>tpd_a_y</td>
</tr>
<tr>
<td>(IOPATH (posedge clk) q (1) (2))</td>
<td>tpd_clk_q_posedge</td>
</tr>
<tr>
<td>(INTERCONNECT u1/y u2/a (5))</td>
<td>tipd_a</td>
</tr>
<tr>
<td>(SETUP d (posedge clk) (5))</td>
<td>tsetup_d_clk_noedge_posedge</td>
</tr>
<tr>
<td>(HOLD (negedge d) (posedge clk) (5))</td>
<td>thold_d_clk_negedge_posedge</td>
</tr>
<tr>
<td>(SETUPHOLD d clk (5) (5))</td>
<td>tsetup_d_clk &amp; thold_d_clk</td>
</tr>
<tr>
<td>(WIDTH (COND (reset==1'b0) clk) (5))</td>
<td>tpw_clk_reset_eq_0</td>
</tr>
</tbody>
</table>
Resolving errors

If the simulator finds the cell instance but not the generic then an error message is issued. For example,

ERROR: myasic.sdf(18):
    Instance '/testbench/dut/u1' does not have a generic named 'tpd_a_y'

In this case, make sure that the design is using the appropriate VITAL library cells. If it is, then there is probably a mismatch between the SDF and the VITAL cells. You need to find the cell instance and compare its generic names to those expected by the annotator. Look in the VHDL source files provided by the cell library vendor.

If none of the generic names look like VITAL timing generic names, then perhaps the VITAL library cells are not being used. If the generic names do look like VITAL timing generic names but don’t match the names expected by the annotator, then there are several possibilities:

• The vendor’s tools are not conforming to the VITAL specification.

• The SDF file was accidentally applied to the wrong instance. In this case, the simulator also issues other error messages indicating that cell instances in the SDF could not be located in the design.

• The vendor’s library and SDF were developed for the older VITAL 2.2b specification. This version uses different name mapping rules. In this case, invoke \texttt{vsim} (CR-284) with the \texttt{-vital2.2b} option:

\texttt{vsim -vital2.2b -sdfmax /testbench/u1=myasic.sdf testbench}

For more information on resolving errors see “Troubleshooting” (UM-331).
Verilog SDF

Verilog designs can be annotated using either the simulator command-line options or the $sdf_annotate system task (also commonly used in other Verilog simulators). The command-line options annotate the design immediately after it is loaded, but before any simulation events take place. The $sdf_annotate task annotates the design at the time it is called in the Verilog source code. This provides more flexibility than the command-line options.

The $sdf_annotate system task

The syntax for $sdf_annotate is:

**Syntax**

$sdf_annotate

"<sdffile">[<instance>, [<config_file>, [<log_file>, [<mtm_spec>, [<scale_factor>, [<scale_type>]]]]];

**Arguments**

"<sdffile>"
String that specifies the SDF file. Required.

<instance>
Hierarchical name of the instance to be annotated. Optional. Defaults to the instance where the $sdf_annotate call is made.

"<config_file>"
String that specifies the configuration file. Optional. Currently not supported, this argument is ignored.

"<log_file>"
String that specifies the logfile. Optional. Currently not supported, this argument is ignored.

"<mtm_spec>"
String that specifies the delay selection. Optional. The allowed strings are "minimum", "typical", "maximum", and "tool_control". Case is ignored and the default is "tool_control". The "tool_control" argument means to use the delay specified on the command line by +mindelays, +typdelays, or +maxdelays (defaults to +typdelays).

"<scale_factor>"
String that specifies delay scaling factors. Optional. The format is "<min_mult>:<typ_mult>:<max_mult>". Each multiplier is a real number that is used to scale the corresponding delay in the SDF file.

"<scale_type>"
String that overrides the <mtm_spec> delay selection. Optional. The <mtm_spec> delay selection is always used to select the delay scaling factor, but if a <scale_type> is specified, then it will determine the min/typ/max selection from the SDF file. The allowed strings are "from_min", "from_minimum", "from_typ", "from_typical", "from_max", "from_maximum", and "from_mtm". Case is ignored, and the default is "from_mtm", which means to use the <mtm_spec> value.
Examples

Optional arguments can be omitted by using commas or by leaving them out if they are at the end of the argument list. For example, to specify only the SDF file and the instance it applies to:

```
$sdf_annotate("myasic.sdf", testbench.u1);
```

To also specify maximum delay values:

```
$sdf_annotate("myasic.sdf", testbench.u1, , , "maximum");
```

SDF to Verilog construct matching

The annotator matches SDF constructs to corresponding Verilog constructs in the cells. Usually, the cells contain path delays and timing checks within specify blocks. For each SDF construct, the annotator locates the cell instance and updates each specify path delay or timing check that matches. An SDF construct can have multiple matches, in which case each matching specify statement is updated with the SDF timing value. SDF constructs are matched to Verilog constructs as follows:

**IOPATH** is matched to specify path delays or primitives:

<table>
<thead>
<tr>
<th>SDF</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>(IOPATH (posedge clk) q (3) (4))</td>
<td>(posedge clk =&gt; q) = 0;</td>
</tr>
<tr>
<td>(IOPATH a y (3) (4))</td>
<td>buf u1 (y, a);</td>
</tr>
</tbody>
</table>

The IOPATH construct usually annotates path delays. If the module contains no path delays, then all primitives that drive the specified output port are annotated.

**INTERCONNECT** and **PORT** are matched to input ports:

<table>
<thead>
<tr>
<th>SDF</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>(INTERCONNECT u1.y u2.a (5))</td>
<td>input a;</td>
</tr>
<tr>
<td>(PORT u2.a (5))</td>
<td>inout a;</td>
</tr>
</tbody>
</table>

Both of these constructs identify a module input or inout port and create an internal net that is a delayed version of the port. This is called a Module Input Port Delay (MIPD). All primitives, specify path delays, and specify timing checks connected to the original port are reconnected to the new MIPD net.

**PATHPULSE** and **GLOBALPATHPULSE** are matched to specify path delays:

<table>
<thead>
<tr>
<th>SDF</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>(PATHPULSE a y (5) (10))</td>
<td>(a =&gt; y) = 0;</td>
</tr>
<tr>
<td>(GLOBALPATHPULSE a y (30) (60))</td>
<td>(a =&gt; y) = 0;</td>
</tr>
</tbody>
</table>

If the input and output ports are omitted in the SDF, then all path delays are matched in the cell.
**DEVICE** is matched to primitives or specify path delays:

<table>
<thead>
<tr>
<th>SDF</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>(DEVICE y (5))</td>
<td>and u1(y, a, b);</td>
</tr>
<tr>
<td>(DEVICE y (5))</td>
<td>(a =&gt; y) = 0; (b =&gt; y) = 0;</td>
</tr>
</tbody>
</table>

If the SDF cell instance is a primitive instance, then that primitive’s delay is annotated. If it is a module instance, then all specify path delays are annotated that drive the output port specified in the DEVICE construct (all path delays are annotated if the output port is omitted). If the module contains no path delays, then all primitives that drive the specified output port are annotated (or all primitives that drive any output port if the output port is omitted).

**SETUP** is matched to $setup and $setuphold:

<table>
<thead>
<tr>
<th>SDF</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>(SETUP d (posedge clk) (5))</td>
<td>$setup(d, posedge clk, 0);</td>
</tr>
<tr>
<td>(SETUP d (posedge clk) (5))</td>
<td>$setuphold(posedge clk, d, 0, 0);</td>
</tr>
</tbody>
</table>

**HOLD** is matched to $hold and $setuphold:

<table>
<thead>
<tr>
<th>SDF</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>(HOLD d (posedge clk) (5))</td>
<td>$hold(posedge clk, d, 0);</td>
</tr>
<tr>
<td>(HOLD d (posedge clk) (5))</td>
<td>$setuphold(posedge clk, d, 0, 0);</td>
</tr>
</tbody>
</table>

**SETUPHOLD** is matched to $setup, $hold, and $setuphold:

<table>
<thead>
<tr>
<th>SDF</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>(SETUPHOLD d (posedge clk) (5) (5))</td>
<td>$setup(d, posedge clk, 0);</td>
</tr>
<tr>
<td>(SETUPHOLD d (posedge clk) (5) (5))</td>
<td>$hold(posedge clk, d, 0);</td>
</tr>
<tr>
<td>(SETUPHOLD d (posedge clk) (5) (5))</td>
<td>$setuphold(posedge clk, d, 0, 0);</td>
</tr>
</tbody>
</table>

**RECOVERY** is matched to $recovery:

<table>
<thead>
<tr>
<th>SDF</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>(RECOVERY (negedge reset) (posedge clk) (5))</td>
<td>$recovery(negedge reset, posedge clk, 0);</td>
</tr>
</tbody>
</table>

**REMOVAL** is matched to $removal:

<table>
<thead>
<tr>
<th>SDF</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>(REMOVAL (negedge reset) (posedge clk) (5))</td>
<td>$removal(negedge reset, posedge clk, 0);</td>
</tr>
</tbody>
</table>
**RECREM** is matched to $recovery, $removal, and $recrem:

<table>
<thead>
<tr>
<th>SDF</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>(RECREM (negedge reset) (posedge clk) (5) (5))</td>
<td>$recovery(negedge reset, posedge clk, 0);</td>
</tr>
<tr>
<td>(RECREM (negedge reset) (posedge clk) (5) (5))</td>
<td>$removal(negedge reset, posedge clk, 0);</td>
</tr>
<tr>
<td>(RECREM (negedge reset) (posedge clk) (5) (5))</td>
<td>$recrem(negedge reset, posedge clk, 0);</td>
</tr>
</tbody>
</table>

**SKEW** is matched to $skew:

<table>
<thead>
<tr>
<th>SDF</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>(SKEW (posedge clk1) (posedge clk2) (5))</td>
<td>$skew(posedge clk1, posedge clk2, 0);</td>
</tr>
</tbody>
</table>

**WIDTH** is matched to $width:

<table>
<thead>
<tr>
<th>SDF</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>(WIDTH (posedge clk) (5))</td>
<td>$width(posedge clk, 0);</td>
</tr>
</tbody>
</table>

**PERIOD** is matched to $period:

<table>
<thead>
<tr>
<th>SDF</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>(PERIOD (posedge clk) (5))</td>
<td>$period(posedge clk, 0);</td>
</tr>
</tbody>
</table>

**NOCHANGE** is matched to $nochange:

<table>
<thead>
<tr>
<th>SDF</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>(NOCHANGE (negedge write) addr (5) (5))</td>
<td>$nochange(negedge write, addr, 0, 0);</td>
</tr>
</tbody>
</table>

**Optional edge specifications**

Timing check ports and path delay input ports can have optional edge specifications. The annotator uses the following rules to match edges:

- A match occurs if the SDF port does not have an edge.
- A match occurs if the specify port does not have an edge.
- A match occurs if the SDF port edge is identical to the specify port edge.
- A match occurs if explicit edge transitions in the specify port edge overlap with the SDF port edge.

These rules allow SDF annotation to take place even if there is a difference between the number of edge-specific constructs in the SDF file and the Verilog specify block. For example, the Verilog specify block may contain separate setup timing checks for a falling...
and rising edge on data with respect to clock, while the SDF file may contain only a single
setup check for both edges:

<table>
<thead>
<tr>
<th>SDF</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>(SETUP data (posedge clock) (5))</td>
<td>$setup(posedge data, posedge clk, 0);</td>
</tr>
<tr>
<td>(SETUP data (posedge clock) (5))</td>
<td>$setup(negedge data, posedge clk, 0);</td>
</tr>
</tbody>
</table>

In this case, the cell accommodates more accurate data than can be supplied by the tool that
created the SDF file, and both timing checks correctly receive the same value. Likewise,
the SDF file may contain more accurate data than the model can accommodate.

<table>
<thead>
<tr>
<th>SDF</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>(SETUP (posedge data) (posedge clock) (4))</td>
<td>$setup(data, posedge clk, 0);</td>
</tr>
<tr>
<td>(SETUP (negedge data) (posedge clock) (6))</td>
<td>$setup(data, posedge clk, 0);</td>
</tr>
</tbody>
</table>

In this case, both SDF constructs are matched and the timing check receives the value from
the last one encountered.

Timing check edge specifiers can also use explicit edge transitions instead of posedge and
negedge. However, the SDF file is limited to posedge and negedge. The explicit edge
specifiers are 01, 0x, 10, 1x, x0, and x1. The set of [01, 0x, x1] is equivalent to posedge,
while the set of [10, 1x, x0] is equivalent to negedge. A match occurs if any of the explicit
edges in the specify port match any of the explicit edges implied by the SDF port. For
example,

<table>
<thead>
<tr>
<th>SDF</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>(SETUP data (posedge clock) (5))</td>
<td>$setup(data, edge[01, 0x] clk, 0);</td>
</tr>
<tr>
<td>(SETUP data (COND (reset!=1) (posedge clock)) (5))</td>
<td>$setup(data, posedge clk &amp;&amp;&amp; (reset==0), 0);</td>
</tr>
</tbody>
</table>

Optional conditions

Timing check ports and path delays can have optional conditions. The annotator uses the
following rules to match conditions:

- A match occurs if the SDF does not have a condition.
- A match occurs for a timing check if the SDF port condition is semantically equivalent
to the specify port condition.
- A match occurs for a path delay if the SDF condition is lexically identical to the specify
  condition.

Timing check conditions are limited to very simple conditions, therefore the annotator can
match the expressions based on semantics. For example,
The conditions are semantically equivalent and a match occurs. In contrast, path delay conditions may be complicated and semantically equivalent conditions may not match. For example,

<table>
<thead>
<tr>
<th>SDF</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>(COND (r1</td>
<td></td>
</tr>
<tr>
<td>(COND (r1</td>
<td></td>
</tr>
</tbody>
</table>

The annotator does not match the second condition above because the order of r1 and r2 are reversed.

**Rounded timing values**

The SDF TIMESCALE construct specifies time units of values in the SDF file. The annotator rounds timing values from the SDF file to the time precision of the module that is annotated. For example, if the SDF TIMESCALE is 1ns and a value of .016 is annotated to a path delay in a module having a time precision of 10ps (from the timescale directive), then the path delay receives a value of 20ps. The SDF value of 16ps is rounded to 20ps. Interconnect delays are rounded to the time precision of the module that contains the annotated MIPD.
SDF for Mixed VHDL and Verilog Designs

Annotation of a mixed VHDL and Verilog design is very flexible. VHDL VITAL cells and Verilog cells can be annotated from the same SDF file. This flexibility is available only by using the simulator’s SDF command-line options. The Verilog $sdf_annotate system task can annotate Verilog cells only. See the `vsim` command (CR-284) for more information on SDF command-line options.

Interconnect delays

An interconnect delay represents the delay from the output of one device to the input of another. With Verilog designs, ModelSim can model single interconnect delays or multisource interconnect delays. See "Arguments, Verilog" (CR-291) under the `vsim` command for more information on the relevant command-line switches.

Per VHDL VITAL ’95, there is no convenient way to handle interconnect delays from multiple outputs to a single input. Interconnect delay is modeled in the receiving device as a single delay from an input port to an internal node. (The node is explicitly declared.) The default is to use the value of the maximum encountered delay in the SDF file. Alternatively, you can choose the minimum or latest value of the multiple delays with the `vsim` command (CR-284) `-multisource_delay` option.

    -multisource_delay min|max|latest

Timing checks are performed on the interconnect delayed versions of input ports. This may result in misleading timing constraint violations, because the ports may satisfy the constraint while the delayed versions may not. If the simulator seems to report incorrect violations, be sure to account for the effect of interconnect delays.
Troubleshooting

Specifying the wrong instance

By far, the most common mistake in SDF annotation is to specify the wrong instance to the simulator’s SDF options. The most common case is to leave off the instance altogether, which is the same as selecting the top-level design unit. This is generally wrong because the instance paths in the SDF are relative to the ASIC or FPGA model, which is usually instantiated under a top-level testbench. See “Instance specification” (UM-320) for an example.

A common example for both VHDL and Verilog test benches is provided below. For simplicity, the test benches do nothing more than instantiate a model that has no ports.

**VHDL testbench**

```vhdl
entity testbench is end;

architecture only of testbench is
  component myasic
    end component;
begin
  dut : myasic;
end;
```

The name of the model is *myasic* and the instance label is *dut*. For either testbench, an appropriate simulator invocation might be:

```
vsim -sdfmax /testbench/dut=myasic.sdf testbench
```

Optionally, you can leave off the name of the top-level:

```
vsim -sdfmax /dut=myasic.sdf testbench
```

The important thing is to select the instance for which the SDF is intended. If the model is deep within the design hierarchy, an easy way to find the instance name is to first invoke the simulator without SDF options, open the structure window, navigate to the model instance, select it, and enter the environment command (CR-137). This command displays the instance name that should be used in the SDF command-line option.
Mistaking a component or module name for an instance label

Another common error is to specify the component or module name rather than the instance label. For example, the following invocation is wrong for the above testbenches:

```plaintext
vsim -sdfmax /testbench/myasic=myasic.sdf testbench
```

This results in the following error message:

```
ERROR: myasic.sdf:
The design does not have an instance named '/testbench/myasic'.
```

Forgetting to specify the instance

If you leave off the instance altogether, then the simulator issues a message for each instance path in the SDF that is not found in the design. For example,

```plaintext
vsim -sdfmax myasic.sdf testbench
```

Results in:

```
ERROR: myasic.sdf:
Failed to find INSTANCE '/testbench/u1'
ERROR: myasic.sdf:
Failed to find INSTANCE '/testbench/u2'
ERROR: myasic.sdf:
Failed to find INSTANCE '/testbench/u3'
ERROR: myasic.sdf:
Failed to find INSTANCE '/testbench/u4'
ERROR: myasic.sdf:
Failed to find INSTANCE '/testbench/u5'
WARNING: myasic.sdf:
This file is probably applied to the wrong instance.
WARNING: myasic.sdf:
Ignoring subsequent missing instances from this file.
```

After annotation is done, the simulator issues a summary of how many instances were not found and possibly a suggestion for a qualifying instance:

```
WARNING: myasic.sdf:
Failed to find any of the 358 instances from this file.
WARNING: myasic.sdf:
Try instance '/testbench/dut' - it contains all instance paths from this file.
```

The simulator recommends an instance only if the file was applied to the top-level and a qualifying instance is found one level down.

Also see "Resolving errors" (UM-323) for specific VHDL VITAL SDF troubleshooting.
This chapter explains Model Technology’s Verilog VCD implementation for ModelSim.

The VCD file format is specified in the IEEE 1364 standard. It is an ASCII file containing header information, variable definitions, and variable value changes. VCD is in common use for Verilog designs, and is controlled by VCD system task calls in the Verilog source code. ModelSim provides simulator command equivalents for these system tasks and extends VCD support to VHDL designs; the ModelSim commands can be used on either VHDL or Verilog designs.

Note: If you need vendor-specific ASIC design-flow documentation that incorporates VCD, please contact your ASIC vendor.
ModelSim VCD commands and VCD tasks

ModelSim VCD commands map to IEEE Std 1364 VCD system tasks and appear in the VCD file along with the results of those commands. The table below maps the VCD commands to their associated tasks.

<table>
<thead>
<tr>
<th>VCD commands</th>
<th>VCD system tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>vcd add (CR-221)</td>
<td>$dumpvars</td>
</tr>
<tr>
<td>vcd checkpoint (CR-222)</td>
<td>$dumpall</td>
</tr>
<tr>
<td>vcd file (CR-231) ▲</td>
<td>$dumpfile</td>
</tr>
<tr>
<td>vcd flush (CR-235)</td>
<td>$dumpflush</td>
</tr>
<tr>
<td>vcd limit (CR-236)</td>
<td>$dumplimit</td>
</tr>
<tr>
<td>vcd off (CR-237)</td>
<td>$dumpoff</td>
</tr>
<tr>
<td>vcd on (CR-238)</td>
<td>$dumpon</td>
</tr>
</tbody>
</table>

ModelSim versions 5.5 and later support multiple VCD files. This functionality is an extension of the IEEE Std 1364 specification. The tasks behave the same as the IEEE equivalent tasks such as $dumpfile, $dumpvar, etc. The difference is that $fdumpfile can be called multiple times to create more than one VCD file, and the remaining tasks require a filename argument to associate their actions with a specific file.

<table>
<thead>
<tr>
<th>VCD commands</th>
<th>VCD system tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>vcd add (CR-221) -file &lt;filename&gt;</td>
<td>$fdumpvars</td>
</tr>
<tr>
<td>vcd checkpoint (CR-222) &lt;filename&gt;</td>
<td>$fdumpall</td>
</tr>
<tr>
<td>vcd files (CR-233) &lt;filename&gt; ▲</td>
<td>$fdumpfile</td>
</tr>
<tr>
<td>vcd flush (CR-235) &lt;filename&gt;</td>
<td>$fdumpflush</td>
</tr>
<tr>
<td>vcd limit (CR-236) &lt;filename&gt;</td>
<td>$fdumplimit</td>
</tr>
<tr>
<td>vcd off (CR-237) &lt;filename&gt;</td>
<td>$fdumpoff</td>
</tr>
<tr>
<td>vcd on (CR-238) &lt;filename&gt;</td>
<td>$fdumpon</td>
</tr>
</tbody>
</table>

▲ Important: Note that two commands (vcd file and vcd files) are available to specify a filename and state mapping for a VCD file. Vcd file allows for only one VCD file and exists for backwards compatibility with ModelSim versions prior to 5.5. Vcd files allows for creation of multiple VCD files and is the preferred command to use in ModelSim versions 5.5 and later.
ModelSim versions 5.5 and later also support dumpports system tasks. The table below maps the VCD dumpports commands to their associated tasks.

<table>
<thead>
<tr>
<th>VCD dumpports commands</th>
<th>VCD system tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>vcd dumpports (CR-224)</td>
<td>$dumpports</td>
</tr>
<tr>
<td>vcd dumpportsall (CR-226)</td>
<td>$dumpportsall</td>
</tr>
<tr>
<td>vcd dumpportsflush (CR-227)</td>
<td>$dumpportsflush</td>
</tr>
<tr>
<td>vcd dumpportslimit (CR-228)</td>
<td>$dumpportslimit</td>
</tr>
<tr>
<td>vcd dumpportsoff (CR-229)</td>
<td>$dumpportsoff</td>
</tr>
<tr>
<td>vcd dumpportson (CR-230)</td>
<td>$dumpportson</td>
</tr>
</tbody>
</table>
Resimulating a design from a VCD file

Note: The following methodology applies only to ModelSim versions 5.5c and later. See Resimulating a VHDL design from a VCD file for the methodology that applies to earlier versions.

To resimulate with a VCD file, you capture the ports of a design unit instance within a testbench or design. The design may be VHDL, Verilog, or mixed HDL. You can resimulate only at the top level of the module for which you captured ports.

The general procedure for resimulating with a VCD file includes two steps:

1. Create a VCD file using the vcd dumpports command (CR-224).

2. Rerun without the testbench, using the -vcdstim argument to vsim (CR-284).

Example 1 — Verilog counter

First, create the VCD file using vcd dumpports:

```
% cd ~/modeltech/examples
% vlib work
% vlog counter.v tcounter.v
% vsim test_counter
VSIM 1> vcd dumpports -file counter.vcd /test_counter/dut/*
VSIM 2> run
VSIM 3> quit -f
```

Next, rerun the counter without the testbench, using the -vcdstim argument:

```
% vsim -vcdstim counter.vcd counter
VSIM 1> add wave /*
VSIM 2> run 200
```

Example 2 — VHDL adder

First, create the VCD file using vcd dumpports:

```
% cd ~/modeltech/examples
% vlib work
% vcom gates.vhd adder.vhd stimulus.vhd
% vsim testbench
VSIM 1> vcd dumpports -file addern.vcd /testbench/uut/*
VSIM 2> run 1000
VSIM 3> quit -f
```

Next, rerun the adder without the testbench, using the -vcdstim argument:

```
% vsim -vcdstim addern.vcd addern -gn=8 -do "add wave /*; run 1000"
```
Example 3 — Mixed-HDL design

First, create three VCD files, one for each module:

```bash
% cd ~/modeltech/examples/mixedHDL
% vlib work
% vlog cache.v memory.v proc.v
% vcom util.vhd set.vhd top.vhd
% vsim top
  VSIM 1> vcd dumpports -file proc.vcd /top/p/*
  VSIM 2> vcd dumpports -file cache.vcd /top/c/*
  VSIM 3> vcd dumpports -file memory.vcd /top/m/*
  VSIM 4> run 1000
  VSIM 5> quit -f
```

Next, rerun each module separately, using the captured VCD stimulus:

```bash
% vsim -vcdstim proc.vcd proc -do "add wave /*; run 1000"
  VSIM 1> quit -f

% vsim -vcdstim cache.vcd cache -do "add wave /*; run 1000"
  VSIM 1> quit -f

% vsim -vcdstim memory.vcd memory -do "add wave /*; run 1000"
  VSIM 1> quit -f
```
A VCD file from source to output

The following example shows the VHDL source, a set of simulator commands, and the resulting VCD output.

VHDL source code

The design is a simple shifter device represented by the following VHDL source code:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity SHIFTER_MOD is
  port (CLK, RESET, data_in : IN STD_LOGIC;
        Q : INOUT STD_LOGIC_VECTOR(8 downto 0));
END SHIFTER_MOD;

architecture RTL of SHIFTER_MOD is
begin
  process (CLK,RESET)
  begin
    if (RESET = '1') then
      Q <= (others => '0');
    elsif (CLK'event and CLK = '1') then
      Q <= Q(Q'left - 1 downto 0) & data_in;
    end if;
  end process;
end;
```

VCD simulator commands

At simulator time zero, the designer executes the following commands and quits the simulator at time 1200:

```bash
vcd files output.vcd
vcd add -r *
force reset 1 0
force data_in 0 0
force clk 0 0
run 100
force clk 1 0, 0 50 -repeat 100
run 100
vcd off
force reset 0 0
force data_in 1 0
run 100
vcd on
run 850
force reset 1 0
run 50
vcd checkpoint
```
VCD output

The VCD file created as a result of the preceding scenario would be called `output.vcd`. The following pages show how it would look.

### VCD output

<table>
<thead>
<tr>
<th><code>$comment</code></th>
<th>0'</th>
</tr>
</thead>
<tbody>
<tr>
<td>File created using the following command:</td>
<td>0(</td>
</tr>
<tr>
<td><code>vcd files output.vcd</code></td>
<td>0*</td>
</tr>
<tr>
<td><code>$date</code></td>
<td>0+</td>
</tr>
<tr>
<td>Fri Jan 12 09:07:17 2000</td>
<td>0,</td>
</tr>
<tr>
<td><code>$end</code></td>
<td>$end</td>
</tr>
<tr>
<td><code>$version</code></td>
<td>$#100</td>
</tr>
<tr>
<td>ModelSim EE/PLUS 5.4</td>
<td>$150</td>
</tr>
<tr>
<td><code>$end</code></td>
<td>0!</td>
</tr>
<tr>
<td><code>$timescale</code></td>
<td>$200</td>
</tr>
<tr>
<td>1ns</td>
<td>1!</td>
</tr>
<tr>
<td><code>$end</code></td>
<td>$dumpoff</td>
</tr>
<tr>
<td><code>$scope module shifter_mod $end</code></td>
<td>$dumpoff</td>
</tr>
<tr>
<td><code>$var wire 1 ! clk $end</code></td>
<td>x!</td>
</tr>
<tr>
<td><code>$var wire 1 &quot; reset $end</code></td>
<td>x&quot;</td>
</tr>
<tr>
<td><code>$var wire 1 # data_in $end</code></td>
<td>x#</td>
</tr>
<tr>
<td><code>$var wire 1 $ q [8] $end</code></td>
<td>x$</td>
</tr>
<tr>
<td><code>$var wire 1 &amp; q [7] $end</code></td>
<td>x&amp;</td>
</tr>
<tr>
<td><code>$var wire 1 ^ q [6] $end</code></td>
<td>x'</td>
</tr>
<tr>
<td><code>$var wire 1 \ q [5] $end</code></td>
<td>x(</td>
</tr>
<tr>
<td><code>$var wire 1 ! q [4] $end</code></td>
<td>x(</td>
</tr>
<tr>
<td><code>$var wire 1 ) q [3] $end</code></td>
<td>x)</td>
</tr>
<tr>
<td><code>$var wire 1 * q [2] $end</code></td>
<td>x*</td>
</tr>
<tr>
<td><code>$var wire 1 + q [1] $end</code></td>
<td>x*</td>
</tr>
<tr>
<td><code>$var wire 1 , q [0] $end</code></td>
<td>x,</td>
</tr>
<tr>
<td><code>$upscope $end</code></td>
<td>$end</td>
</tr>
<tr>
<td><code>$enddefinitions $end</code></td>
<td>$#300</td>
</tr>
<tr>
<td><code>$dumpvars</code></td>
<td>$dumpoff</td>
</tr>
<tr>
<td>1!</td>
<td>x!</td>
</tr>
<tr>
<td>0&quot;</td>
<td>x&quot;</td>
</tr>
<tr>
<td>0#</td>
<td>x#</td>
</tr>
<tr>
<td>0$</td>
<td>x$</td>
</tr>
<tr>
<td>0%</td>
<td>x%</td>
</tr>
<tr>
<td>0$</td>
<td>x$</td>
</tr>
<tr>
<td>0%</td>
<td>#1000</td>
</tr>
<tr>
<td>0'</td>
<td>1!</td>
</tr>
<tr>
<td>0(</td>
<td>1%</td>
</tr>
<tr>
<td>0)</td>
<td>#1050</td>
</tr>
<tr>
<td>0*</td>
<td>0!</td>
</tr>
<tr>
<td>0+</td>
<td>#1100</td>
</tr>
<tr>
<td>1,</td>
<td>1!</td>
</tr>
<tr>
<td>$end</td>
<td>1$</td>
</tr>
<tr>
<td>#350</td>
<td>#1150</td>
</tr>
<tr>
<td>0!</td>
<td>0!</td>
</tr>
</tbody>
</table>
| #400 | 1
| 1!  | 0$    |
| 1+  | 0%    |
| #450 | 0&    |
| 0!  | 0'    |
| #500 | 0(    |
| 1!  | 0)    |
| 1*  | 0*    |
| #550 | 0+    |
| 0!  | 0,    |
| #600 | #1200 |
| 1!  | 1!    |
| 1)  | $dumpall |
| #650 | 1!    |
| 0!  | 1"    |
| #700 | 1#    |
| 1!  | 0$    |
| 1(  | 0%    |
| #750 | 0&    |
| 0!  | 0'    |
| #800 | 0(    |
| 1!  | 0)    |
| 1'  | 0*    |
| #850 | 0+    |
| 0!  | 0,    |
| #900 | $end  |
| 1!  | 1%    |
| #950 | 0!    |
Capturing port driver data

Some ASIC vendor’s toolkits read a VCD file format that provides details on port drivers. This information can be used, for example, to drive a tester. See the ASIC vendor’s documentation for toolkit specific information.

In ModelSim use the vcd dumpports command (CR-224) to create a VCD file that captures port driver data.

Port driver direction information is captured as TSSI states in the VCD file. Each time an external or internal port driver changes values, a new value change is recorded in the VCD file with the following format:

\[ p<TSSI\ \text{state}> \ <0\ \text{strength}> <1\ \text{strength}> <\text{identifier\_code}> \]

Supported TSSI states

The supported <TSSI states> are:

<table>
<thead>
<tr>
<th>Input (testfixture)</th>
<th>Output (dut)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D  low</td>
<td>L  low</td>
</tr>
<tr>
<td>U  high</td>
<td>H  high</td>
</tr>
<tr>
<td>N  unknown</td>
<td>X  unknown</td>
</tr>
<tr>
<td>Z  tri-state</td>
<td>T  tri-state</td>
</tr>
</tbody>
</table>

**Unknown direction**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>low (both input and output are driving low)</td>
</tr>
<tr>
<td>1</td>
<td>high (both input and output are driving high)</td>
</tr>
<tr>
<td>?</td>
<td>unknown (both input and output are driving unknown)</td>
</tr>
<tr>
<td>f</td>
<td>tri-state</td>
</tr>
<tr>
<td>A</td>
<td>unknown (input driving low and output driving high)</td>
</tr>
<tr>
<td>a</td>
<td>unknown (input driving low and output driving unknown)</td>
</tr>
<tr>
<td>C</td>
<td>unknown (input driving unknown and output driving low)</td>
</tr>
<tr>
<td>b</td>
<td>unknown (input driving high and output driving unknown)</td>
</tr>
<tr>
<td>B</td>
<td>unknown (input driving high and output driving low)</td>
</tr>
<tr>
<td>c</td>
<td>unknown (input driving unknown and output driving high)</td>
</tr>
</tbody>
</table>
Strength values

The `<strength>` values are based on Verilog strengths:

<table>
<thead>
<tr>
<th>Strength</th>
<th>VHDL std_logic mappings</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>'Z'</td>
</tr>
<tr>
<td>1</td>
<td>small</td>
</tr>
<tr>
<td>2</td>
<td>medium</td>
</tr>
<tr>
<td>3</td>
<td>weak</td>
</tr>
<tr>
<td>4</td>
<td>large</td>
</tr>
<tr>
<td>5</td>
<td>pull</td>
</tr>
<tr>
<td>6</td>
<td>strong</td>
</tr>
<tr>
<td>7</td>
<td>supply</td>
</tr>
</tbody>
</table>

Port identifier code

The `<identifier_code>` is an integer preceded by `<` that starts at zero and is incremented for each port in the order the ports are specified. Also, the variable type recorded in the VCD header is "port".
Example VCD output from `vcd dumpports`

The following is an example VCD file created with the `vcd dumpports` command.

```
$comment
File created using the following command:
vcd dumpports results/dump1
$end
$date
Tue Aug 20 13:33:02 2000
$end
$version
ModelSim Version 5.4c
$end
$timescale
1ns
$end
$scope module top1 $end
$scope module u1 $end
$var port 1 <0 a $end
$var port 1 <1 b $end
$var port 1 <2 c $end
$upscope $end
$upscope $end
$enddefinitions $end
#0
$vcd dumpports
pN 6 6 <0
pX 6 6 <1
p? 6 6 <2
$end
#10
pX 6 6 <1
pN 6 6 <0
p? 6 6 <2
```

#20
pL 6 0 <1
pD 6 0 <0
pa 6 6 <2

#30
pH 0 6 <1
pU 0 6 <0
pb 6 6 <2

#40
pT 0 0 <1
pZ 0 0 <0
px 6 6 <2

#50
px 5 5 <1
pN 5 5 <0
p? 6 6 <2

#60
pL 5 0 <1
pD 5 0 <0
pa 6 6 <2

#70
pH 0 5 <1
pU 0 5 <0
pb 6 6 <2

#80
px 6 6 <1
pN 6 6 <0
p? 6 6 <2
14 - Logic Modeling SmartModels

Chapter contents

VHDL SmartModel interface . . . . . . . . . . . . . . . . . . . . . . . . UM-346
Creating foreign architectures with sm_entity . . . . . . . . . . . . UM-347
Vector ports . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-349
Command channel . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-350
SmartModel Windows . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-351
Memory arrays . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-352

Verilog SmartModel interface . . . . . . . . . . . . . . . . . . . . . . . . UM-353
Linking the LMTV interface to the simulator . . . . . . . . . . . . . . . . . UM-353

The Logic Modeling SWIFT-based SmartModel library can be used with ModelSim VHDL and Verilog. The SmartModel library is a collection of behavioral models supplied in binary form with a procedural interface that is accessed by the simulator. This chapter describes how to use the SmartModel library with ModelSim.

Note: The SmartModel library must be obtained from Logic Modeling along with the SmartModel library documentation that describes how to use it. This chapter only describes the specifics of using the library with ModelSim SE.
VHDL SmartModel interface

ModelSim VHDL interfaces to a SmartModel through a foreign architecture. The foreign architecture contains a foreign attribute string that associates a specific SmartModel with the architecture. On elaboration of the foreign architecture, the simulator automatically loads the SmartModel library software and establishes communication with the specific SmartModel.

The ModelSim software locates the SmartModel interface software based on entries in the modelsim.ini initialization file. The simulator and the sm_entity tool (for creating foreign architectures) both depend on these entries being set correctly. These entries are found under the [lmc] section of the default modelsim.ini file located in the ModelSim installation directory. The default settings are as follows:

```
[lmc]

; ModelSim's interface to Logic Modeling's SmartModel SWIFT software
libsm = $MODEL_TECH/libsm.sl
; ModelSim's interface to Logic Modeling's SmartModel SWIFT software (Windows NT)
; libsm = $MODEL_TECH/libsm.dll
; Logic Modeling's SmartModel SWIFT software (HP 9000 Series 700)
; libswift = $LMC_HOME/lib/hp700.lib/libswift.sl
; Logic Modeling's SmartModel SWIFT software (IBM RISC System/6000)
; libswift = $LMC_HOME/lib/ibmrs.lib/swift.o
; Logic Modeling's SmartModel SWIFT software (Sun4 Solaris)
; libswift = $LMC_HOME/lib/sun4Solaris.lib/libswift.so
; Logic Modeling's SmartModel SWIFT software (Windows NT)
; libswift = $LMC_HOME/lib/pcnt.lib/libswift.dll
; Logic Modeling's SmartModel SWIFT software (Linux)
; libswift = $LMC_HOME/lib/x86_linux.lib/libswift.so
```

The libsm entry points to the ModelSim dynamic link library that interfaces the foreign architecture to the SmartModel software. The libswift entry points to the Logic Modeling dynamic link library software that accesses the SmartModels. The simulator automatically loads both the libsm and libswift libraries when it elaborates a SmartModel foreign architecture.

By default, the libsm entry points to the libsm.sl supplied in the ModelSim installation directory indicated by the MODEL_TECH environment variable. ModelSim automatically sets the MODEL_TECH environment variable to the appropriate directory containing the executables and binaries for the current operating system. If you are running the Windows operating system, then you must comment out the default libsm entry (precede the line with the ";" character) and uncomment the libsm entry for the Windows operating system.

Uncomment the appropriate libswift entry for your operating system. The LMC_HOME environment variable must be set to the root of the SmartModel library installation directory. Consult Logic Modeling’s SmartModel library documentation for details.
Creating foreign architectures with sm_entity

The ModelSim sm_entity tool automatically creates entities and foreign architectures for SmartModels. Its usage is as follows:

Syntax

```
sm_entity
   [ - ] [ -xe ] [ -xa ] [ -c ] [ -all ] [ -v ] [ -93 ] [ <SmartModelName>... ]
```

Arguments

- Read SmartModel names from standard input.

-xe
  Do not generate entity declarations.

-xa
  Do not generate architecture bodies.

-c
  Generate component declarations.

-all
  Select all models installed in the SmartModel library.

-v
  Display progress messages.

-93
  Use extended identifiers where needed.

<SmartModelName>
  Name of a SmartModel (see the SmartModel library documentation for details on SmartModel names).

By default, the sm_entity tool writes an entity and foreign architecture to stdout for each SmartModel name listed on the command line. Optionally, you can include the component declaration (-c), exclude the entity (-xe), and exclude the architecture (-xa).

The simplest way to prepare SmartModels for use with ModelSim VHDL is to generate the entities and foreign architectures for all installed SmartModels, and compile them into a library named lmc. This is easily accomplished with the following commands:

```
% sm_entity -all > sml.vhd
% vlib lmc
% vcom -work lmc sml.vhd
```

To instantiate the SmartModels in your VHDL design, you also need to generate component declarations for the SmartModels. Add these component declarations to a package named sml (for example), and compile the package into the lmc library:

```
% sm_entity -all -c -xe -xa > smlcomp.vhd
```

Edit the resulting smlcomp.vhd file to turn it into a package of SmartModel component declarations as follows:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
```
package sml is
    <component declarations go here>
end sml;

Compile the package into the lmc library:

    % vcom -work lmc smlcomp.vhd

The SmartModels can now be referenced in your design by adding the following library and use clauses to your code:

    library lmc;
    use lmc.sml.all;

The following is an example of an entity and foreign architecture created by sm_entity for the cy7c285 SmartModel.

    library ieee;
    use ieee.std_logic_1164.all;

    entity cy7c285 is
        generic (TimingVersion : STRING := "CY7C285-65";
            DelayRange : STRING := "Max";
            MemoryFile : STRING := "memory" );
        port ( A0 : in std_logic;
            A1 : in std_logic;
            A2 : in std_logic;
            A3 : in std_logic;
            A4 : in std_logic;
            A5 : in std_logic;
            A6 : in std_logic;
            A7 : in std_logic;
            A8 : in std_logic;
            A9 : in std_logic;
            A10 : in std_logic;
            A11 : in std_logic;
            A12 : in std_logic;
            A13 : in std_logic;
            A14 : in std_logic;
            A15 : in std_logic;
            CS : in std_logic;
            O0 : out std_logic;
            O1 : out std_logic;
            O2 : out std_logic;
            O3 : out std_logic;
            O4 : out std_logic;
            O5 : out std_logic;
            O6 : out std_logic;
            O7 : out std_logic;
            WAIT_PORT : inout std_logic );
    end;

    architecture SmartModel of cy7c285 is
        attribute FOREIGN : STRING;
        attribute FOREIGN of SmartModel : architecture is
            "sm_init SMODEL_TECH/libsm.sl ; cy7c285";
    begin
    end SmartModel;
**Entity details**

- The entity name is the SmartModel name (you can manually change this name if you like).

- The port names are the same as the SmartModel port names (*these names must not be changed*). If the SmartModel port name is not a valid VHDL identifier, then `sm_entity` automatically converts it to a valid name. If `sm_entity` is invoked with the `-93` option, then the identifier is converted to an extended identifier, and the resulting entity must also be compiled with the `-93` option. If the `-93` option had been specified in the example above, then `WAIT` would have been converted to `\WAIT`. Note that in this example the port `WAIT` was converted to `WAIT_PORT` because `wait` is a VHDL reserved word.

- The port types are `std_logic`. This data type supports the full range of SmartModel logic states.

- The `DelayRange`, `TimingVersion`, and `MemoryFile` generics represent the SmartModel attributes of the same name. Consult your SmartModel library documentation for a description of these attributes (and others). `Sm_entity` creates a generic for each attribute of the particular SmartModel. The default generic value is the default attribute value that the SmartModel has supplied to `sm_entity`.

**Architecture details**

- The first part of the foreign attribute string (`sm_init`) is the same for all SmartModels.

- The second part (`$MODEL_TECH/libsm.sl`) is taken from the `libsm` entry in the initialization file, `modelsim.ini`.

- The third part (`cy7c285`) is the SmartModel name. This name correlates the architecture with the SmartModel at elaboration.

**Vector ports**

The entities generated by `sm_entity` only contain single-bit ports, never vectored ports. This is necessary because ModelSim correlates entity ports with the SmartModel SWIFT interface by name. However, for ease of use in component instantiations, you may want to create a custom component declaration and component specification that groups ports into vectors. You can also rename and reorder the ports in the component declaration. You can also reorder the ports in the entity declaration, but you can't rename them!

The following is an example component declaration and specification that groups the address and data ports of the CY7C285 SmartModel:

```vhdl
component cy7c285
  generic ( TimingVersion : STRING := "CY7C285-65";
             DelayRange : STRING := "Max";
             MemoryFile : STRING := "memory" );
  port ( A : in std_logic_vector (15 downto 0);
         CS : in std_logic;
         O : out std_logic_vector (7 downto 0);
         WAIT_PORT : inout std_logic );
end component;

for all: cy7c285
  use entity work.cy7c285
  port map (A0 => A(0),
            A1 => A(1),
```
Command channel

The command channel is a SmartModel feature that lets you invoke SmartModel specific commands. These commands are documented in the SmartModel library documentation. ModelSim provides access to the Command Channel from the command line. The form of a SmartModel command is:

```
  lmc <instance_name>|-all "<SmartModel command>"
```

The instance_name argument is either a full hierarchical name or a relative name of a SmartModel instance. A relative name is relative to the current environment setting (see environment command (CR-137)). For example, to turn timing checks off for SmartModel /top/u1:

```
  lmc /top/u1 "SetConstraints Off"
```

Use -all to apply the command to all SmartModel instances. For example, to turn timing checks off for all SmartModel instances:

```
  lmc -all "SetConstraints Off"
```

There are also some SmartModel commands that apply globally to the current simulation session rather than to models. The form of a SmartModel session command is:

```
  lmcsession "<SmartModel session command>"
```

Once again, consult your SmartModel library documentation for details on these commands.
SmartModel Windows

Some models in the SmartModel library provide access to internal registers with a feature called SmartModel Windows. Refer to Logic Modeling’s SmartModel library documentation (available on Synopsys’ web site) for details on this feature. The simulator interface to this feature is described below.

Window name syntax is important. Beginning in version 5.3c of ModelSim, window names that are not valid VHDL or Verilog identifiers are converted to VHDL extended identifiers. For example, with a window named z1110.GSR.OR, ModelSim will treat the name as \z1110.GSR.OR\ (for all commands including lmcwin, add wave, and examine). You must then use that name in all commands. For example,

```bash
add wave /top/swift_model/\z1110.GSR.OR\n```

As with all extended identifiers, case is important.

ReportStatus

The ReportStatus command displays model information, including the names of window registers. For example,

```bash
lmc /top/u1 ReportStatus
```

SmartModel Windows description:

- WA "Read-Only (Read Only)"
- WB "1-bit"
- WC "64-bit"

This model contains window registers named wa, wb, and wc. These names can be used in subsequent window (lmcwin) commands.

SmartModel lmcwin commands

The following window commands are supported:

- `lmcwin read <window_instance> [-<radix>]`
- `lmcwin write <window_instance> <value>`
- `lmcwin enable <window_instance>`
- `lmcwin disable <window_instance>`
- `lmcwin release <window_instance>`

Each command requires a window instance argument that identifies a specific model instance and window name. For example, /top/u1/wa refers to window wa in model instance /top/u1.

Imcwin read

The lmcwin read command displays the current value of a window. The optional radix argument is `binary`, `decimal`, or `hexadecimal` (these names can be abbreviated). The default is to display the value using the `std_logic` characters. For example, the following command displays the 64-bit window wc in hexadecimal:

```bash
lmcwin read /top/u1/wc -h
```
**lmcwin write**

The `lmcwin write` command writes a value into a window. The format of the value argument is the same as used in other simulator commands that take value arguments. For example, to write 1 to window `wb`, and all 1’s to window `wc`:

```markdown
lmcwin write /top/u1/wb 1
lmcwin write /top/u1/wc X"FFFFFFFFFFFFFFFFFF"
```

**lmcwin enable**

The `lmcwin enable` command enables continuous monitoring of a window. The specified window is added to the model instance as a signal (with the same name as the window) of type `std_logic` or `std_logic_vector`. This signal can then be referenced in other simulator commands just like any other signal (the `add list` command (CR-49) is shown below). The window signal is continuously updated to reflect the value in the model. For example, to list window `wa`:

```markdown
lmcwin enable /top/u1/wa
add list /top/u1/wa
```

**lmcwin disable**

The `lmcwin disable` command disables continuous monitoring of a window. The window signal is not deleted, but it no longer is updated when the model’s window register changes value. For example, to disable continuous monitoring of window `wa`:

```markdown
lmcwin disable /top/u1/wa
```

**lmcwin release**

Some windows are actually nets, and the `lmcwin write` command behaves more like a continuous force on the net. The `lmcwin release` command disables the effect of a previous `lmcwin write` command on a window net.

**Memory arrays**

A memory model usually makes the entire register array available as a window. In this case, the window commands operate only on a single element at a time. The element is selected as an array reference in the window instance specification. For example, to read element 5 from the window memory `mem`:

```markdown
lmcwin read /top/u2/mem(5)
```

Omitting the element specification defaults to element 0. Also, continuous monitoring is limited to a single array element. The associated window signal is updated with the most recently enabled element for continuous monitoring.
Verilog SmartModel interface

The SWIFT SmartModel library, beginning with release r40b, provides an optional library of Verilog modules and a PLI application that communicates between a simulator’s PLI and the SWIFT simulator interface. The Logic Modeling documentation refers to this as the Logic Models to Verilog (LMTV) interface. To install this option, you must select the simulator type "Verilog" when you run Logic Modeling’s SmartInstall program.

Linking the LMTV interface to the simulator

Synopsys provides a dynamically loadable library that links ModelSim to the LMTV interface. See chapter 5, “Using MTI Verilog with Synopsys Models,” in the “Simulator Configuration Guide for Synopsys Models” (available on Synopsys’ web site) for directions on how to link to this library.
Logic Modeling hardware models can be used with ModelSim VHDL and Verilog. A hardware model allows simulation of a device using the actual silicon installed as a hardware model in one of Logic Modeling's hardware modeling systems. The hardware modeling system is a network resource with a procedural interface that is accessed by the simulator. This chapter describes how to use Logic Modeling hardware models with ModelSim.

**Note:** Please refer to the Logic Modeling documentation for details on using the hardware modeler. This chapter only describes the specifics of using hardware models with ModelSim SE.
VHDL Hardware Model interface

ModelSim VHDL interfaces to a hardware model through a foreign architecture. The foreign architecture contains a foreign attribute string that associates a specific hardware model with the architecture. On elaboration of the foreign architecture, the simulator automatically loads the hardware modeler software and establishes communication with the specific hardware model.

The ModelSim software locates the hardware modeler interface software based on entries in the modelsim.ini initialization file. The simulator and the hm_entity tool (for creating foreign architectures) both depend on these entries being set correctly. These entries are found under the [lmc] section of the default modelsim.ini file located in the ModelSim installation directory. The default settings are as follows:

```
[lmc]
; ModelSim's interface to Logic Modeling's hardware modeler SFI software
libhm = $MODEL_TECH/libhm.sl
; ModelSim's interface to Logic Modeling's hardware modeler SFI software
(libhm) ; (Windows NT)
; libhm = $MODEL_TECH/libhm.dll
; Logic Modeling's hardware modeler SFI software (HP 9000 Series 700)
; libsfi = <$sfi_dir>/lib/hp700/libsfi.sl
; Logic Modeling's hardware modeler SFI software (IBM RISC System/6000)
; libsfi = <$sfi_dir>/lib/rs6000/libsfi.a
; Logic Modeling's hardware modeler SFI software (Sun4 Solaris)
; libsfi = <$sfi_dir>/lib/sun4.solaris/libsfi.so
; Logic Modeling's hardware modeler SFI software (Window NT)
; libsfi = <$sfi_dir>/lib/pcnt/lm_sfi.dll
```

The `libhm` entry points to the ModelSim dynamic link library that interfaces the foreign architecture to the hardware modeler software. The `libsfi` entry points to the Logic Modeling dynamic link library software that accesses the hardware modeler. The simulator automatically loads both the `libhm` and `libsfi` libraries when it elaborates a hardware model foreign architecture.

By default, the `libhm` entry points to the `libhm.sl` supplied in the ModelSim installation directory indicated by the MODEL_TECH environment variable. ModelSim automatically sets the MODEL_TECH environment variable to the appropriate directory containing the executables and binaries for the current operating system. If you are running the Windows operating system, then you must comment out the default `libhm` entry (precede the line with the ";" character) and uncomment the `libhm` entry for the Windows operating system.

Uncomment the appropriate `libsfi` entry for your operating system, and replace `<sfi_dir>` with the path to the hardware modeler software installation directory. In addition, you must set the LM_LIB and LM_DIR environment variables as described in the Logic Modeling documentation.
Creating foreign architectures with hm_entity

The ModelSim hm_entity tool automatically creates entities and foreign architectures for hardware models. Its usage is as follows:

Syntax

hm_entity

Arguments

-xe
   Do not generate entity declarations.

-xa
   Do not generate architecture bodies.

-c
   Generate component declarations.

-93
   Use extended identifiers where needed.

<shell software filename>
   Hardware model shell software filename (see Logic Modeling documentation for details on shell software files)

By default, the hm_entity tool writes an entity and foreign architecture to stdout for the hardware model. Optionally, you can include the component declaration (-c), exclude the entity (-xe), and exclude the architecture (-xa).

Once you have created the entity and foreign architecture, you must compile it into a library. For example, the following commands compile the entity and foreign architecture for a hardware model named LMTEST:

% hm_entity LMTEST.MDL > lmtest.vhd
% vlib lmc
% vcom -work lmc lmtest.vhd

To instantiate the hardware model in your VHDL design, you will also need to generate a component declaration. If you have multiple hardware models, you may want to add all of their component declarations to a package so that you can easily reference them in your design. The following command writes the component declaration to stdout for the LMTEST hardware model.

% hm_entity -c -xe -xa LMTEST.MDL

Paste the resulting component declaration into the appropriate place in your design or into a package.

The following is an example of the entity and foreign architecture created by hm_entity for the CY7C285 hardware model:

library ieee;
use ieee.std_logic_1164.all;

entity cy7c285 is
   generic ( DelayRange : STRING := "Max" );
   port ( A0 : in std_logic;

...
A1 : in std_logic;
A2 : in std_logic;
A3 : in std_logic;
A4 : in std_logic;
A5 : in std_logic;
A6 : in std_logic;
A7 : in std_logic;
A8 : in std_logic;
A9 : in std_logic;
A10 : in std_logic;
A11 : in std_logic;
A12 : in std_logic;
A13 : in std_logic;
A14 : in std_logic;
A15 : in std_logic;
CS : in std_logic;
O0 : out std_logic;
O1 : out std_logic;
O2 : out std_logic;
O3 : out std_logic;
O4 : out std_logic;
O5 : out std_logic;
O6 : out std_logic;
O7 : out std_logic;
W : inout std_logic );
end;

architecture Hardware of cy7c285 is
attribute FOREIGN : STRING;
attribute FOREIGN of Hardware : architecture is
  "hm_init $MODEL_TECH/libhm.sl ; CY7C285.MDL";
begin
end Hardware;

Entity details

- The entity name is the hardware model name (you can manually change this name if you like).
- The port names are the same as the hardware model port names (these names must not be changed). If the hardware model port name is not a valid VHDL identifier, then hm_entity issues an error message. If hm_entity is invoked with the -93 option, then the identifier is converted to an extended identifier, and the resulting entity must also be compiled with the -93 option. Another option is to create a pin-name mapping file. Consult the Logic Modeling documentation for details.
- The port types are std_logic. This data type supports the full range of hardware model logic states.
- The DelayRange generic selects minimum, typical, or maximum delay values. Valid values are "min", "typ", or "max" (the strings are not case-sensitive). The default is "max".
Architecture details

- The first part of the foreign attribute string (hm_init) is the same for all hardware models.
- The second part ($MODEL_TECH/libhm.sl) is taken from the libhm entry in the initialization file, modelsim.ini.
- The third part (CY7C285.MDL) is the shell software filename. This name correlates the architecture with the hardware model at elaboration.

Vector ports

The entities generated by hm_entity only contain single-bit ports, never vectored ports. However, for ease of use in component instantiations, you may want to create a custom component declaration and component specification that groups ports into vectors. You can also rename and reorder the ports in the component declaration. You can also reorder the ports in the entity declaration, but you can’t rename them!

The following is an example component declaration and specification that groups the address and data ports of the CY7C285 hardware model:

```vhdl
component cy7c285
    generic ( DelayRange : STRING := "Max");
    port ( A : in std_logic_vector (15 downto 0);
         CS : in std_logic;
         O : out std_logic_vector (7 downto 0);
         WAIT_PORT : inout std_logic );
end component;

for all: cy7c285
    use entity work.cy7c285
    port map (A0 => A(0),
                A1 => A(1),
                A2 => A(2),
                A3 => A(3),
                A4 => A(4),
                A5 => A(5),
                A6 => A(6),
                A7 => A(7),
                A8 => A(8),
                A9 => A(9),
                A10 => A(10),
                A11 => A(11),
                A12 => A(12),
                A13 => A(13),
                A14 => A(14),
                A15 => A(15),
                CS => CS,
                O0 => O(0),
                O1 => O(1),
                O2 => O(2),
                O3 => O(3),
                O4 => O(4),
                O5 => O(5),
                O6 => O(6),
                O7 => O(7),
                WAIT_PORT => W );
```

ModelSim SE User's Manual
Hardware model commands

The following simulator commands are available for hardware models. Refer to the Logic Modeling documentation for details on these operations.

`lm_vectors on|off <instance_name> [<filename>]`
Enable/disable test vector logging for the specified hardware model.

`lm_measure_timing on|off <instance_name> [<filename>]`
Enable/disable timing measurement for the specified hardware model.

`lm_timing_checks on|off <instance_name>`
Enable/disable timing checks for the specified hardware model.

`lm_loop_patterns on|off <instance_name>`
Enable/disable pattern looping for the specified hardware model.

`lm_unknowns on|off <instance_name>`
Enable/disable unknown propagation for the specified hardware model.
This chapter provides an overview of Tcl (tool command language) as used with ModelSim. Macros in ModelSim are simply Tcl scripts that contain ModelSim and, optionally, Tcl commands.

Tcl is a scripting language for controlling and extending ModelSim. Within ModelSim you can develop implementations from Tcl scripts without the use of C code. Because Tcl is interpreted, development is rapid; you can generate and execute Tcl scripts on the fly without stopping to recompile or restart ModelSim. In addition, if ModelSim does not provide the command you need, you can use Tcl to create your own commands.
Tcl features within ModelSim

Using Tcl with ModelSim gives you these features:

- command history (like that in C shells)
- full expression evaluation and support for all C-language operators
- a full range of math and trig functions
- support of lists and arrays
- regular expression pattern matching
- procedures
- the ability to define your own commands
- command substitution (that is, commands may be nested)
- robust scripting language for macros

Tcl References

Two books about Tcl are *Tcl and the Tk Toolkit* by John K. Ousterhout, published by Addison-Wesley Publishing Company, Inc., and *Practical Programming in Tcl and Tk* by Brent Welch published by Prentice Hall. You can also consult the following online references:

- Select **Help > Tcl Man Pages** (Main window).
- The Model Technology web site lists a variety of Tcl resources:
  
  www.model.com/resources/tcltk.asp

Tcl tutorial

For some hands-on experience using Tcl with ModelSim, see the "Tcl/Tk and ModelSim" lesson in the *ModelSim SE Tutorial*. 
Tcl commands

The Tcl commands are listed below. For complete information on Tcl commands, select Help > Tcl Man Pages (Main window). Also see "Preference variables located in TCL files" (UM-396) for information on Tcl variables.

<table>
<thead>
<tr>
<th>append</th>
<th>array</th>
<th>break</th>
<th>case</th>
<th>continue</th>
<th>catch</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd</td>
<td>close</td>
<td>concat</td>
<td>continue</td>
<td>eof</td>
<td></td>
</tr>
<tr>
<td>error</td>
<td>eval</td>
<td>exec</td>
<td>expr</td>
<td>file</td>
<td></td>
</tr>
<tr>
<td>flush</td>
<td>for</td>
<td>foreach</td>
<td>format</td>
<td>gets</td>
<td></td>
</tr>
<tr>
<td>glob</td>
<td>global</td>
<td>history</td>
<td>if</td>
<td>incr</td>
<td></td>
</tr>
<tr>
<td>info</td>
<td>insert</td>
<td>join</td>
<td>lappend</td>
<td>list</td>
<td></td>
</tr>
<tr>
<td>llength</td>
<td>lindex</td>
<td>lrange</td>
<td>lreplace</td>
<td>lsearch</td>
<td></td>
</tr>
<tr>
<td>lsort</td>
<td>open</td>
<td>pid</td>
<td>proc</td>
<td>puts</td>
<td></td>
</tr>
<tr>
<td>pwd</td>
<td>read</td>
<td>regexp</td>
<td>regsub</td>
<td>rename</td>
<td></td>
</tr>
<tr>
<td>return</td>
<td>scan</td>
<td>seek</td>
<td>set</td>
<td>split</td>
<td></td>
</tr>
<tr>
<td>string</td>
<td>switch</td>
<td>tell</td>
<td>time</td>
<td>trace</td>
<td></td>
</tr>
<tr>
<td>source</td>
<td>unset</td>
<td>uplevel</td>
<td>upvar</td>
<td>while</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** ModelSim command names that conflict with Tcl commands have been renamed or have been replaced by Tcl commands. See the list below:

<table>
<thead>
<tr>
<th>Previous ModelSim command</th>
<th>Command changed to (or replaced by)</th>
</tr>
</thead>
<tbody>
<tr>
<td>continue</td>
<td><strong>run</strong> (CR-199) with the <strong>-continue</strong> option</td>
</tr>
<tr>
<td>format list</td>
<td>wave</td>
</tr>
<tr>
<td>if</td>
<td>replaced by the Tcl <strong>if</strong> command, see &quot;if command syntax&quot; (UM-366) for more information</td>
</tr>
<tr>
<td>list</td>
<td><strong>add list</strong> (CR-49)</td>
</tr>
<tr>
<td>nolist</td>
<td>nowave</td>
</tr>
<tr>
<td>set</td>
<td>replaced by the Tcl <strong>set</strong> command, see &quot;set command syntax&quot; (UM-367) for more information</td>
</tr>
<tr>
<td>source</td>
<td><strong>vsource</strong> (CR-297)</td>
</tr>
<tr>
<td>wave</td>
<td><strong>add wave</strong> (CR-58)</td>
</tr>
</tbody>
</table>
Tcl command syntax

The following eleven rules define the syntax and semantics of the Tcl language. Additional details on if command syntax (UM-366) and set command syntax (UM-367) follow.

1 A Tcl script is a string containing one or more commands. Semi-colons and newlines are command separators unless quoted as described below. Close brackets ("]") are command terminators during command substitution (see below) unless quoted.

2 A command is evaluated in two steps. First, the Tcl interpreter breaks the command into words and performs substitutions as described below. These substitutions are performed in the same way for all commands. The first word is used to locate a command procedure to carry out the command, then all of the words of the command are passed to the command procedure. The command procedure is free to interpret each of its words in any way it likes, such as an integer, variable name, list, or Tcl script. Different commands interpret their words differently.

3 Words of a command are separated by white space (except for newlines, which are command separators).

4 If the first character of a word is double-quote (""”) then the word is terminated by the next double-quote character. If semi-colons, close brackets, or white space characters (including newlines) appear between the quotes then they are treated as ordinary characters and included in the word. Command substitution, variable substitution, and backslash substitution are performed on the characters between the quotes as described below. The double-quotes are not retained as part of the word.

5 If the first character of a word is an open brace ("{") then the word is terminated by the matching close brace ("}”). Braces nest within the word: for each additional open brace there must be an additional close brace (however, if an open brace or close brace within the word is quoted with a backslash then it is not counted in locating the matching close brace). No substitutions are performed on the characters between the braces except for backslash-newline substitutions described below, nor do semi-colons, newlines, close brackets, or white space receive any special interpretation. The word will consist of exactly the characters between the outer braces, not including the braces themselves.

6 If a word contains an open bracket ("[") then Tcl performs command substitution. To do this it invokes the Tcl interpreter recursively to process the characters following the open bracket as a Tcl script. The script may contain any number of commands and must be terminated by a close bracket ("]"). The result of the script (i.e. the result of its last command) is substituted into the word in place of the brackets and all of the characters between them. There may be any number of command substitutions in a single word. Command substitution is not performed on words enclosed in braces.
7 If a word contains a dollar-sign ("$") then Tcl performs variable substitution: the dollar-sign and the following characters are replaced in the word by the value of a variable. Variable substitution may take any of the following forms:

\$name

Name is the name of a scalar variable; the name is terminated by any character that isn’t a letter, digit, or underscore.

\$name(index)

Name gives the name of an array variable and index gives the name of an element within that array. Name must contain only letters, digits, and underscores. Command substitutions, variable substitutions, and backslash substitutions are performed on the characters of index.

\${name}

Name is the name of a scalar variable. It may contain any characters whatsoever except for close braces.

There may be any number of variable substitutions in a single word. Variable substitution is not performed on words enclosed in braces.

8 If a backslash ("\") appears within a word then backslash substitution occurs. In all cases but those described below the backslash is dropped and the following character is treated as an ordinary character and included in the word. This allows characters such as double quotes, close brackets, and dollar signs to be included in words without triggering special processing. The following table lists the backslash sequences that are handled specially, along with the value that replaces each sequence.

| \a | Audible alert (bell) (0x7). |
| \b | Backspace (0x8). |
| \f | Form feed (0xc). |
| \n | Newline (0xa). |
| \r | Carriage-return (0xd). |
| \t | Tab (0x9). |
| \v | Vertical tab (0xb). |
| \<newline>whiteSpace | A single space character replaces the backslash, newline, and all spaces and tabs after the newline. This backslash sequence is unique in that it is replaced in a separate pre-pass before the command is actually parsed. This means that it will be replaced even when it occurs between braces, and the resulting space will be treated as a word separator if it isn’t in braces or quotes. |
| \\ | Backslash ("\`). |
Backslash substitution is not performed on words enclosed in braces, except for backslash-newline as described above.

9 If a hash character ("#") appears at a point where Tcl is expecting the first character of the first word of a command, then the hash character and the characters that follow it, up through the next newline, are treated as a comment and ignored. The comment character only has significance when it appears at the beginning of a command.

10 Each character is processed exactly once by the Tcl interpreter as part of creating the words of a command. For example, if variable substitution occurs then no further substitutions are performed on the value of the variable; the value is inserted into the word verbatim. If command substitution occurs then the nested command is processed entirely by the recursive call to the Tcl interpreter; no substitutions are performed before making the recursive call and no additional substitutions are performed on the result of the nested script.

11 Substitutions do not affect the word boundaries of a command. For example, during variable substitution the entire value of the variable becomes part of a single word, even if the variable's value contains spaces.

**if command syntax**

The Tcl if command executes scripts conditionally. Note that in the syntax below the "?" indicates an optional argument.

**Syntax**

```tcl
if expr1 ?then? body1 elseif expr2 ?then? body2 elseif ... ?else? ?bodyN?
```

**Description**

The if command evaluates `expr1` as an expression. The value of the expression must be a boolean (a numeric value, where 0 is false and anything else is true, or a string value such as `true` or `yes` for true and `false` or `no` for false); if it is true then `body1` is executed by passing it to the Tcl interpreter. Otherwise `expr2` is evaluated as an expression and if it is true then `body2` is executed, and so on. If none of the expressions evaluates to true then `bodyN` is executed. The `then` and `else` arguments are optional "noise words" to make the command easier to read. There may be any number of `elseif` clauses, including zero. `BodyN` may also be omitted as long as `else` is omitted too. The return value from the command is the result of the body script that was executed, or an empty string if none of the expressions was non-zero and there was no `bodyN`. 

| \ooo | The digits ooo (one, two, or three of them) give the octal value of the character. |
| \xhh | The hexadecimal digits hh give the hexadecimal value of the character. Any number of digits may be present. |
set command syntax

The Tcl `set` command reads and writes variables. Note that in the syntax below the "?" indicates an optional argument.

**Syntax**

```
set varName ?value?
```

**Description**

Returns the value of variable `varName`. If value is specified, then sets the value of `varName` to value, creating a new variable if one doesn’t already exist, and returns its value. If `varName` contains an open parenthesis and ends with a close parenthesis, then it refers to an array element: the characters before the first open parenthesis are the name of the array, and the characters between the parentheses are the index within the array. Otherwise `varName` refers to a scalar variable. Normally, `varName` is unqualified (does not include the names of any containing namespaces), and the variable of that name in the current namespace is read or written. If `varName` includes namespace qualifiers (in the array name if it refers to an array element), the variable in the specified namespace is read or written.

If no procedure is active, then `varName` refers to a namespace variable (global variable if the current namespace is the global namespace). If a procedure is active, then `varName` refers to a parameter or local variable of the procedure unless the global command was invoked to declare `varName` to be global, or unless a Tcl `variable` command was invoked to declare `varName` to be a namespace variable.

**Command substitution**

Placing a command in square brackets `[ ]` will cause that command to be evaluated first and its results returned in place of the command. An example is:

```
set a 25
set b 11
set c 3
echo "the result is [expr ($a + $b)/$c]"
```

will output:

"the result is 12"

This feature allows VHDL variables and signals, and Verilog nets and registers to be accessed using:

```
[examine -<radix> name]
```

The `%name` substitution is no longer supported. Everywhere `%name` could be used, you now can use `[examine -value -<radix> name]` which allows the flexibility of specifying command options. The radix specification is optional.
Command separator

A semicolon character (;) works as a separator for multiple commands on the same line. It is not required at the end of a line in a command sequence.

Multiple-line commands

With Tcl, multiple-line commands can be used within macros and on the command line. The command line prompt will change (as in a C shell) until the multiple-line command is complete.

In the example below, note the way the opening brace '{' is at the end of the if and else lines. This is important because otherwise the Tcl scanner won't know that there is more coming in the command and will try to execute what it has up to that point, which won't be what you intend.

```
if { [exa sig_a] == "0011ZZ"} {
    echo "Signal value matches"
    do macro_1.do
} else {
    echo "Signal value fails"
    do macro_2.do
}
```

Evaluation order

An important thing to remember when using Tcl is that anything put in curly brackets {} is not evaluated immediately. This is important for if-then-else, procedures, loops, and so forth.

Tcl relational expression evaluation

When you are comparing values, the following hints may be useful:

- Tcl stores all values as strings, and will convert certain strings to numeric values when appropriate. If you want a literal to be treated as a numeric value, don't quote it.

  ```tcl
  if {[exa var_1] == 345}...
  ```

  The following will also work:

  ```tcl
  if {[exa var_1] == "345"}...
  ```

- However, if a literal cannot be represented as a number, you must quote it, or Tcl will give you an error. For instance:

  ```tcl
  if {[exa var_2] == 0012}...
  ```

  will give an error.

  ```tcl
  if {[exa var_2] == "0012"}...
  ```

  will work okay.

- Don't quote single characters in single quotes:

  ```tcl
  if {[exa var_3] == 'X'}...
  ```

  will give an error

  ```tcl
  if {[exa var_3] == "X"}...
  ```

  will work okay.
For the equal operator, you must use the C operator "==". For not-equal, you must use the C operator "!=".

Variable substitution

When a $<var_name>$ is encountered, the Tcl parser will look for variables that have been defined either by ModelSim or by you, and substitute the value of the variable.

▶ Note: Tcl is case sensitive for variable names.

To access environment variables, use the construct:

```
$env(<var_name>)
```

```
echo My user name is $env(USER)
```

Environment variables can also be set using the env array:

```
set env(SHELL) /bin/csh
```

See "Simulator state variables" (UM-398) for more information about ModelSim-defined variables.

System commands

To pass commands to the UNIX shell or DOS window, use the Tcl exec command:

```
echo The date is [exec date]
```
List processing

In Tcl a "list" is a set of strings in curly braces separated by spaces. Several Tcl commands are available for creating lists, indexing into lists, appending to lists, getting the length of lists and shifting lists. These commands are:

<table>
<thead>
<tr>
<th>Command syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lappend var_name val1 val2 ...</td>
<td>appends val1, val2, etc. to list var_name</td>
</tr>
<tr>
<td>lindex list_name index</td>
<td>returns the index-th element of list_name; the first element is 0</td>
</tr>
<tr>
<td>linsert list_name index val1 val2 ...</td>
<td>inserts val1, val2, etc. just before the index-th element of list_name</td>
</tr>
<tr>
<td>list val1, val2 ...</td>
<td>returns a Tcl list consisting of val1, val2, etc.</td>
</tr>
<tr>
<td>llength list_name</td>
<td>returns the number of elements in list_name</td>
</tr>
<tr>
<td>lrange list_name first last</td>
<td>returns a sublist of list_name, from index first to index last; first or last may be &quot;end&quot;, which refers to the last element in the list</td>
</tr>
<tr>
<td>lreplace list_name first last val1, val2, ...</td>
<td>replaces elements first through last with val1, val2, etc.</td>
</tr>
</tbody>
</table>

Two other commands, *lsearch* and *lsort*, are also available for list manipulation. See the Tcl man pages (Help > Tcl Man Pages) for more information on these commands.

See also the ModelSim Tcl command: *lecho* (CR-151)

ModelSim Tcl commands

These additional commands enhance the interface between Tcl and ModelSim. Only brief descriptions are provided here; for more information and command syntax see the "ModelSim Commands" (CR-29).

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias (CR-62)</td>
<td>creates a new Tcl procedure that evaluates the specified commands; used to create a user-defined alias</td>
</tr>
<tr>
<td>find (CR-142)</td>
<td>locates incrTcl classes and objects</td>
</tr>
<tr>
<td>lecho (CR-151)</td>
<td>takes one or more Tcl lists as arguments and pretty-prints them to the Main window</td>
</tr>
<tr>
<td>lshift (CR-156)</td>
<td>takes a Tcl list as argument and shifts it in-place one place to the left, eliminating the 0th element</td>
</tr>
<tr>
<td>lsublist (CR-157)</td>
<td>returns a sublist of the specified Tcl list that matches the specified Tcl glob pattern</td>
</tr>
<tr>
<td>printenv (CR-175)</td>
<td>echoes to the Main window the current names and values of all environment variables</td>
</tr>
</tbody>
</table>
ModelSim Tcl time commands

ModelSim Tcl time commands make simulator-time-based values available for use within other Tcl procedures.

Time values may optionally contain a units specifier where the intervening space is also optional. If the space is present, the value must be quoted (e.g. 10ns, “10 ns”). Time values without units are taken to be in the UserTimeScale. Return values are always in the current Time Scale Units. All time values are converted to a 64-bit integer value in the current Time Scale. This means that values smaller than the current Time Scale will be truncated to 0.

Conversions

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intToTime &lt;intHi32&gt; &lt;intLo32&gt;</td>
<td>converts two 32-bit pieces (high and low order) into a 64-bit quantity (Time in ModelSim is a 64-bit integer)</td>
</tr>
<tr>
<td>RealToTime &lt;real&gt;</td>
<td>converts a &lt;real&gt; number to a 64-bit integer in the current Time Scale</td>
</tr>
<tr>
<td>scaleTime &lt;time&gt; &lt;scaleFactor&gt;</td>
<td>returns the value of &lt;time&gt; multiplied by the &lt;scaleFactor&gt; integer</td>
</tr>
</tbody>
</table>

Relations

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eqTime &lt;time&gt; &lt;time&gt;</td>
<td>evaluates for equal</td>
</tr>
<tr>
<td>neqTime &lt;time&gt; &lt;time&gt;</td>
<td>evaluates for not equal</td>
</tr>
<tr>
<td>gtTime &lt;time&gt; &lt;time&gt;</td>
<td>evaluates for greater than</td>
</tr>
<tr>
<td>gteTime &lt;time&gt; &lt;time&gt;</td>
<td>evaluates for greater than or equal</td>
</tr>
<tr>
<td>ltTime &lt;time&gt; &lt;time&gt;</td>
<td>evaluates for less than</td>
</tr>
<tr>
<td>lteTime &lt;time&gt; &lt;time&gt;</td>
<td>evaluates for less than or equal</td>
</tr>
</tbody>
</table>

All relation operations return 1 or 0 for true or false respectively and are suitable return values for TCL conditional expressions. For example,

```
if {{eqTime $Now 1750ns}} {
    ...
}
```
Arithmetic

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addTime &lt;time&gt;  &lt;time&gt;</td>
<td>add time</td>
</tr>
<tr>
<td>divTime &lt;time&gt;  &lt;time&gt;</td>
<td>64-bit integer divide</td>
</tr>
<tr>
<td>mulTime &lt;time&gt;  &lt;time&gt;</td>
<td>64-bit integer multiply</td>
</tr>
<tr>
<td>subTime &lt;time&gt;  &lt;time&gt;</td>
<td>subtract time</td>
</tr>
</tbody>
</table>
Tcl examples

Example 1

The following Tcl/ModelSim example for UNIX shows how you can access system information and transfer it into VHDL variables or signals and Verilog nets or registers. When a particular HDL source breakpoint occurs, a Tcl function is called that gets the date and time and deposits it into a VHDL signal of type STRING. If a particular environment variable (DO_ECHO) is set, the function also echoes the new date and time to the transcript file by examining the VHDL variable.

Note: In a Windows environment, the Tcl exec command shown below will execute compiled files only, not system commands.

(in VHDL source):

```vhdl
signal datime : string(1 to 28) := " "
```

(on VSIM command line or in macro):

```tcl
proc set_date {} {
    global env
    set do_the_echo [getenv DO_ECHO]
    set s [exec date]
    force -deposit datime $s
    if {do_the_echo} {
        echo "New time is [examine -value datime]"
    }
}
bp src/waveadd.vhd 133 {set_date; continue}
```

This is an example of using the Tcl while loop to copy a list from variable a to variable b, reversing the order of the elements along the way:

```tcl
set a [list 1 2 3 4]
set b [list]
set i [expr [llength $a]-1]
while {$i >= 0} {
    lappend b [lindex $a $i]
    incr i -1
}
```

This example uses the Tcl for command to copy a list from variable a to variable b, reversing the order of the elements along the way:

```tcl
set a [list 1 2 3 4]
set b [list]
for {set i [expr [llength $a] -1]} {$i >= 0} {incr i -1} {
    lappend b [lindex $a $i]
}
```

This example uses the Tcl foreach command to copy a list from variable a to variable b, reversing the order of the elements along the way (the foreach command iterates over all of the elements of a list):

```tcl
set a [list 1 2 3 4]
foreach i $a {
    set b [linsert $b 0 $i]
}
```
This example shows a list reversal as above, this time aborting on a particular element using the Tcl **break** command:

```tcl
set b ""
foreach i $a {
    if {$i = "ZZZ") break
    set b [linsert $b 0 $i]
}
```

This example is a list reversal that skips a particular element by using the Tcl **continue** command:

```tcl
set b ""
foreach i $a {
    if {$i = "ZZZ") continue
    set b [linsert $b 0 $i]
}
```

The last example is of the Tcl **switch** command:

```tcl
switch $x {
    a {incr t1}
    b {incr t2}
    c {incr t3}
}
```

## Example 2

This next example shows a complete Tcl script that restores multiple Wave windows to their state in a previous simulation, including signals listed, geometry, and screen position. It also adds buttons to the Main window toolbar to ease management of the wave files. This example works in ModelSim SE only.

```tcl
## This file contains procedures to manage multiple wave files.
## Source this file from the command line or as a startup script.
## source <path>/wave_mgr.tcl
## add_wave_buttons
##      Add wave management buttons to the main toolbar (new, save and load)
## new_wave
##      Dialog box creates a new wave window with the user provided name
## named_wave <name>
##      Creates a new wave window with the specified title
## save_wave <file-root>
##      Saves name, window location and contents for all open
## wave windows
##      Creates <file-root><n>.do file for each window where <n> is 1 to the number of windows. Default file-root is "wave". Also creates windowSet.do file that contains title and geometry info.
## load_wave <file-root>
##      Opens and loads wave windows for all files matching <file-root><n>.do where <n> are the numbers from 1-9. Default <file-root> is "wave". Also runs windowSet.do file if it exists.
```
## Add wave management buttons to the main toolbar

```tcl
proc add_wave_buttons {} {
    _add_menu main controls right SystemMenu SystemWindowFrame {Load Waves}
    load_wave
    _add_menu main controls right SystemMenu SystemWindowFrame {Save Waves}
    save_wave
    _add_menu main controls right SystemMenu SystemWindowFrame {New Wave}
    new_wave
}
```

## Simple Dialog requests name of new wave window.  Defaults to Wave<n>

```tcl
proc new_wave {} {
    global dialog_prompt vsimPriv
    set defaultName "Wave[llength $vsimPriv(WaveWindows)]"
    set dialog_prompt(result) $defaultName
    set windowName [GetValue . "Create Named Wave Window:" ]
    ## Debug
    puts "Window name: $windowName"
    if {$windowName == "{}"} {
        set windowName ""
    } else {
        named_wave $windowName
    } else {
        named_wave $defaultName
    }
}
```

## Creates a new wave window with the provided name (defaults to "Wave")

```tcl
proc named_wave {{name "Wave"}} {
    global vsimPriv
    view -new wave
    set newWave [lindex $vsimPriv(WaveWindows) [expr [llength $vsimPriv(WaveWindows)] - 1]]
    wm title $newWave $name
}
```

## Writes out format of all wave windows, stores geometry and title info in windowSet.do file.  Removes any extra files with the same fileroot.

```tcl
proc save_wave {{fileroot "wave"}} {
    global vsimPriv
    set n 1
    set fileId [open windowSet_$fileroot.do w 755]
    foreach w $vsimPriv(WaveWindows) {
        echo "Saving: [wm title $w]"
        set filename $fileroot$n.do
        write format wave -window $w $filename
        puts $fileId "wm title $w "[wm title $w"
        puts $fileId "wm geometry $w [wm geometry $w]"
        puts $fileId "mtiGrid_colconfig $w.grid name -width \"[mtiGrid_colcget $w.grid name -width]"
        puts $fileId "mtiGrid_colconfig $w.grid value -width \"[mtiGrid_colcget $w.grid value -width]"
        flush $fileId
        incr n
    }
    flush $fileId
    close $fileId
}
```
if {[catch {glob $fileroot\[$n-9\].do}]} {
    foreach f [lsort [glob $fileroot\[$n-9\].do]] {
        echo "Removing: $f"
        exec rm $f
    }
}

## Provide file root argument and load_wave restores all saved widows.
## Default file root is "wave".

proc load_wave {{fileroot "wave"}} {
    global vsimPriv
    foreach f [lsort [glob $fileroot\[1-9\].do]] {
        echo "Loading: $f"
        view -new wave
        do $f
    }
    if {[file exists windowSet_$fileroot.do]} {
        do windowSet_$fileroot.do
    }
}
Macros (DO files)

ModelSim macros (also called DO files) are simply scripts that contain ModelSim and, optionally, Tcl commands. You invoke these scripts with the Macro > Execute Macro (Main window) menu selection or the do command (CR-127).

Creating DO files

You can create DO files, like any other Tcl script, by typing the required commands in any editor and saving the file. Alternatively, you can save the Main window transcript to a DO file (see "Saving the Main window transcript file" (UM-153)).

The following is a simple DO file that was saved from the Main window transcript. It is used in the dataset exercise in the ModelSim Tutorial. This DO file adds several signals to the Wave window, provides stimulus to those signals, and then advances the simulation.

```tcl
add wave ld
add wave rst
add wave clk
add wave d
add wave q
force -freeze clk 0 0, 1 {50 ns} -r 100
force rst 1
force rst 0 10
force ld 0
force d 1010
run 1700
force ld 1
run 100
force ld 0
run 400
force rst 1
run 200
force rst 0 10
run 1500
```

Using Parameters with DO files

You can increase the flexibility of DO files using parameters. Parameters specify values that are passed to the corresponding parameters $1 through $9 in the macro file. For example,

```tcl
do testfile design.vhd 127
```

If the macro file testfile contains the line bp $1 $2, this command would place a breakpoint in the source file named design.vhd at line 127.

There is no limit on the number of parameters that can be passed to macros, but only nine values are visible at one time. You can use the shift command (CR-206) to see the other parameters.
Useful commands for handling breakpoints and errors

If you are executing a macro when your simulation hits a breakpoint or causes a run-time error, ModelSim interrupts the macro and returns control to the command line. The following commands may be useful for handling such events. (Any other legal command may be executed as well.)

<table>
<thead>
<tr>
<th>command</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>run (CR-199) -continue</td>
<td>continue as if the breakpoint had not been executed, completes the run (CR-199) that was interrupted</td>
</tr>
<tr>
<td>resume (CR-196)</td>
<td>continue running the macro</td>
</tr>
<tr>
<td>onbreak (CR-167)</td>
<td>specify a command to run when you hit a breakpoint within a macro</td>
</tr>
<tr>
<td>onElabError (CR-168)</td>
<td>specify a command to run when an error is encountered during elaboration</td>
</tr>
<tr>
<td>onerror (CR-169)</td>
<td>specify a command to run when an error is encountered within a macro</td>
</tr>
<tr>
<td>status (CR-209)</td>
<td>get a traceback of nested macro calls when a macro is interrupted</td>
</tr>
<tr>
<td>abort (CR-46)</td>
<td>terminate a macro once the macro has been interrupted or paused</td>
</tr>
<tr>
<td>pause (CR-170)</td>
<td>cause the macro to be interrupted, the macro can be resumed by entering a resume command (CR-196) via the command line</td>
</tr>
<tr>
<td>transcript (CR-217)</td>
<td>control echoing of macro commands to the Main window transcript</td>
</tr>
</tbody>
</table>

▶ **Note:** You can also set the OnErrorDefaultAction Tcl variable (see "Preference variables located in TCL files" (UM-396)) in the pref.tcl file to dictate what action ModelSim takes when an error occurs.

Error action in DO files

If a command in a macro returns an error, ModelSim does the following:

1. If an onerror (CR-169) command has been set in the macro script, ModelSim executes that command.

2. If no onerror command has been specified in the script, ModelSim checks the OnErrorDefaultAction Tcl variable. If the variable is defined, it will be invoked.

3. If neither 1 or 2 is true, the macro aborts.
Using the Tcl source command with DO files

Either the do command or Tcl source command can execute a DO file, but they behave differently.

With the source command, the DO file is executed exactly as if the commands in it were typed in by hand at the prompt. Each time a breakpoint is hit the Source window is updated to show the breakpoint. This behavior could be inconvenient with a large DO file containing many breakpoints.

When a do command is interrupted by an error or breakpoint, it does not update any windows, and keeps the DO file "locked". This keeps the Source window from flashing, scrolling, and moving the arrow when a complex DO file is executed. Typically an onbreak resume command is used to keep the macro running as it hits breakpoints. Add an onbreak abort command to the DO file if you want to exit the macro and update the Source window.

See also

See the do command (CR-127). Also see the DOPATH (UM-383) variable for adding a DO file path to your environment.
This appendix documents the following types of ModelSim variables:

- **environment variables**
  Variables referenced and set according to operating system conventions. Environment variables prepare the ModelSim environment prior to simulation.

- **ModelSim preference variables**
  Variables used to control compiler or simulator functions (usually in .tcl files) and modify the appearance of the ModelSim GUI (usually in INI files).

- **simulator state variables**
  Variables that provide feedback on the state of the current simulation.
Variable settings report

The `report` command returns a list of current settings for either the simulator state, or simulator control variables. Use the following commands at either the ModelSim or VSIM prompt:

```
report simulator state
report simulator control
```

Personal preferences

There are several preferences stored by ModelSim on a personal basis, independent of `modelsim.ini` or `modelsim.tcl` files. These preferences are stored in `$(HOME)/.modelsim` on UNIX and in the Windows Registry under HKEY_CURRENT_USER\Software\Model Technology Incorporated\ModelSim.

- **cwd**
  History of the last five working directories (pwd). This history appears in the Main window File menu.

- **phst**
  Project History

- **pinit**
  Project Initialization state (one of: Welcome | OpenLast | NoWelcome). This determines whether the Welcome To ModelSim dialog box appears when you invoke the tool.

- **printersetup**
  All setup parameters related to Printing (i.e., current printer, etc.)

The HKEY_CURRENT_USER key is unique for each user Login on Windows NT.

Returning to the original ModelSim defaults

If you would like to return ModelSim’s interface to its original state, simply rename or delete the existing `modelsim.tcl` and `modelsim.ini` files. ModelSim will use `pref.tcl` for GUI preferences and make a copy of `<install_dir>/modeltech/modelsim.ini` to use the next time ModelSim is invoked without an existing project (if you start a new project the new MPF file will use the settings in the new `modelsim.ini` file).
Environment variables

Before compiling or simulating, several environment variables may be set to provide the functions described in the table below. The variables are in the autoexec.bat file on Windows 95/98 machines, and set through the System control panel on NT machines. For UNIX, the variables are typically found in the .login script. The LM_LICENSE_FILE variable is required, all others are optional.

### ModelSim Environment Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOPATH</td>
<td>Used by ModelSim to search for simulator command files (do files); consists of a colon-separated (semi-colon for Windows) list of paths to directories; optional; this variable can be overridden by the DOPATH .tcl file variable</td>
</tr>
<tr>
<td>EDITOR</td>
<td>Specifies the editor to invoke with the edit command (CR-133)</td>
</tr>
<tr>
<td>HOME</td>
<td>Used by ModelSim to look for an optional graphical preference file and optional location map file; see: &quot;Preference variables located IN INI files&quot; (UM-386) and &quot;Using location mapping&quot; (UM-425)</td>
</tr>
<tr>
<td>LM_LICENSE_FILE</td>
<td>Used by the ModelSim license file manager to find the location of the license file; may be a colon-separated (semi-colon for Windows) set of paths, including paths to other vendor license files; REQUIRED; see: &quot;Using the FLEXlm License Manager&quot; (UM-407)</td>
</tr>
<tr>
<td>MODEL_TECH</td>
<td>Set by all ModelSim tools to the directory in which the binary executables reside; YOU SHOULD NOT SET THIS VARIABLE</td>
</tr>
<tr>
<td>MODEL_TECH_TCL</td>
<td>Used by ModelSim to find Tcl libraries for: Tcl/Tk 8.0, Tix, and vsim; defaults to /modeltech/../tcl; may be set to an alternate path</td>
</tr>
<tr>
<td>MGC_LOCATION_MAP</td>
<td>Used by ModelSim tools to find source files based on easily reallocated &quot;soft&quot; paths; optional; see: &quot;Using location mapping&quot; (UM-425); also see the Tcl variables: SourceDir and SourceMap</td>
</tr>
<tr>
<td>MODELSIM</td>
<td>Used by all ModelSim tools to find the modelsim.ini file; consists of a path including the file name; optional. An alternative use of this variable is to set it to the path of a project file (Project_Root_Dir/Project_Name.mpf). This allows you to use project settings with command line tools. However, if you do this, the .mpf file will replace modelsim.ini as the initialization file for all ModelSim tools.</td>
</tr>
<tr>
<td>MODELSIM_TCL</td>
<td>Used by ModelSim to look for an optional graphical preference file; can be a colon-separated (UNIX) or semi-colon (Windows) separated list of file paths</td>
</tr>
<tr>
<td>MTI_TF_LIMIT</td>
<td>Limits the size of the VSOUT temp file (generated by the ModelSim kernel); the value of the variable is the size of k-bytes; TMPDIR (below) controls the location of this file, STDOUT controls the name; default = 10, 0 = no limit</td>
</tr>
<tr>
<td>MTI_USELIB_DIR</td>
<td>Specifies the directory into which object libraries are compiled when using the -compile_uselibs argument to the vlog command (CR-274)</td>
</tr>
</tbody>
</table>
Creating environment variables in Windows

In addition to the predefined variables shown above, you can define your own environment variables. This example shows a user-defined library path variable that can be referenced by the `vmap` command to add library mapping to the `modelsim.ini` file.

**Using Windows 95/98/Me**

Open and edit the `autoexec.bat` file by adding this line:

```bash
set MY_PATH=\temp\work
```

Restart Windows to initialize the new variable.

**Using Windows NT/2000**

Right-click the My Computer icon and select Properties, then select the Environment tab (in Windows 2000 select the Advanced tab and then Environment Variables). Add the new variable with this data—Variable:MY_PATH and Value:temp\work.

Click Set and Apply to initialize the variable (you don’t need to restart NT).

**Library mapping with environment variables**

Once the MY_PATH variable is set, you can use it with the `vmap` command (CR-283) to add library mappings to the current `modelsim.ini` file.

If you’re using the `vmap` command from DOS prompt type:

```bash
vmap MY_VITAL %MY_PATH%
```

If you’re using `vmap` from ModelSim/VSIM prompt type:

```bash
vmap MY_VITAL $MY_PATH
```

If you used DOS `vmap`, this line will be added to the `modelsim.ini`:

```
MY_VITAL = c:\temp\work
```

If `vmap` is used from the ModelSim/VSIM prompt, the `modelsim.ini` file will be modified with this line:

```
MY_VITAL = $MY_PATH
```
You can easily add additional hierarchy to the path. For example,

vmap MORE_VITAL %MY_PATH\more_path\and_more_path
vmap MORE_VITAL $MY_PATH\more_path\and_more_path

▶ Note: The "$" character in the examples above is Tcl syntax that precedes a variable. The "\" character is an escape character that keeps the variable from being evaluated during the execution of `vmap`.

Referencing environment variables within ModelSim

There are two ways to reference environment variables within ModelSim. Environment variables are allowed in a FILE variable being opened in VHDL. For example,

```vhdl
entity test is end;
use std.textio.all;
architecture only of test is
begin
process
  FILE in_file : text is in "$ENV_VAR_NAME";
begin
  wait;
end process;
end;
```

Environment variables may also be referenced from the ModelSim command line or in macros using the Tcl `env` array mechanism:

```tcl
echo "$env(ENV_VAR_NAME)"
```

Removing temp files (VSOUT)

The VSOUT temp file is the communication mechanism between the simulator kernel and the ModelSim GUI. In normal circumstances the file is deleted when the simulator exits. If ModelSim crashes, however, the temp file must be deleted manually. Specifying the location of the temp file with `TMPDIR` (above) will help you locate and remove the file.

▶ Note: There is one environment variable, `MODEL_TECH`, that you cannot — and should not — set. `MODEL_TECH` is a special variable set by Model Technology software. Its value is the name of the directory from which the `vcom` compiler or `vsim` simulator was invoked. `MODEL_TECH` is used by the other Model Technology tools to find the libraries.
Preference variables located in INI files

ModelSim initialization (INI) files contain control variables that specify reference library paths and compiler and simulator settings. See “System initialization” (UM-33) for more details on how these variables are loaded.

The following tables list the variables by section, and in order of their appearance within the INI file:

<table>
<thead>
<tr>
<th>INI file sections</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Library] library path variables (UM-386)</td>
</tr>
<tr>
<td>[vcom] VHDL compiler control variables (UM-386)</td>
</tr>
<tr>
<td>[vlog] Verilog compiler control variables (UM-388)</td>
</tr>
<tr>
<td>[vsim] simulator control variables (UM-388)</td>
</tr>
<tr>
<td>[lmc] Logic Modeling variables (UM-392)</td>
</tr>
</tbody>
</table>

[Library] library path variables

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Value range</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ieee</td>
<td>any valid path; may include environment variables</td>
<td>sets the path to the library containing IEEE and Synopsys arithmetic packages; the default is /modeltech/../ieee</td>
</tr>
<tr>
<td>std</td>
<td>any valid path; may include environment variables</td>
<td>sets the path to the VHDL STD library; the default is /modeltech/../std</td>
</tr>
<tr>
<td>std_developerskit</td>
<td>any valid path; may include environment variables</td>
<td>sets the path to the libraries for MGC standard developer’s kit; the default is /modeltech/../std_developerskit</td>
</tr>
<tr>
<td>synopsys</td>
<td>any valid path; may include environment variables</td>
<td>sets the path to the accelerated arithmetic packages; the default is /modeltech/../synopsys</td>
</tr>
<tr>
<td>verilog</td>
<td>any valid path; may include environment variables</td>
<td>sets the path to the library containing VHDL/Verilog type mappings; the default is /modeltech/../verilog</td>
</tr>
</tbody>
</table>

[vcom] VHDL compiler control variables

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Value range</th>
<th>Purpose</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckSynthesis</td>
<td>0, 1</td>
<td>if 1, turns on limited synthesis rule compliance checking; checks only signals used (read) by a process</td>
<td>off (0)</td>
</tr>
<tr>
<td>Variable name</td>
<td>Value range</td>
<td>Purpose</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------</td>
<td>-------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Explicit</td>
<td>0, 1</td>
<td>if 1, turns on resolving of ambiguous function overloading in favor of the &quot;explicit&quot; function declaration (not the one automatically created by the compiler for each type declaration)</td>
<td>on (1)</td>
</tr>
<tr>
<td>IgnoreVitalErrors</td>
<td>0, 1</td>
<td>if 1, ignores VITAL compliance checking errors</td>
<td>off (0)</td>
</tr>
<tr>
<td>NoCaseStaticError</td>
<td>0, 1</td>
<td>if 1, changes case statement static errors to warnings</td>
<td>off (0)</td>
</tr>
<tr>
<td>NoDebug</td>
<td>0, 1</td>
<td>if 1, turns off inclusion of debugging info within design units</td>
<td>off (0)</td>
</tr>
<tr>
<td>NoOthersStaticError</td>
<td>0, 1</td>
<td>if 1, disables errors caused by aggregates that are not locally static</td>
<td>off (0)</td>
</tr>
<tr>
<td>NoVital</td>
<td>0, 1</td>
<td>if 1, turns off acceleration of the VITAL packages</td>
<td>off (0)</td>
</tr>
<tr>
<td>NoVitalCheck</td>
<td>0, 1</td>
<td>if 1, turns off VITAL compliance checking</td>
<td>off (0)</td>
</tr>
<tr>
<td>Optimize_1164</td>
<td>0, 1</td>
<td>if 0, turns off optimization for IEEE std_logic_1164 package</td>
<td>on (1)</td>
</tr>
<tr>
<td>Quiet</td>
<td>0, 1</td>
<td>if 1, turns off &quot;loading...&quot; messages</td>
<td>off (0)</td>
</tr>
<tr>
<td>RequireConfigForAllDefault</td>
<td>0, 1</td>
<td>if 1, instructs the compiler not to generate a default binding during compilation</td>
<td>off (0)</td>
</tr>
<tr>
<td>Scalar_OPTS</td>
<td>0, 1</td>
<td>if 1, activates optimizations on expressions that don’t involve signals, waits or function/procedure/task invocations</td>
<td>off (0)</td>
</tr>
<tr>
<td>Show_source</td>
<td>0, 1</td>
<td>if 1, shows source line containing error</td>
<td>off (0)</td>
</tr>
<tr>
<td>Show_VitalChecksWarnings</td>
<td>0, 1</td>
<td>if 0, turns off VITAL compliance-check warnings</td>
<td>on (1)</td>
</tr>
<tr>
<td>Show_Warning1</td>
<td>0, 1</td>
<td>if 0, turns off unbound-component warnings</td>
<td>on (1)</td>
</tr>
<tr>
<td>Show_Warning2</td>
<td>0, 1</td>
<td>if 0, turns off process-without-a-wait-statement warnings</td>
<td>on (1)</td>
</tr>
<tr>
<td>Show_Warning3</td>
<td>0, 1</td>
<td>if 0, turns off null-range warnings</td>
<td>on (1)</td>
</tr>
<tr>
<td>Show_Warning4</td>
<td>0, 1</td>
<td>if 0, turns off no-space-in-time-literal warnings</td>
<td>on (1)</td>
</tr>
<tr>
<td>Show_Warning5</td>
<td>0, 1</td>
<td>if 0, turns off multiple-drivers-on-unresolved-signal warnings</td>
<td>on (1)</td>
</tr>
<tr>
<td>VHDL93</td>
<td>0, 1</td>
<td>if 1, turns on VHDL-1993</td>
<td>off (0)</td>
</tr>
</tbody>
</table>
### [vlog] Verilog compiler control variables

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Value range</th>
<th>Purpose</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hazard</td>
<td>0, 1</td>
<td>if 1, turns on Verilog hazard checking (order-dependent accessing of global vars)</td>
<td>off (0)</td>
</tr>
<tr>
<td>Incremental</td>
<td>0, 1</td>
<td>if 1, turns on incremental compilation of modules</td>
<td>off (0)</td>
</tr>
<tr>
<td>NoDebug</td>
<td>0, 1</td>
<td>if 1, turns off inclusion of debugging info within design units</td>
<td>off (0)</td>
</tr>
<tr>
<td>Quiet</td>
<td>0, 1</td>
<td>if 1, turns off &quot;loading...&quot; messages</td>
<td>off (0)</td>
</tr>
<tr>
<td>Show_Lint</td>
<td>0, 1</td>
<td>if 1, turns on lint-style checking</td>
<td>off (0)</td>
</tr>
<tr>
<td>ScalarOpts</td>
<td>0, 1</td>
<td>if 1, activates optimizations on expressions that don’t involve signals, waits or function/procedure/task invocations</td>
<td>off (0)</td>
</tr>
<tr>
<td>Show_source</td>
<td>0, 1</td>
<td>if 1, shows source line containing error</td>
<td>off (0)</td>
</tr>
<tr>
<td>UpCase</td>
<td>0, 1</td>
<td>if 1, turns on converting regular Verilog identifiers to uppercase. Allows case insensitivity for module names; see also &quot;Verilog-XL compatible compiler options&quot; (UM-73)</td>
<td>off (0)</td>
</tr>
</tbody>
</table>

### [vsim] simulator control variables

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Value range</th>
<th>Purpose</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssertFile</td>
<td>any valid filename</td>
<td>alternative file for storing assertion messages</td>
<td>transcript</td>
</tr>
<tr>
<td>AssertionFormat</td>
<td>see purpose</td>
<td>sets the message to display after a break on assertion; message formats include: %S - severity level, %R - report message, %I - instance or region pathname (if available), %D - delta, %T - time of assertion %D - delta, %I - instance or region pathname, %D - delta</td>
<td>&quot;** %S: %R\n Time: %T\n Iteration: %D%I\n&quot;</td>
</tr>
<tr>
<td>BreakOnAssertion</td>
<td>0-4</td>
<td>defines severity of assertion that causes a simulation break (0 = note, 1 = warning, 2 = error, 3 = failure, 4 = fatal)</td>
<td>3</td>
</tr>
<tr>
<td>CheckpointCompressMode</td>
<td>0, 1</td>
<td>if 1, checkpoint files are written in compressed format</td>
<td>on (1)</td>
</tr>
<tr>
<td>CommandHistory</td>
<td>any valid filename</td>
<td>sets the name of a file in which to store the Main window command history</td>
<td>commented out (;)</td>
</tr>
<tr>
<td>Variable name</td>
<td>Value range</td>
<td>Purpose</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>ConcurrentFileLimit</td>
<td>any positive integer</td>
<td>controls the number of VHDL files open concurrently; this number should be less than the current limit setting for max file descriptors; 0 = unlimited</td>
<td>40</td>
</tr>
<tr>
<td>DatasetSeparator</td>
<td>any single character</td>
<td>the dataset separator for fully-rooted contexts, for example sim:/top; must not be the same character as PathSeparator</td>
<td>:</td>
</tr>
<tr>
<td>DefaultForceKind</td>
<td>freeze, drive, or deposit</td>
<td>defines the kind of force used when not otherwise specified</td>
<td>drive for resolved signals; freeze for unresolved signals</td>
</tr>
<tr>
<td>DefaultRadix</td>
<td>symbolic, binary, octal, decimal, unsigned, hexadecimal, ascii</td>
<td>any radix may be specified as a number or name (i.e., binary can be specified as binary or 2)</td>
<td>symbolic</td>
</tr>
<tr>
<td>DefaultRestartOptions</td>
<td>one or more of: -force, -nobreakpoint, -nolist, -nolog, -nowave</td>
<td>sets default behavior for the restart command</td>
<td>commented out (;)</td>
</tr>
<tr>
<td>DelayFileOpen</td>
<td>0, 1</td>
<td>if 1, open VHDL87 files on first read or write, else open files when elaborated</td>
<td>off (0)</td>
</tr>
<tr>
<td>GenerateFormat</td>
<td>Any non-quoted string containing at a minimum a %s followed by a %d</td>
<td>control the format of a generate statement label (don’t quote it)</td>
<td>%s__%d</td>
</tr>
<tr>
<td>IgnoreError</td>
<td>0,1</td>
<td>if 1, ignore assertion errors</td>
<td>off (0)</td>
</tr>
<tr>
<td>IgnoreFailure</td>
<td>0,1</td>
<td>if 1, ignore assertion failures</td>
<td>off (0)</td>
</tr>
<tr>
<td>IgnoreNote</td>
<td>0,1</td>
<td>if 1, ignore assertion notes</td>
<td>off (0)</td>
</tr>
<tr>
<td>IgnoreWarning</td>
<td>0,1</td>
<td>if 1, ignore assertion warnings</td>
<td>off (0)</td>
</tr>
<tr>
<td>IterationLimit</td>
<td>positive integer</td>
<td>limit on simulation kernel iterations during one time delta</td>
<td>5000</td>
</tr>
<tr>
<td>Variable name</td>
<td>Value range</td>
<td>Purpose</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>License</td>
<td>any single &lt;license_option&gt;</td>
<td>if set, controls ModelSim license file search; license options include:</td>
<td>search all licenses</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nomgc - excludes MGC licenses</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>nomti - excludes MTI licenses</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>noqueue - do not wait in license queue if no licenses are available</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>plus - only use PLUS license</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>vlog - only use VLOG license</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>vhdl - only use VHDL license</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>viewsim - accepts a simulation license rather than being queued for a</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>viewer license</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>see also the vsim command (CR-284)</td>
<td></td>
</tr>
<tr>
<td>LockedMemory</td>
<td>positive integer; mb of memory to</td>
<td>for HP-UX 10.2 use only; enables memory locking to speed up large</td>
<td>disabled</td>
</tr>
<tr>
<td></td>
<td>lock</td>
<td>designs (&gt;500mb memory footprint); see &quot;Improve performance by locking</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>memory on HP-UX 10.2&quot; (UM-427)</td>
<td></td>
</tr>
<tr>
<td>NumericStdNoWarnings</td>
<td>0, 1</td>
<td>if 1, warnings generated within the accelerated numeric_std and</td>
<td>off (0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>numeric_bit packages are suppressed</td>
<td></td>
</tr>
<tr>
<td>PathSeparator</td>
<td>any single character</td>
<td>used for hierarchical path names; must not be the same character as</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DatasetSeparator</td>
<td></td>
</tr>
<tr>
<td>Resolution</td>
<td>fs, ps, ns, us, ms, or sec with</td>
<td>simulator resolution; this value must be less than or equal to the</td>
<td>ns</td>
</tr>
<tr>
<td></td>
<td>optional prefix of 1, 10, or 100</td>
<td>UserTimeUnit specified below; NOTE - if your delays are truncated,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>set the resolution smaller; no space between value and units (i.e.,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10fs, not 10 fs)</td>
<td></td>
</tr>
<tr>
<td>RunLength</td>
<td>positive integer</td>
<td>default simulation length in units specified by the UserTimeUnit</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>variable</td>
<td></td>
</tr>
<tr>
<td>Startup</td>
<td>= do &lt;DO filename&gt;; any valid</td>
<td>specifies the ModelSim startup macro; see the do command (CR-127)</td>
<td>commented out (;)</td>
</tr>
<tr>
<td></td>
<td>macro (do) file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>StdArithNoWarnings</td>
<td>0, 1</td>
<td>if 1, warnings generated within the accelerated Synopsys std_arith</td>
<td>off (0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>packages are suppressed</td>
<td></td>
</tr>
<tr>
<td>TranscriptFile</td>
<td>any valid filename</td>
<td>file for saving command transcript; environment variables may be</td>
<td>transcript</td>
</tr>
<tr>
<td></td>
<td></td>
<td>included in the path name</td>
<td></td>
</tr>
<tr>
<td>Variable name</td>
<td>Value range</td>
<td>Purpose</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>UnbufferedOutput</td>
<td>0, 1</td>
<td>controls VHDL and Verilog files open for write; 0 = Buffered, 1 = Unbuffered</td>
<td>0</td>
</tr>
<tr>
<td>UserTimeUnit</td>
<td>fs, ps, ns, us, ms, sec, or default</td>
<td>specifies the default units to use for the &quot;&lt;timesteps&gt; [&lt;time_units&gt;]&quot; argument to the run command (CR-199); NOTE - the value of this variable must be set equal to, or larger than, the current simulator resolution specified by the Resolution variable shown above</td>
<td>ns</td>
</tr>
<tr>
<td>Veriuser</td>
<td>one or more valid shared objects</td>
<td>list of dynamically loadable objects for Verilog PLI/VPI applications; see &quot;Verilog PLI/VPI&quot; (UM-101)</td>
<td>commented out (;)</td>
</tr>
<tr>
<td>WaveSignalNameWidth</td>
<td>0, positive integer</td>
<td>controls the number of visible hierarchical regions of a signal name shown in the Wave window (UM-210); the default value of zero displays the full name, a setting of one or above displays the corresponding level(s) of hierarchy</td>
<td>0</td>
</tr>
<tr>
<td>WLFCompress</td>
<td>0, 1</td>
<td>turns WLF file compression on (1) or off (0)</td>
<td>1</td>
</tr>
<tr>
<td>WLFDeleteOnQuit</td>
<td>0, 1</td>
<td>specifies whether a WLF file should be deleted when the simulation ends; if set to 0, the file is not deleted; if set to 1, the file is deleted</td>
<td>0</td>
</tr>
<tr>
<td>WLFSaveAllRegions</td>
<td>0, 1</td>
<td>specifies whether to save all design hierarchy in the WLF file (1) or only regions containing logged signals (0)</td>
<td>0</td>
</tr>
<tr>
<td>WLFSizeLimit</td>
<td>0 - n MB</td>
<td>WLF file size limit; limits WLF file by size (as closely as possible) to the specified number of megabytes; if both size and time limits are specified the most restrictive is used; setting to 0 results in no limit</td>
<td>0</td>
</tr>
<tr>
<td>WLFTimeLimit</td>
<td>0 - n</td>
<td>WLF file time limit; limits WLF file by time (as closely as possible) to the specified amount of time. If both time and size limits are specified the most restrictive is used; setting to 0 results in no limit</td>
<td>0</td>
</tr>
</tbody>
</table>
[imc] Logic Modeling variables

**Logic Modeling SmartModels and hardware modeler interface**

ModelSim’s interface with Logic Modeling’s SmartModels and hardware modeler are specified in the [imc] section of the INI/MPF file; for more information see “VHDL SmartModel interface” (UM-346) and “VHDL Hardware Model interface” (UM-356) respectively.

**Spaces in path names**

For the Src_Files and Work_Libs variables, each element in the list is enclosed within curly braces ({ }). This allows spaces inside elements (since Windows allows spaces inside path names). For example a source file list might look like:

```
Src_Files = {${MODELSIM_PROJECT}/counter.v} {${MODELSIM_PROJECT}/tb counter.v}
```

Where the file `tb counter.v` contains a space character between the "b" and "c".

**Setting variables in INI files**

Edit the initialization file directly with any text editor to change or add a variable. The syntax for variables in the file is:

```
<variable> = <value>
```

Comments within the file are preceded with a semicolon (;).

▶ **Note**: The `vmap` command (CR-283) automatically modifies library mappings in the current INI file.

**Reading variable values from the INI file**

These Tcl functions allow you to read values from the `modelsim.ini` file.

- `GetIniInt <var_name> <default_value>`
  - Reads the integer value for the specified variable.

- `GetIniReal <var_name> <default_value>`
  - Reads the real value for the specified variable.

- `GetProfileString <section> <var_name> [<default>]`
  - Reads the string value for the specified variable in the specified section. Optionally provides a default value if no value is present.

Setting Tcl variables with values from the `modelsim.ini` file is one use of these Tcl functions. For example,

```
set MyCheckpointCompressMode [GetIniInt "CheckpointCompressMode" 1]
```

```
set PrefMain(file) [GetProfileString vsim TranscriptFile ""]
```
Commonly used INI variables

Several of the more commonly used *modelsim.ini* variables are further explained below.

**Environment variables**

You can use environment variables in your initialization files. Use a dollar sign ($) before the environment variable name.

**Examples**

```
[Library]
work = $HOME/work_lib
test_lib = ./$TESTNUM/work
...
[vsim]
IgnoreNote = $IGNORE_ASSERTS
IgnoreWarning = $IGNORE_ASSERTS
IgnoreError = 0
IgnoreFailure = 0
```

Tip:

There is one environment variable, MODEL_TECH, that you cannot — and should not — set. MODEL_TECH is a special variable set by Model Technology software. Its value is the name of the directory from which the VCOM compiler or VSIM simulator was invoked. MODEL_TECH is used by the other Model Technology tools to find the libraries.

**Hierarchical library mapping**

By adding an "others" clause to your *modelsim.ini* file, you can have a hierarchy of library mappings. If the ModelSim tools don’t find a mapping in the *modelsim.ini* file, then they will search just the library section of the initialization file specified by the "others" clause.

**Examples**

```
[Library]
asic_lib = /cae/asic_lib
work = my_work
others = /install_dir/modeltech/modelsim.ini
```

Tip:

Since the file referred to by the "others" clause may itself contain an "others" clause, you can use this feature to chain a set of hierarchical INI files for library mappings.

**Creating a transcript file**

A feature in the system initialization file allows you to keep a record of everything that occurs in the transcript: error messages, assertions, commands, command outputs, etc. To do this, set the value for the TranscriptFile line in the *modelsim.ini* file to the name of the file in which you would like to record the ModelSim history.

```
; Save the command window contents to this file
TranscriptFile = trnscrpt
```
Using a startup file

The system initialization file allows you to specify a command or a *do* file that is to be executed after the design is loaded. For example:

```verbatim
; VSIM Startup command
Startup = do mystartup.do
```

The line shown above instructs ModelSim to execute the commands in the macro file named *mystartup.do*.

```verbatim
; VSIM Startup command
Startup = run -all
```

The line shown above instructs VSIM to run until there are no events scheduled.

See the *do* command (CR-127) for additional information on creating do files.

Turning off assertion messages

You can turn off assertion messages from your VHDL code by setting a switch in the *modelsim.ini* file. This option was added because some utility packages print a huge number of warnings.

```verbatim
[vsim]
IgnoreNote = 1
IgnoreWarning = 1
IgnoreError = 1
IgnoreFailure = 1
```

Messages may also be turned off with Tcl variables; see "Preference variables located in TCL files" (UM-396).

Turning off warnings from arithmetic packages

You can disable warnings from the synopsys and numeric standard packages by adding the following lines to the [vsim] section of the *modelsim.ini* file.

```verbatim
[vsim]
NumericStdNoWarnings = 1
StdArithNoWarnings = 1
```

Warnings may also be turned off with Tcl variables; see "Preference variables located in TCL files" (UM-396).

Force command defaults

The *force* command has *-freeze*, *-drive*, and *-deposit* options. When none of these is specified, then *-freeze* is assumed for unresolved signals and *-drive* is assumed for resolved signals. This is designed to provide compatibility with version 4.1 and earlier force files. But if you prefer *-freeze* as the default for both resolved and unresolved signals, you can change the defaults in the *modelsim.ini* file.

```verbatim
[vsim]
; Default Force Kind
; The choices are freeze, drive, or deposit
DefaultForceKind = freeze
```
**Restart command defaults**

The `restart` command has `-force`, `-nobreakpoint`, `-nolist`, `-nolog`, and `-nowave` options. You can set any of these as defaults by entering the following line in the `modelsim.ini` file:

```
DefaultRestartOptions = <options>
```

where `<options>` can be one or more of `-force`, `-nobreakpoint`, `-nolist`, `-nolog`, and `-nowave`.

**Example:** `DefaultRestartOptions = -nolog -force`

**Note:** You can also set these defaults in the `modelsim.tcl` file. The Tcl file settings will override the `.ini` file settings.

**VHDL93**

You can make the VHDL93 standard the default by including the following line in the `INI` file:

```
[vcom]
; Turn on VHDL-1993 as the default. Default is off (VHDL-1987).
VHDL93 = 1
```

**Opening VHDL files**

You can delay the opening of VHDL files with an entry in the `INI` file if you wish. Normally VHDL files are opened when the file declaration is elaborated. If the `DelayFileOpen` option is enabled, then the file is not opened until the first read or write to that file.

```
[vsim]
DelayFileOpen = 1
```
Preference variables located in TCL files

ModelSim TCL preference variables give you control over fonts, colors, prompts, window positions and other simulator window characteristics. Preference files, which contain Tcl commands that set preference variables, are loaded before any windows are created, and so will affect all windows. For complete documentation on Tcl preference variables, see the following URL:

http://www.model.com/resources/pref_variables/frameset.htm

When ModelSim is invoked for the first time, default preferences are loaded from the pref.tcl file. (See "System initialization" (UM-33) for more details.) Customized variable settings may be set from within the ModelSim GUI, on the ModelSim command line, or by directly editing the preference file.

The default file for customized preferences is modelsim.tcl. If your preference file is not named modelsim.tcl, you must refer to it with the MODELSIM_TCL (UM-383) environment variable.

User-defined variables

Temporary, user-defined variables can be created with the Tcl set command. Like simulator variables, user-defined variables are preceded by a dollar sign when referenced.

To create a variable with the set command:

```
set user1 7
```

You can use the variable in a command like:

```
echo "user1 = $user1"
```

More preferences

Additional compiler and simulator preferences may be set in the modelsim.ini and MPF files; see "Preference variables located in INI files" (UM-386).
Variable precedence

Note that some variables can be set in a .tcl file or a .ini file. A variable set in a .tcl file takes precedent over the same variable set in a .ini file. For example, assume you have the following line in your modelsim.ini file:

```
TranscriptFile = transcript
```

And assume you have the following line in your modelsim.tcl file:

```
set PrefMain(file) {}
```

In this case the setting in the modelsim.tcl file will override that in the modelsim.ini file, and a transcript file will not be produced.
Simulator state variables

Unlike other variables that must be explicitly set, simulator state variables return a value relative to the current simulation. Simulator state variables can be useful in commands, especially when used within ModelSim DO files (macros).

<table>
<thead>
<tr>
<th>Variable</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>argc</td>
<td>returns the total number of parameters passed to the current macro</td>
</tr>
<tr>
<td>architecture</td>
<td>returns the name of the top-level architecture currently being simulated; for a configuration or Verilog module, this variable returns an empty string</td>
</tr>
<tr>
<td>configuration</td>
<td>returns the name of the top-level configuration currently being simulated; returns an empty string if no configuration</td>
</tr>
<tr>
<td>delta</td>
<td>returns the number of the current simulator iteration</td>
</tr>
<tr>
<td>entity</td>
<td>returns the name of the top-level VHDL entity or Verilog module currently being simulated</td>
</tr>
<tr>
<td>library</td>
<td>returns the library name for the current region</td>
</tr>
<tr>
<td>MacroNestingLevel</td>
<td>returns the current depth of macro call nesting</td>
</tr>
<tr>
<td>n</td>
<td>represents a macro parameter, where n can be an integer in the range 1-9</td>
</tr>
<tr>
<td>Now</td>
<td>returns the current simulation time expressed in the current time resolution (e.g., 1000 ns)</td>
</tr>
<tr>
<td>now</td>
<td>returns the current simulation time as an absolute number of time steps (e.g., 1000)</td>
</tr>
<tr>
<td>resolution</td>
<td>returns the current simulation time resolution</td>
</tr>
</tbody>
</table>

Referencing simulator state variables

Variable values may be referenced in simulator commands by preceding the variable name with a $ sign. For example, to use the now and resolution variables in an echo command type:

```
echo "The time is $now $resolution."
```

Depending on the current simulator state, this command could result in:

```
The time is 12390 10ps.
```

If you do not want the dollar sign to denote a simulator variable, precede it with a \". For example, \$now will not be interpreted as the current simulator time.
Special considerations for $now

The $now variable is set within ModelSim by a procedure that converts the current simulator time to user-time units, as specified in the -t argument to vsim command (CR-284). When no multiplier is specified with the time unit (e.g., 1ps), the procedure formats $now without a time unit. For example:

```
ModelSim> vsim -t 1ps
VSIM > echo $now
# 0
```

However, when a multiplier is specified (e.g., 10ps), it's difficult to know how it should behave for a given simulation time. For example, if the current simulation time is 500ps, and resolution is 10ps, should $now be 50 or 500 or 500ps? To remove any ambiguity ModelSim prints the 3rd alternative. For example:

```
ModelSim> vsim -t 10ps
VSIM > echo $now
# 0 ps
```

For the when command (CR-298), special processing is performed on comparisons involving the $now variable. If you specify "when ($now = 100)...", the simulator will stop at time 100, regardless of the multiplier applied to the time unit.
Appendix contents

Wave window mouse and keyboard shortcuts . . . . . . . . . . UM-401
List window keyboard shortcuts . . . . . . . . . . . . . . . . . UM-402
Command shortcuts . . . . . . . . . . . . . . . . . . . . . . . . UM-403
Command history shortcuts . . . . . . . . . . . . . . . . . . . UM-403
Mouse and keyboard shortcuts in Main and Source windows . . UM-403
Right mouse button . . . . . . . . . . . . . . . . . . . . . . . . UM-405

This appendix is a collection of the keyboard and command shortcuts available in the ModelSim GUI.

Wave window mouse and keyboard shortcuts

The following mouse actions and keystrokes can be used in the Wave window.

<table>
<thead>
<tr>
<th>Mouse action</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; control - left-button - click on a scroll arrow &gt;</td>
<td>scrolls window to very top or bottom (vertical scroll) or far left or right (horizontal scroll)</td>
</tr>
<tr>
<td>&lt; middle mouse-button - click in scroll bar trough&gt; (UNIX) only</td>
<td>scrolls window to position of click</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Keystroke</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>i I or +</td>
<td>zoom in</td>
</tr>
<tr>
<td>o O or -</td>
<td>zoom out</td>
</tr>
<tr>
<td>f or F</td>
<td>zoom full; mouse pointer must be over the the cursor or waveform panes</td>
</tr>
<tr>
<td>l or L</td>
<td>zoom last</td>
</tr>
<tr>
<td>r or R</td>
<td>zoom range</td>
</tr>
<tr>
<td>&lt;arrow up&gt;</td>
<td>scroll waveform display up by selecting the item above the currently selected item</td>
</tr>
</tbody>
</table>
List window keyboard shortcuts

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;arrow up&gt;</td>
<td>scroll listing up (selects and highlights the line above the currently selected line)</td>
</tr>
<tr>
<td>&lt;arrow down&gt;</td>
<td>scroll listing down (selects and highlights the line below the currently selected line)</td>
</tr>
<tr>
<td>&lt;arrow left&gt;</td>
<td>scroll listing left</td>
</tr>
<tr>
<td>&lt;arrow right&gt;</td>
<td>scroll listing right</td>
</tr>
<tr>
<td>&lt;page up&gt;</td>
<td>scroll listing up by page</td>
</tr>
<tr>
<td>&lt;page down&gt;</td>
<td>scroll listing down by page</td>
</tr>
<tr>
<td>&lt;tab&gt;</td>
<td>searches forward (down) to the next transition on the selected signal</td>
</tr>
<tr>
<td>&lt;shift-tab&gt;</td>
<td>searches backward (up) to the previous transition on the selected signal (does not function on HP workstations)</td>
</tr>
<tr>
<td>&lt;control-f&gt; Windows &lt;control-s&gt; UNIX</td>
<td>opens the find dialog box; finds the specified item label within the list display</td>
</tr>
</tbody>
</table>
Command shortcuts

You may abbreviate command syntax, but there’s a catch — the minimum characters required to execute a command are those that make it unique. Remember, as we add new commands some of the old shortcuts may not work. For this reason ModelSim does not allow command name abbreviations in macro files. This minimizes your need to maintain macro files as new commands are added.

Command history shortcuts

The simulator command history may be reviewed, or commands may be reused, with these shortcuts at the ModelSim/VSIM prompt:

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!!</td>
<td>repeats the last command</td>
</tr>
<tr>
<td>!n</td>
<td>repeats command number n; n is the VSIM prompt number (e.g., for this prompt: VSIM 12&gt;, n =12)</td>
</tr>
<tr>
<td>!abc</td>
<td>repeats the most recent command starting with &quot;abc&quot;</td>
</tr>
<tr>
<td>^xyz^ab^</td>
<td>replaces &quot;xyz&quot; in the last command with &quot;ab&quot;</td>
</tr>
<tr>
<td>up and down arrows</td>
<td>scrolls through the command history with the keyboard arrows</td>
</tr>
<tr>
<td>click on prompt</td>
<td>left-click once on a previous ModelSim or VSIM prompt in the transcript to copy the command typed at that prompt to the active cursor</td>
</tr>
<tr>
<td>his or history</td>
<td>shows the last few commands (up to 50 are kept)</td>
</tr>
</tbody>
</table>

Mouse and keyboard shortcuts in Main and Source windows

The following mouse actions and special keystrokes can be used to edit commands in the entry region of the Main window. They can also be used in editing the file displayed in the Source window and all Notepad windows (enter the notepad command within ModelSim to open the Notepad editor).

<table>
<thead>
<tr>
<th>Mouse - UNIX</th>
<th>Mouse - Windows</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; left-button - click &gt;</td>
<td></td>
<td>move the insertion cursor</td>
</tr>
<tr>
<td>&lt; left-button - press &gt; + drag</td>
<td></td>
<td>select</td>
</tr>
<tr>
<td>&lt; shift - left-button - press &gt;</td>
<td></td>
<td>extend selection</td>
</tr>
<tr>
<td>&lt; left-button - double-click &gt;</td>
<td></td>
<td>select word</td>
</tr>
<tr>
<td>&lt; left-button - double-click &gt; + drag</td>
<td></td>
<td>select word + word</td>
</tr>
</tbody>
</table>
### Mouse - UNIX

< control - left-button - click >

move insertion cursor without changing the selection

< left-button - click > on previous ModelSim or VSIM prompt

copy and paste previous command string to current prompt

< middle-button - click >

none

< middle-button - press > + drag

none

scroll the window

### Keystrokes - UNIX

< left | right - arrow >

move cursor left | right one character

< control > < left | right - arrow >

move cursor left | right one word

< shift > < left | right | up | down - arrow >

extend selection of text

< control > < shift > < left | right - arrow >

extend selection of text by word

< up | down - arrow >

scroll through command history (in Source window, moves cursor one line up | down)

< control > < up | down >

moves cursor up | down one paragraph

< control > < home >

move cursor to the beginning of the text

< control > < end >

move cursor to the end of the text

< backspace >, < control-h >

< backspace >
delete character to the left

< delete >, < control-d >

< delete >
delete character to the right

none

cancel

< alt >

activate or inactivate menu bar mode

< alt > < F4 >

close active window

< control - a >, < home >

< home >
move cursor to the beginning of the line

< control - b >

move cursor left

< control - d >

delete character to the right

< control - e >, < end >

< end >
move cursor to the end of the line

< control - f >

move cursor right one character

< control - k >

delete to the end of line

< control - n >

move cursor one line down (Source window only under Windows)
Mouse and keyboard shortcuts in Main and Source windows

The Main window allows insertions or pastes only after the prompt; therefore, you don’t need to set the cursor when copying strings to the command line.

### Right mouse button

The right mouse button provides shortcut menus in the Main and Wave windows. In the Source window, the button gives you feedback on any HDL item under the cursor. See *Chapter 8 - Graphic Interface* for menu descriptions.

<table>
<thead>
<tr>
<th>Keystrokes - UNIX</th>
<th>Keystrokes - Windows</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt; control - o &gt;</code></td>
<td>none</td>
<td>insert a newline character in front of the cursor</td>
</tr>
<tr>
<td><code>&lt; control - p &gt;</code></td>
<td></td>
<td>move cursor one line up (Source window only under Windows)</td>
</tr>
<tr>
<td><code>&lt; control - s &gt;</code></td>
<td><code>&lt; control - f &gt;</code></td>
<td>find</td>
</tr>
<tr>
<td><code>&lt; F3 &gt;</code></td>
<td></td>
<td>find next</td>
</tr>
<tr>
<td><code>&lt; control - t &gt;</code></td>
<td></td>
<td>reverse the order of the two characters to the right of the cursor</td>
</tr>
<tr>
<td><code>&lt; control - u &gt;</code></td>
<td></td>
<td>delete line</td>
</tr>
<tr>
<td><code>&lt; control - v &gt;</code></td>
<td><code>PageDn</code></td>
<td>move cursor down one screen</td>
</tr>
<tr>
<td><code>&lt; control - w &gt;</code></td>
<td><code>&lt; control - x &gt;</code></td>
<td>cut the selection</td>
</tr>
<tr>
<td><code>&lt; control - x &gt;, &lt; control - s&gt;</code></td>
<td><code>&lt; control - s &gt;</code></td>
<td>save</td>
</tr>
<tr>
<td><code>&lt; control - y &gt;, F18</code></td>
<td><code>&lt; control - v &gt;</code></td>
<td>paste the selection</td>
</tr>
<tr>
<td>none</td>
<td><code>&lt; control - a &gt;</code></td>
<td>select the entire contents of the widget</td>
</tr>
<tr>
<td><code>&lt; control - \&gt;</code>, <code>&lt; control - /&gt;</code></td>
<td><code>&lt; control - Z &gt;</code></td>
<td>undoes previous edits in the Source window</td>
</tr>
<tr>
<td><code>&lt; meta - &quot;&lt;&quot; &gt;</code></td>
<td>none</td>
<td>move cursor to the beginning of the file</td>
</tr>
<tr>
<td><code>&lt; meta - &quot;&gt;&quot; &gt;</code></td>
<td>none</td>
<td>move cursor to the end of the file</td>
</tr>
<tr>
<td><code>&lt; meta - v &gt;</code></td>
<td><code>PageUp</code></td>
<td>move cursor up one screen</td>
</tr>
<tr>
<td><code>&lt; Meta - w&gt;</code></td>
<td><code>&lt; control - c &gt;</code></td>
<td>copy selection</td>
</tr>
<tr>
<td><code>&lt; F8 &gt;</code></td>
<td></td>
<td>search for the most recent command that matches the characters typed (Main window only)</td>
</tr>
<tr>
<td><code>&lt; F9&gt;</code></td>
<td></td>
<td>run simulation</td>
</tr>
<tr>
<td><code>&lt; F10 &gt;</code></td>
<td></td>
<td>continue simulation</td>
</tr>
<tr>
<td><code>&lt; F11 &gt;</code></td>
<td></td>
<td>single-step</td>
</tr>
<tr>
<td><code>&lt; F12&gt;</code></td>
<td></td>
<td>step-over</td>
</tr>
</tbody>
</table>

The Main window allows insertions or pastes only after the prompt; therefore, you don’t need to set the cursor when copying strings to the command line.
C - Using the FLEXlm License Manager

Appendix contents

Starting the license server daemon . . . . . . . . . . . . . . . . . . . . . . . UM-408
   Locating the license file . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-408
   Controlling the license file search . . . . . . . . . . . . . . . . . . . . . . . UM-408
   Manual start . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-409
   Automatic start at boot time . . . . . . . . . . . . . . . . . . . . . . . . . . UM-409
   What to do if another application uses FLEXlm . . . . . . . . . . . . . UM-409

Format of the license file . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-410

Format of the daemon options file . . . . . . . . . . . . . . . . . . . . . . . . UM-410

License administration tools . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-412
   lms tat . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-412
   lmdown . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-412
   lmremove . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-413
   lmreread . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-413

Administration tools for Windows . . . . . . . . . . . . . . . . . . . . . . . . UM-413

This appendix covers Model Technology’s application of FLEXlm for ModelSim licensing.
Globetrotter Software’s Flexible License Manager (FLEXlm) is a network floating
licensing package that allows the application to be licensed on a concurrent usage basis, as
well as on a per-computer basis.

**FLEXlm user’s manual**

The content of this appendix is limited to the use of FLEXlm with Model Technology’s
software. For more information, see the FLEXlm user’s manual that is available from
Globetrotter Software’s web site:

Starting the license server daemon

Locating the license file

When the license manager daemon is started, it must be able to find the license file. The default location is `/usr/local/flexlm/licenses/license.dat` for Unix or `c:\flexlm\license.dat` for Windows. You can change where the daemon looks for the license file using one of two methods:

- By starting the license manager using the `-c <pathname>` option.
- By setting the `LM_LICENSE_FILE` environment variable to the path of the file.

More information about installing ModelSim and using a license file is available in Model Technology’s `Start Here for ModelSim` guide, see "Where to find our documentation" (UM-19), or email us at license@model.com.

Controlling the license file search

By default, ModelSim checks for the existence of both Model Technology and Mentor Graphics generated licenses. When vsim is invoked it will first locate and use any available MTI licenses, then search for MGC licenses as needed. The following `vsim` command switches narrow the search to exclude or include specific licenses:

<table>
<thead>
<tr>
<th>license option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-lic_nomgc</td>
<td>excludes any MGC licenses from the search</td>
</tr>
<tr>
<td>-lic_nomti</td>
<td>excludes any MTI licenses from the search</td>
</tr>
<tr>
<td>-lic_noqueue</td>
<td>do not wait in license queue if no licenses are available</td>
</tr>
<tr>
<td>-lic_plus</td>
<td>searches only for PLUS licenses</td>
</tr>
<tr>
<td>-lic_vlog</td>
<td>searches only for VLOG licenses</td>
</tr>
<tr>
<td>-lic_vhdl</td>
<td>searches only for VHDL licenses</td>
</tr>
<tr>
<td>-lic_viewsim</td>
<td>accepts a simulator license rather than being queued for a viewer license</td>
</tr>
</tbody>
</table>

The options may also be specified with the `License` (UM-390) variable in the `modelsim.ini` file; see the [vsim] simulator control variables (UM-388). Note that settings made from the command line are additive to options set in the License variable. For example, if you set the License variable to nomgc and use the -lic_plus option from the command line, vsim will search only for MTI SE/PLUS licenses.
Starting the license server daemon

Manual start

Unix
To start the license manager daemon, place the license file in the
/</install_dir>/modeltech/<platform> directory and enter the following commands:

```
  cd /<install_dir>/modeltech/<platform>
  lmgrd -c license.dat >& report.log
```

where <platform> can be sunos5, sunos5v9, hp700, hppa64, rs6000, rs64, or linux.
This can be done by an ordinary user; you should not be logged in as root.

Windows
To start the license manager daemon in Windows, place the license file in the modeltech
installation directory and enter the following commands:

```
  cd \<install_dir>\modeltech\win32
  lmgrd -app -c license.dat
```

Automatic start at boot time

Unix
You can cause the license manager daemon to start automatically at boot time by adding
the following line to the file /etc/rc.boot or to /etc/rc.local:

```
  /<install_dir>/modeltech/<platform>/lmgrd -c /<install_dir>/license.dat &
```

Windows
You can use the FLEXlm Control Panel to enact an automatic start. See the FLEXlm End
User’s Manual for more information.

What to do if another application uses FLEXlm

Case 1: All applications use the same license server nodes
You can combine the license files by taking the set of SERVER lines from one license file,
and adding the DAEMON, FEATURE, and FEATURESET lines from all of the license
files. This combined file can be copied to /<install_dir>/license/license.dat and to any
location required by the other applications.

Case 2: The applications use different license server nodes
You cannot combine the license files if the applications use different servers. Instead, set
the LM_LICENSE_FILE (UM-383) environment variable to be a list of files, as follows:

```
  setenv LM_LICENSE_FILE \
  lic_file1:lic_file2:/<install_dir>/license.dat
```

In Windows use semi-colons (;) to separate the file names.
Do not use the -c option when you start the license manager daemon. For example:

```
  lmgrd > report.log
```
Format of the license file

ModelSim license files contain three types of lines: SERVER lines, DAEMON lines, and FEATURE lines. For example:

```plaintext
SERVER hostname hostid [TCP_portnumber]
DAEMON daemon-name path-to-daemon [path-to-options-file]
FEATURE name daemon-name version exp_date #users_code \ 
"description" [hostid]
```

Only the following items may be modified:

- the hostname on SERVER lines
- the TCP_portnumber on SERVER lines
- the path-to-daemon on DAEMON lines
- the path-to-options-file on DAEMON lines
- anything in the daemon options file (described in the following section)

Format of the daemon options file

You can customize your ModelSim licensing with the daemon options file. This options file allows you to reserve licenses for specified users or groups of users, to determine which users have access to ModelSim software, to set software time-outs, and to log activity to an optional report writer.

RESERVE

Ensures that ModelSim will always be available to one or more users on one or more host computers.

INCLUDE

Allows you to specify a list of users who are allowed access to the ModelSim software.

EXCLUDE

Allows you to disallow access to ModelSim for certain users.

GROUP

Allows you to define a group of users for use in the other commands.

NOLOG

Causes messages of the specified type to be filtered out of the daemon’s log output.

To use the daemon options capability, you must create a daemon options file and list its pathname as the fourth field on the line that begins with DAEMON modeltech.

A daemon options file consists of lines in the following format:

```plaintext
RESERVE number feature {USER | HOST | DISPLAY | GROUP} name
INCLUDE feature {USER | HOST | DISPLAY | GROUP} name
EXCLUDE feature {USER | HOST | DISPLAY | GROUP} name
GROUP name <list_of_users>
NOLOG {IN | OUT | DENIED | QUEUED}
REPORTLOG file
```
Lines beginning with the number character (#) are treated as comments. If the filename in the REPORTLOG line starts with a plus (+) character, the old report logfile will be opened for appending.

For example, the following options file would reserve one copy of the feature `vsim` for the user walter, three copies for the user john, one copy for anyone on a computer with the hostname of bob, and would cause QUEUED messages to be omitted from the logfile. The user rita would not be allowed to use the vsim feature.

```plaintext
RESERVE 1 vsim USER walter
RESERVE 3 vsim USER john
RESERVE 1 vsim HOST bob
EXCLUDE vsim USER rita
NOLOG QUEUED
```

If this data were in the file named:

```plaintext
/usr/local/options
```

modify the license file DAEMON line as follows:

```plaintext
DAEMON modeltech /<install_dir>/<platform>/modeltech \
/usr/local/options
```
License administration tools

**Imstat**

License administration is simplified by the **Imstat** utility. **Imstat** allows a user of FLEXlm to instantly monitor the status of all network licensing activities. **Imstat** allows a system administrator at a user site to monitor license management operations, including:

- which daemons are running;
- which users are using individual features; and
- which users are using features served by a specific DAEMON.

The case-sensitive syntax is shown below:

**Syntax**

```
Imstat
   -a  -A
   -S <daemon>
   -c <license_file>
   -f <feature_name>
   -s <server_name>
   -t <value>
```

**Arguments**

- **-a**
  Displays everything.

- **-A**
  Lists all active licenses.

- **-S <daemon>**
  Lists all users of the specified daemon’s features.

- **-c <license_file>**
  Specifies that the specified license file is to be used.

- **-f <feature_name>**
  Lists users of the specified feature(s).

- **-s <server_name>**
  Displays the status of the specified server node(s).

- **-t <value>**
  Sets the Imstat time-out to the specified value.

**Imdown**

The **Imdown** utility allows for the graceful shutdown of all license daemons (both **lmgrd** and all vendor daemons) on all nodes.

**Syntax**

```
Imdown
   -c [<license_file_path>]
```
If not supplied here, the license file used is in either /user/local/flexlm/licenses/license.dat, or the license file pathname in the environment variable LM_LICENSE_FILE (UM-383).

The system administrator should protect the execution of **lmdown**, since shutting down the servers will cause loss of licenses.

**lmremove**

The **lmremove** utility allows the system administrator to remove a single user’s license for a specified feature. This could be required in the case where the licensed user was running the software on a node that subsequently crashed. This situation will sometimes cause the license to remain unusable. **lmremove** will allow the license to return to the pool of available licenses.

**Syntax**

```bash
lmremove
-c <file> <feature> <user> <host> <display>
```

**lmremove** removes all instances of **user** on the node **host** (on the display, if specified) from usage of **feature**. If the optional **-c <file>** switch is specified, the indicated file will be used as the license file. The system administrator should protect the execution of **lmremove**, since removing a user’s license can be disruptive.

**Imreread**

The **lmreread** utility causes the license daemon to reread the license file and start any new vendor daemons that have been added. In addition, all preexisting daemons will be signaled to reread the license file for changes in feature licensing information.

**Syntax**

```bash
lmreread [daemon]
-c <license_file>
```

**Note:** If the **-c** option is used, the license file specified will be read by the daemon, not by **lmgrd**. **lmgrd** rereads the file it read originally. Also, **lmreread** cannot be used to change server node names or port numbers. Vendor daemons will not reread their option files as a result of **lmreread**.

**Administration tools for Windows**

All of the Unix administration tools listed above may be used on Windows platforms as well. However, in Windows, all of the tools are launched via the program "lmutil." For example, if you want to run **lmstat**, you would type the following at a command prompt:

```bash
lmutil lmstat [-args]
```

The arguments for Windows are the same as those listed above for Unix.
D - Tips and Techniques

Appendix contents

How to use checkpoint/restore . . . . . . . . . . . . . . . . . UM-416
Running command-line and batch-mode simulations . . . . . . . UM-418
Source code security and -nodebug . . . . . . . . . . . . . . . UM-418
Saving and viewing waveforms . . . . . . . . . . . . . . . . . . UM-421
Setting up libraries for group use . . . . . . . . . . . . . . . UM-421
Maintaining 32-bit and 64-bit modules in the same library . . . UM-421
Bus contention checking . . . . . . . . . . . . . . . . . . . . . . UM-422
Bus float checking . . . . . . . . . . . . . . . . . . . . . . . . . UM-422
Design stability checking . . . . . . . . . . . . . . . . . . . . . UM-423
Toggle checking . . . . . . . . . . . . . . . . . . . . . . . . . . . UM-423
Detecting infinite zero-delay loops . . . . . . . . . . . . . . . UM-424
Referencing source files with location maps . . . . . . . . . . UM-425
Improve performance by locking memory on HP-UX 10.2 . . . . UM-427
Improve performance of large simulations on Sun/Solaris . . . UM-428
Performance affected by scheduled events being cancelled . . . UM-429
Modeling memory in VHDL . . . . . . . . . . . . . . . . . . . . UM-430
Setting up a List trigger with Expression Builder . . . . . . . UM-434

This appendix contains various tips and techniques collected from several parts of the manual and from answers to questions received by tech support. Your suggestions, tips, and techniques for this section would be appreciated.
How to use checkpoint/restore

The **checkpoint** (CR-83) and **restore** (CR-195) commands will save and restore the simulator state within the same invocation of vsim or between vsim sessions.

If you want to **restore** while running vsim, use the **restore** command (CR-195); we call this a "warm restore". If you want to start up vsim and restore a previously-saved checkpoint, use the **-restore** switch with the vsim command (CR-284); we call this a "cold restore".

**Note:** Checkpoint/restore allows a cold restore, followed by simulation activity, followed by a warm restore back to the original cold-restore checkpoint file. Warm restores to checkpoint files that were not created in the current run are not allowed except for this special case of an original cold restore file.

The things that are saved with **checkpoint** and restored with the **restore** command are:

- simulation kernel state
- vsim.wlf file
- signals listed in the list and wave windows
- file pointer positions for files opened under VHDL
- file pointer positions for files opened by the Verilog $fopen system task
- state of foreign architectures

Things that are NOT restored are:

- state of macros
- changes made with the command-line interface (such as user-defined Tcl commands)
- state of graphical user interface windows
- toggle statistics

In order to save the simulator state, use the command

```plaintext
checkpoint <filename>
```

To restore the simulator state during the same session as when the state was saved, use the command:

```plaintext
restore <filename>
```

To restore the state after quitting ModelSim, invoke vsim as follows:

```plaintext
vsim -restore <filename> [-nocompress]
```

The checkpoint file is normally compressed. If there is a need to turn off the compression, you can do so by setting a special Tcl variable. Use:

```plaintext
set CheckpointCompressMode 0
```

to turn compression off, and turn compression back on with:

```plaintext
set CheckpointCompressMode 1
```
You can also control checkpoint compression using the `modelsim.ini` file in the `[vsim]` section (use the same 0 or 1 switch):

```
[vsim]
  CheckpointCompressMode = <switch>
```

If you use the foreign interface, you will need to add additional function calls in order to use `checkpoint/restore`. See the FLI Reference Manual for more information.

**The difference between checkpoint/restore and restarting**

The `restart` (CR-193) command resets the simulator to time zero, clears out any logged waveforms, and closes any files opened under VHDL and the Verilog `$fopen` system task. You can get the same effect by first doing a checkpoint at time zero and later doing a `restore`. But with `restart` you don’t have to save the checkpoint and the `restart` is likely to be faster. But when you need to set the state to anything other than time zero, you will need to use `checkpoint/restore`.

**Using macros with restart and checkpoint/restore**

The `restart` (CR-193) command resets and restarts the simulation kernel, and zeros out any user-defined commands, but it does not touch the state of the macro interpreter. This lets you do `restart` commands within macros.

The pause mode indicates that a macro has been interrupted. That condition will not be affected by a restart, and if the restart is done with an interrupted macro, the macro will still be interrupted after the restart.

The situation is similar for using `checkpoint/restore` without quitting ModelSim, that is, doing a `checkpoint` (CR-83) and later in the same session doing a `restore` (CR-195) of the earlier checkpoint. The `restore` does not touch the state of the macro interpreter so you may also do `checkpoint` and `restore` commands within macros.
Running command-line and batch-mode simulations

The typical method of running ModelSim is interactive: you push buttons and/or pull down menus in a series of windows in the GUI (graphic user interface). But there are really three specific modes of ModelSim operation: GUI, command line, and batch. Here are their characteristics:

- **GUI mode**
  This is the usual interactive mode; it has graphical windows, push-buttons, menus, and a command line in the text window. This is the default mode.

- **Command-line mode**
  This an operational mode that has only an interactive command line; no interactive windows are opened. To run vsim in this manner, invoke it with the -c option as the first argument from either the UNIX prompt or the DOS prompt in Windows 95/98/2000/NT.

- **Batch mode**
  Batch mode is an operational mode that provides neither an interactive command line, nor interactive windows.

  In a UNIX environment, vsim can be invoked in batch mode by redirecting standard input using the “here-document” technique. Batch mode does not require the -c option. In a Windows environment, vsim is run from a Windows 95/98/2000/NT DOS prompt and standard input and output are redirected to and from files. An example is:

  ```
  vsim ent arch <<!
  log -r *
  run 100
  do test.do
  quit -f
  !
  ```

  Here is another example of batch mode, this time using a file as input:

  ```
  vsim counter < yourfile
  ```

  From a user’s point of view, command-line mode can look like batch mode if you use the vsim command (CR-284) with the -do option to execute a macro that does a quit -f (CR-188) before returning, or if the startup.do macro does a quit -f before returning. But technically, that mode of operation is still command-line mode because stdin is still operating from the terminal.

  The following paragraphs describe the behavior defined for the batch and command-line modes.

**Command-line mode**

In command-line mode ModelSim executes any startup command specified by the Startup (UM-390) variable in the modelsim.ini file. If vsim (CR-284) is invoked with the -do <"command_string"> option a DO file (macro) is called. A DO file executed in this manner will override any startup command in the modelsim.ini file.
During simulation a transcript file is created containing any messages to stdout. A transcript file created in command-line mode may be used as a DO file if you invoke the transcript on command (CR-217) after the design loads (see the example below). The transcript on command will write all of the commands you invoke to the transcript file. For example, the following series of commands will result in a transcript file that can be used for command input if top is resimulated (remove the quit -f command from the transcript file if you want to remain in the simulator).

```vsim -c top
```

*library and design loading messages... then execute:*

```transcript on
force clk 1 50, 0 100 -repeat 100
run 500
run @5000
quit -f```

Rename transcript files that you intend to use as DO files. They will be overwritten the next time you run vsim if you don’t rename them. Also, simulator messages are already commented out, but any messages generated from your design (and subsequently written to the transcript file) will cause the simulator to pause. A transcript file that contains only valid simulator commands will work fine; comment out anything else with a “#”.

Stand-alone tools will pick-up project settings in command-line mode if they are invoked in the project’s root directory. If invoked outside the project directory, stand-alone tools will pick up project settings only if you set the MODELSIM environment variable to the path to the project file (`<Project_Root_Dir>/<Project_Name>.mpf`).

**Batch mode**

In batch mode ModelSim behaves much as in command-line mode except that there are no prompts, and commands from re-directed stdin are not echoed to stdout. Do not use the -c argument with vsim for batch mode simulations because -c invokes the command-line mode, which supplies the prompts and echoes the commands.

Tcl user_hook_variables may also be used for Tcl customization during batch-mode simulation; see [http://www.model.com/resources/pref_variables/frameset.htm](http://www.model.com/resources/pref_variables/frameset.htm).
Source code security and -nodebug

The -nodebug option on both vcom (CR-240) and vlog (CR-274) hides internal model data. This allows a model supplier to provide pre-compiled libraries without providing source code and without revealing internal model variables and structure.

Note: ModelSim’s -nodebug compiler option provides protection for proprietary model information. The Verilog protect compiler directive provides similar protection, but uses a Cadence encryption algorithm that is unavailable to Model Technology.

If a design unit is compiled with -nodebug the Source window will not display the design unit’s source code, the Structure window will not display the internal structure, the Signals window will not display internal signals (it still displays ports), the Process window will not display internal processes, and the Variables window will not display internal variables. In addition, none of the hidden objects may be accessed through the Dataflow window or with ModelSim commands.

Even with the data hiding of -nodebug, there remains some visibility into models compiled with -nodebug. The names of all design units comprising your model are visible in the library, and you may invoke vsim (CR-284) directly on any of these design units and see the ports. Design units or modules compiled with -nodebug can only instantiate design units or modules that are also compiled -nodebug.

To restrict visibility into the lower levels of your design you can use the following -nodebug switches when you compile.

<table>
<thead>
<tr>
<th>Command and switch</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>vcom -nodebug=ports</td>
<td>makes the ports of a VHDL design unit invisible</td>
</tr>
<tr>
<td>vlog -nodebug=ports</td>
<td>makes the ports of a Verilog design unit invisible</td>
</tr>
<tr>
<td>vlog -nodebug=pli</td>
<td>prevents the use of PLI functions to interrogate the module for information</td>
</tr>
<tr>
<td>vlog -nodebug=ports+pli</td>
<td>combines the functions of -nodebug=ports and -nodebug=pli</td>
</tr>
<tr>
<td>(or =pli+ports)</td>
<td></td>
</tr>
</tbody>
</table>

Note: Don’t use the =ports switch on a design without hierarchy, or on the top level of a hierarchical design; if you do, no ports will be visible for simulation. To properly use the switch, compile all lower portions of the design with -nodebug=ports first, then compile the top level with -nodebug alone.

Also note the =pli switch will not work with vcom (the VHDL compiler). PLI functions are valid only for Verilog design units.
Saving and viewing waveforms

You can run `vsim` as a batch job, but view the resulting waveforms later.

1. When you invoke `vsim` the first time, use the `-wlf` option to rename the wave log format (WLF) file, and redirect stdin to invoke the batch mode. The command should look like this:

   ```
   vsim -wlf wavesav1.wlf counter < command.do
   ```

   Within your `command.do` file, use the `log` command (CR-154) to save the waveforms you want to look at later, run the simulation, and quit.

   When `vsim` runs in batch mode, it does not write to the screen, and can be run in the background.

2. When you return to work the next day after running several batch jobs, you can start up `vsim` in its viewing mode with this command and the appropriate `.wlf` files:

   ```
   vsim -view wavesav1.wlf
   ```

   Now you will be able to use the Waveform and List windows normally.

Setting up libraries for group use

By adding an “others” clause to your `modelsim.ini` file, you can have a hierarchy of library mappings. If the ModelSim tools don’t find a mapping in the `modelsim.ini` file, then they will search the library section of the initialization file specified by the “others” clause. For example:

```ini
[library]
asic_lib = /cae/asic_lib
work = my_work
others = /usr/modeltech/modelsim.ini
```

Maintaining 32-bit and 64-bit modules in the same library

It is possible with ModelSim to maintain 64-bit and 32-bit versions of a design in the same library. To do this, you must compile the design with one of the versions (64-bit or 32-bit), and "refresh" the design with the other version. For example:

Using the 32-bit version of ModelSim:

```
vcom file1.vhd
vcom file2.vhd
``` 

Next, using the 64-bit version of ModelSim:

```
vcom -refresh
``` 

Do not compile the design with one version, and then recompile it with the other. If you do this, ModelSim will remove the first module, because it could be "stale."
Bus contention checking

Bus contention checking detects bus fights on nodes that have multiple drivers. A bus fight occurs when two or more drivers drive a node with the same strength and that strength is the strongest of all drivers currently driving the node. The following table provides some examples for two drivers driving a std_logic signal:

<table>
<thead>
<tr>
<th>driver 1</th>
<th>driver 2</th>
<th>fight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>Z</td>
<td>no</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td>Z</td>
<td>no</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>yes</td>
</tr>
<tr>
<td>L</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>yes</td>
</tr>
</tbody>
</table>

Detection of a bus fight results in an error message specifying the node and its drivers’ current driving values. If a node’s drivers later change value and the node is still in contention, a message is issued giving the new values of the drivers. A message is also issued when the contention ends. The bus contention checking commands can be used on VHDL and Verilog designs.

These bus checking commands are in "ModelSim Commands" (CR-29):

- `check contention add` (CR-75)
- `check contention config` (CR-76)
- `check contention off` (CR-77)

Bus float checking

Bus float checking detects nodes that are in the high impedance state for a time equal to or exceeding a user-defined limit. This is an error in some technologies. Detection of a float violation results in an error message identifying the node. A message is also issued when the float violation ends. The bus float checking commands can be used on VHDL and Verilog designs.

These bus float checking commands are in "ModelSim Commands" (CR-29):

- `check float add` (CR-78)
- `check float config` (CR-79)
- `check float off` (CR-80)
Design stability checking

Design stability checking detects when circuit activity has not settled within a period you define for synchronous designs. You specify the clock period for the design and the strobe time within the period during which the circuit must be stable. A violation is detected and an error message is issued if there are pending driver events at the strobe time. The message identifies the driver that has a pending event, the node that it drives, and the cycle number. The design stability checking commands can be used on VHDL and Verilog designs.

These design stability checking commands are in "ModelSim Commands" (CR-29):

- `check stable on` (CR-82)
- `check stable off` (CR-81)

Toggle checking

Toggle checking counts the number of transitions to 0 and 1 on specified nodes. Once the nodes have been selected, a toggle report may be requested at any time during the simulation. The toggle commands can be used on VHDL and Verilog designs.

These toggle checking commands are in "ModelSim Commands" (CR-29):

- `toggle add` (CR-213)
- `toggle reset` (CR-215)
- `toggle report` (CR-214)
Detecting infinite zero-delay loops

Simulations use steps that advance simulated time, and steps that do not advance simulated time. Steps that do not advance simulated time are called "delta cycles". Delta cycles are used when signal assignments are made with zero time delay.

If a large number of delta cycles occur without advancing time, it is usually a symptom of an infinite zero-delay loop in the design. In order to detect the presence of these loops, ModelSim defines a limit, the “iteration_limit”, on the number of successive delta cycles that can occur. When the iteration_limit is exceeded, vsim stops the simulation and gives a warning message.

You can set the iteration_limit from the Options > Simulation menu, by modifying the modelsim.ini file, or by setting a Tcl variable called IterationLimit (UM-389).

The iteration_limit default value is 5000.

When you get an iteration_limit warning, first increase the iteration limit and try to continue simulation. If the problem persists, look for zero-delay loops.

One approach to finding zero-delay loops is to increase the iteration limit again and start single stepping. You should be able to see the assignment statements or processes that are looping. Looking at the Process window will also help you to see the active looping processes.

When the loop is found, you will need to change the design to eliminate the unstable loop.

See "Projects and system initialization" (UM-21) for more information on modifying the modelsim.ini file. And see "Preference variables located in TCL files" (UM-396) for more information on Tcl variables. Also see the Main window Help menu for Tcl Help and man pages.
Referencing source files with location maps

Pathnames to source files are recorded in libraries by storing the working directory from which the compile is invoked and the pathname to the file as specified in the invocation of the compiler. The pathname may be either a complete pathname or a relative pathname.

ModelSim tools that reference source files from the library locate a source file as follows:
• If the pathname stored in the library is complete, then this is the path used to reference the file.
• If the pathname is relative, then the tool looks for the file relative to the current working directory. If this file does not exist, then the path relative to the working directory stored in the library is used.

This method of referencing source files generally works fine if the libraries are created and used on a single system. However, when multiple systems access a library across a network the physical pathnames are not always the same and the source file reference rules do not always work.

Using location mapping

Location maps are used to replace prefixes of physical pathnames in the library with environment variables. The location map defines a mapping between physical pathname prefixes and environment variables.

ModelSim tools open the location map file on invocation if the MGC_LOCATION_MAP (UM-383) environment variable is set. If MGC_LOCATION_MAP is not set, ModelSim will look for a file named "mgc_location_map" in the following locations, in order:
• the current directory
• your home directory
• the directory containing the ModelSim binaries
• the ModelSim installation directory

Use these two steps to map your files:

1 Set the environment variable MGC_LOCATION_MAP to the path to your location map file.

2 Specify the mappings from physical pathnames to logical pathnames:

```bash
$SRC
/home/vhd1/src
/usr/vhd1/src

$IEEE
/usr/modeltech/ieee
```
Pathname syntax

The logical pathnames must begin with $ and the physical pathnames must begin with /.
The logical pathname is followed by one or more equivalent physical pathnames. Physical
pathnames are equivalent if they refer to the same physical directory (they just have
different pathnames on different systems).

How location mapping works

When a pathname is stored, an attempt is made to map the physical pathname to a path
relative to a logical pathname. This is done by searching the location map file for the first
physical pathname that is a prefix to the pathname in question. The logical pathname is then
substituted for the prefix. For example, "/usr/vhdl/src/test.vhd" is mapped to "$SRC/
test.vhd". If a mapping can be made to a logical pathname, then this is the pathname that is
saved. The path to a source file entry for a design unit in a library is a good example of a
typical mapping.

For mapping from a logical pathname back to the physical pathname, ModelSim expects an
environment variable to be set for each logical pathname (with the same name). ModelSim
reads the location map file when a tool is invoked. If the environment variables
corresponding to logical pathnames have not been set in your shell, ModelSim sets the
variables to the first physical pathname following the logical pathname in the location map.
For example, if you don’t set the SRC environment variable, ModelSim will automatically
set it to "/home/vhdl/src".

Mapping with Tcl variables

Two Tcl variables may also be used to specify alternative source-file paths; SourceDir and
Improve performance by locking memory on HP-UX 10.2

ModelSim 5.3 and later versions contain a feature to allow HP-UX 10.2 to use locked memory. This feature provides significant acceleration of simulation time for large designs – i.e. with a memory footprint > 500Mb. (Test cases showed 2x acceleration of large simulations.) The following steps show how to set up the HP-UX 10.2 so memory can be locked.

1  Allow the average-user to lock memory. By default, this privilege is not allowed, so it has to be enabled. To allow everyone MLOCK privileges, the administrator needs to execute this command on the machine that will be running ModelSim:

   /usr/sbin/setprivgrp -g MLOCK

   To only allow a particular group MLOCK privileges, use the command:

   /usr/sbin/setprivgrp <group-name> MLOCK

   This allows you to lock memory. No other privileges are enabled.

2  Once the MLOCK privilege is enabled, you merely have to modify the modelsim.ini file, and add the following entry to the [vsim] section:

   LockedMemory = <some-value>

   Where <some-value> is an integer representing the number of megabytes of memory to be locked. Once this is done, the memory will be locked when vsim invokes (using this .ini file).

   ModelSim will not lock more memory than is available in the system. The maximum memory that can be locked is: system physical memory (RAM) - 100 Mb = locked memory

   When ModelSim locks memory, other processes will not have access to it. Therefore, you should consider how much memory is locked on a per-design basis to avoid locking more than is needed.

   System parameters used for shared/locked memory may not be set (by default) high enough to take full advantage of this feature in later generations of HP-UX. Using the "sam" program, go to the "Configurable Parameters" window (under "Kernel Configuration"). There are several values that may need to be increased.

   First, enable shared memory. The value for "shmeng" should be equal to 1. Set the value for "shmmmax" as large as possible. The defaults for the values of "shmmnin" and "shmseg" should be ok. To change these parameters, you have to rebuild the kernel and reboot.
Improve performance of large simulations on Sun/Solaris

Starting with the ModelSim 5.5b you can improve simulation performance on Sun/Solaris by enabling shared memory. Up to a 2x improvement has been seen in large Verilog gate-level simulations, though the feature should speed up any simulation that consumes a large amount of memory.

Follow these steps to use the feature:

1. Enable a large shared memory segment by adding the following line to the /etc/system file:

   ```bash
   set shmsys:shminfo_shmmax=0xffffffff
   ```

2. Reboot your machine.

3. Immediately after the machine has been rebooted, run the program `vshminit` (found in the modeltech tree). The program takes a single parameter which is the amount of memory in megabytes to reserve for use by the simulator. For example, running `vshminit` like this:

   ```bash
   <modeltech-tree>/sunos5/vshminit 700
   ```

   would reserve 700mb of space for use by the simulator. The next time you run the simulator it will automatically detect the reserved memory and use it.

   **Important**: `vshminit` must be run immediately after you reboot the machine. (You might want to add the program to the system startup scripts.) There may be no performance benefit if you don’t run it immediately after reboot.

The amount of memory you supply as a parameter to `vshminit` depends on the configuration of your system. Typically you might want to reserve 50-80% of the system memory for the simulator, depending on whether the machine is multi-use or is dedicated to running simulations.

To free the memory reserved by `vshminit`, execute the following command:

```bash
/bin/ipcrm -M 0x10761364
```
Performance affected by scheduled events being cancelled

Performance will suffer if events are scheduled far into the future but then cancelled before they take effect. This situation will act like a memory leak and slow down simulation.

In VHDL this situation can occur several ways. The most common are waits with time-out clauses and projected wave forms in signal assignments.

The following code shows a wait with a time-out:

```vhdl
signals synch : bit := '0';
...
p: process
begin
    wait for 10 ms until synch = 1;
    end process;

    synch <= not synch after 10 ns;
```

At time 0 p makes an event for time 10ms. When synch goes to 1 at 10 ns, the event at 10 ms is marked as cancelled but not deleted, and a new event is scheduled at 10ms + 10ns. The cancelled events are not reclaimed until time 10ms is reached and the cancelled event is processed. As a result there will be 500000 (10ms/20ns) cancelled but undeleted events. Once 10ms is reached memory will no longer increase because we will be reclaiming events as fast as we add them.

For projected wave forms the following would behave the same way:

```vhdl
signals synch : bit := '0';
...
p: process(synch)
begin
    output <= '0', '1' after 10ms;
    end process;

    synch <= not synch after 10 ns;
```
Modeling memory in VHDL

As a VHDL user, you might be tempted to model a memory using signals. Two common simulator problems are the likely result:

- You may get a "memory allocation error" message, which typically means the simulator ran out of memory and failed to allocate more storage.
- Or, you may get very long load, elaboration or run times.

These problems are usually explained by the fact that signals consume a substantial amount of memory (many dozens of bytes per bit), all of which needs to be loaded or initialized before your simulation starts.

A simple alternative implementation provides some excellent performance benefits:

- storage required to model the memory can be reduced by 1-2 orders of magnitude
- startup and run times are reduced
- associated memory allocation errors are eliminated

The trick is to model memory using variables instead of signals.

In the example below, we illustrate three alternative architectures for entity "memory". Architecture "style_87_bad" uses a vhdl signal to store the ram data. Architecture "style_87" uses variables in the "memory" process, and architecture "style_93" uses variables in the architecture.

For large memories, architecture "style_87_bad" runs many times longer than the other two, and uses much more memory. This style should be avoided.

Both architectures "style_87" and "style_93" work with equal efficiency. You’ll find some additional flexibility with the VHDL 1993 style, however, because the ram storage can be shared between multiple processes. For example, a second process is shown that initializes the memory; you could add other processes to create a multi-ported memory.

To implement this model, you will need functions that convert vectors to integers. To use it you will probably need to convert integers to vectors.

Example functions are provided below in package "conversions".

```vhdl
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;
use work.conversions.all;

entity memory is
  generic(add_bits : integer := 12;
          data_bits : integer := 32);
  port(add_in : in std_ulogic_vector(add_bits-1 downto 0);
       data_in : in std_ulogic_vector(data_bits-1 downto 0);
       data_out : out std_ulogic_vector(data_bits-1 downto 0);
       cs, mwrite : in std_ulogic;
       do_init : in std_ulogic);
  subtype word is std_ulogic_vector(data_bits-1 downto 0);
  constant nwords : integer := 2 ** add_bits;
  type ram_type is array(0 to nwords-1) of word;
end;

architecture style_93 of memory is
  ```
shared variable ram : ram_type;
----------------------------------
begin
memory:
process (cs)
variable address : natural;
begin
  if rising_edge(cs) then
    address := sulp_to_natural(add_in);
    if (mwrite = '1') then
      ram(address) := data_in;
      data_out <= ram(address);
    else
      data_out <= ram(address);
    end if;
  end if;
end process memory;
-- illustrates a second process using the shared variable
initialize:
process (do_init)
variable address : natural;
begin
  if rising_edge(do_init) then
    for address in 0 to nwords-1 loop
      ram(address) := data_in;
    end loop;
  end if;
end process initialize;
end architecture style_93;

architecture style_87 of memory is
begin
memory:
process (cs)
variable ram : ram_type;
----------------------------------
variable address : natural;
begin
  if rising_edge(cs) then
    address := sulp_to_natural(add_in);
    if (mwrite = '1') then
      ram(address) := data_in;
      data_out <= ram(address);
    else
      data_out <= ram(address);
    end if;
  end if;
end process;
end style_87;

architecture bad_style_87 of memory is
----------------------------------
signal ram : ram_type;
----------------------------------
begin
memory:
process (cs)
variable address : natural := 0;
begin
  if rising_edge(cs) then

```
address := sulv_to_natural(add_in);
if (mwrite = '1') then
  ram(address) <= data_in;
  data_out <= data_in;
else
  data_out <= ram(address);
end if;
end if;
end process;
end bad_style_87;

------------------------------------------------------------
------------------------------------------------------------
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;

package conversions is
  function sulv_to_natural(x : std_ulogic_vector) return natural;
  function natural_to_sulv(n, bits : natural) return std_ulogic_vector;
end conversions;

package body conversions is
  function sulv_to_natural(x : std_ulogic_vector) return natural is
    variable n : natural := 0;
    variable failure : boolean := false;
    begin
      assert (x'high - x'low + 1) <= 31
      report "Range of sulv_to_natural argument exceeds natural range"
      severity error;
      for i in x'range loop
        n := n * 2;
        case x(i) is
          when '1' | 'H' => n := n + 1;
          when '0' | 'L' => null;
          when others => failure := true;
        end case;
      end loop;
      assert not failure
      report "sulv_to_natural cannot convert indefinite std_ulogic_vector"
      severity error;
      if failure then
        return 0;
      else
        return n;
      end if;
    end sulv_to_natural;

  function natural_to_sulv(n, bits : natural) return std_ulogic_vector is
    variable x : std_ulogic_vector(bits-1 downto 0) :=
      (others => '0');
    variable tempn : natural := n;
    begin
      ...
for i in x’reverse_range loop
  if (tempn mod 2) = 1 then
    x(i) := '1';
  end if;
  tempn := tempn / 2;
end loop;
return x;
end natural_to_sulv;

end conversions;
Setting up a List trigger with Expression Builder

This example shows you how to set a List window trigger based on a gating expression created with the ModelSim Expression Builder.

If you want to look at a set of signal values ONLY during the simulation cycles during which an enable signal rises, you would need to use the List window Trigger Gating feature. The gating feature suppresses all display lines except those for which a specified gating function evaluates to true.

Select Prop > Display Props (List window) to access the Triggers tab.

![Modify Display Properties (list)](image)

Check the Trigger Gating: Expression check box. Then click on Use Expression Builder. Select the signal in the List window that you want to be the enable signal by
clicking on its name in the header area of the List window. Then click **Insert Selected Signal** and 'rising' in the Expression Builder.

[Image of Expression Builder]

Click OK to close the Expression Builder. You should see the name of the signal plus "rising" added to the Expression entry box of the Modify Display Properties dialog box. (Leave the **On Duration** field zero for now.) Click the OK button.

If you already have simulation data in the List window, the display should immediately switch to showing only those cycles for which the gating signal is rising. If that isn't quite what you want, you can go back to the expression builder and play with it until you get it the way you want it.

If you want the enable signal to work like a "One-Shot" that would display all values for the next, say 10 ns, after the rising edge of enable, then set the **On Duration** value to **10 ns**. Otherwise, leave it at zero, and select **Apply** again. When everything is correct, click **OK** to close the Modify Display Properties dialog box.

When you save the List window configuration, the list gating parameters will be saved as well, and can be set up again by reading in that macro. You can take a look at the macro to see how the gating can be set up using macro commands.
ModelSim 5.5 includes many new features and enhancements that are described in the tables below. Links within the groups will connect you to more detail. GUI changes are described toward the end of the appendix.

## New features

<table>
<thead>
<tr>
<th>What</th>
<th>Description</th>
<th>Where (select a link)</th>
<th>ModelSim release</th>
</tr>
</thead>
<tbody>
<tr>
<td>shared memory on Sun/Solaris</td>
<td>access to shared memory speeds up large simulations</td>
<td>Improve performance of large simulations on Sun/Solaris (UM-428)</td>
<td>5.5b</td>
</tr>
<tr>
<td>cursor time field enhanced</td>
<td>you can type in a cursor time to scroll the Wave window</td>
<td>Finding a cursor (UM-234)</td>
<td>5.5b</td>
</tr>
<tr>
<td>waveform comparison</td>
<td>compare simulations and datasets</td>
<td>Chapter 11 - Waveform Comparison</td>
<td>5.5</td>
</tr>
<tr>
<td>ModelSim projects</td>
<td>projects have been completely revamped to ease getting started with ModelSim</td>
<td>Projects and system initialization (UM-21)</td>
<td>5.5</td>
</tr>
<tr>
<td>gate-level optimizations</td>
<td>gate-level Verilog designs can now be optimized using -fast</td>
<td>Compiling for faster performance (UM-83)</td>
<td>5.5</td>
</tr>
<tr>
<td>VCD file enhancements</td>
<td>support multiple VCD files and dumpports tasks</td>
<td>ModelSim VCD commands and VCD tasks (UM-334)</td>
<td>5.5</td>
</tr>
<tr>
<td>enhanced Code Coverage feature</td>
<td>new interface and ability to exclude files and lines</td>
<td>Chapter 10 - Code Coverage</td>
<td>5.5</td>
</tr>
<tr>
<td>vcd2wlf</td>
<td>new utility converts VCD files to WLF files</td>
<td>vcd2wlf (CR-239)</td>
<td>5.5</td>
</tr>
<tr>
<td>bookmarks</td>
<td>save zoom and scroll settings in Wave window</td>
<td>Saving zoom range and scroll position with bookmarks (UM-236)</td>
<td>5.5</td>
</tr>
</tbody>
</table>
## Command and variable changes

<table>
<thead>
<tr>
<th>What</th>
<th>Description</th>
<th>Where (select a link)</th>
<th>ModelSim release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workspace</td>
<td>new Main window eases working with design units and datasets</td>
<td><a href="UM-152">Workspace</a></td>
<td>5.5</td>
</tr>
<tr>
<td>find and replace in Source window</td>
<td>Source window now supports search and replace for text and regular expressions</td>
<td><a href="UM-202">Finding and replacing in the Source window</a></td>
<td>5.5</td>
</tr>
<tr>
<td>breakpoints dialog</td>
<td>manage breakpoints via dialog boxes</td>
<td><a href="UM-230">Setting signal breakpoints</a></td>
<td>5.5</td>
</tr>
<tr>
<td>import library wizard</td>
<td>imports FPGA libraries</td>
<td><a href="UM-49">Importing FPGA libraries</a></td>
<td>5.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>What</th>
<th>Description</th>
<th>Where (select a link)</th>
<th>ModelSim release</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrefWave(WaveformPopupEnabled) preference variable</td>
<td>toggles popup in waveform pane</td>
<td>Preference variable database</td>
<td>5.5d</td>
</tr>
<tr>
<td>+opt argument to vlog</td>
<td>optimizes designs that have been previously compile unoptimized</td>
<td>vlog (CR-274)</td>
<td>5.5c</td>
</tr>
<tr>
<td>.wave.tree interrupt</td>
<td>halts waveform drawing in the Wave window</td>
<td>.wave.tree interrupt (CR-40)</td>
<td>5.5c</td>
</tr>
<tr>
<td>-vcdstim argument to vsim</td>
<td>resimulates a design from a VCD file</td>
<td>-vcdstim (CR-289)</td>
<td>5.5c</td>
</tr>
<tr>
<td>PrefSource(OpenOnBreak) preference variable</td>
<td>disables automatic opening of Source window on break</td>
<td>Preference variable database</td>
<td>5.5c</td>
</tr>
<tr>
<td>+v2k_int_delays argument to vsim</td>
<td>causes interconnect delay to be visible at the load module port</td>
<td>vsim (CR-284)</td>
<td>5.5b</td>
</tr>
<tr>
<td>+nocheck arguments to vlog</td>
<td>increase optimizations of -fast</td>
<td>vlog (CR-274)</td>
<td>5.5b</td>
</tr>
<tr>
<td>mti_CreateTimeType() function</td>
<td>new function in FLI gets a handle to a type descriptor for a VHDL time type</td>
<td>ModelSim FLI Reference</td>
<td>5.5b</td>
</tr>
<tr>
<td>PrefMain(stallKernel) preference variable</td>
<td>pauses kernal while Wave window updates occur</td>
<td>Preference variable database</td>
<td>5.5b</td>
</tr>
<tr>
<td>bit- and part-selects supported in when command</td>
<td>the when command will evaluate bit- and part-select comparisons</td>
<td>Examples (CR-299)</td>
<td>5.5a</td>
</tr>
<tr>
<td>What</td>
<td>Description</td>
<td>Where (select a link)</td>
<td>ModelSim release</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>--------------------------------------------------</td>
<td>--------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>pedantic errors argument</td>
<td>new -pedanticerrors argument to vcom forces errors on two conditions</td>
<td>-pedanticerrors (CR-243)</td>
<td>5.5a</td>
</tr>
<tr>
<td>-compile_uselibs argument for vlog</td>
<td>eases use of `uselib directives</td>
<td>-compile_uselibs argument (UM-76)</td>
<td>5.5</td>
</tr>
<tr>
<td>-lint argument for vlog</td>
<td>enables lint-style checks</td>
<td>-lint (CR-276)</td>
<td>5.5</td>
</tr>
<tr>
<td>middle mouse button pasting control</td>
<td>enables/disables middle mouse button pasting</td>
<td>Middle Mouse Button Paste (UM-197)</td>
<td>5.5</td>
</tr>
<tr>
<td>init_signal_spy utility</td>
<td>reference signals, registers, or wires at any level of hierarchy</td>
<td>init_signal_spy() (UM-64) and $init_signal_spy (UM-97)</td>
<td>5.5</td>
</tr>
<tr>
<td>get_resolution function</td>
<td>returns the current simulator resolution as a real</td>
<td>get_resolution() (UM-63)</td>
<td>5.5</td>
</tr>
<tr>
<td>to_real function</td>
<td>converts the physical type time to the type real</td>
<td>to_real() (UM-65)</td>
<td>5.5</td>
</tr>
<tr>
<td>to_time function</td>
<td>converts the type real to the physical type time</td>
<td>to_time() (UM-66)</td>
<td>5.5</td>
</tr>
<tr>
<td>compare commands</td>
<td>several commands for doing waveform comparisons</td>
<td>Waveform Comparison commands (UM-317)</td>
<td>5.5</td>
</tr>
<tr>
<td>bookmark commands</td>
<td>several commands for saving/editing bookmarks</td>
<td>bookmark add wave (CR-65)</td>
<td>5.5</td>
</tr>
<tr>
<td>PrefCompare Tcl variables</td>
<td>Tcl preference variables for waveform comparisons</td>
<td>Preference variable database</td>
<td>5.5</td>
</tr>
<tr>
<td>-delay argument for virtual signal and virtual function</td>
<td>assign delay to signals within a virtual command</td>
<td>virtual function (CR-257) &amp; virtual signal (CR-269)</td>
<td>5.5</td>
</tr>
<tr>
<td>-keeploaded and -keeploadedrestart arguments for vsim</td>
<td>leaves FLI/PLI/VPI shared libraries loaded during a restart or design load</td>
<td>-keeploaded (CR-286) and -keeploadedrestart (CR-286)</td>
<td>5.5</td>
</tr>
<tr>
<td>vsim arguments related to WLF files</td>
<td>four arguments control WLF file creation</td>
<td>-wlf &lt;filename&gt; (CR-289), -wlfslim &lt;size&gt; (CR-290), -wlftlim &lt;duration&gt; (CR-290), and -wlfnocompress (CR-290)</td>
<td>5.5</td>
</tr>
<tr>
<td>delay in GUI_expression_format</td>
<td>assign delay to signals in a GUI_expression</td>
<td>Signal attributes (CR-25)</td>
<td>5.5</td>
</tr>
<tr>
<td>acc_fetch_paramval_str() function in PLI</td>
<td>allows fetching of a string on 64-bit platforms</td>
<td>64-bit support in the PLI (UM-118)</td>
<td>5.5</td>
</tr>
</tbody>
</table>
### What's new in ModelSim

<table>
<thead>
<tr>
<th>What</th>
<th>Description</th>
<th>Where (select a link)</th>
<th>ModelSim release</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLF file control variables</td>
<td>new vsim control variables configure WLF file creation</td>
<td>Setting default simulation options (UM-259)</td>
<td>5.5</td>
</tr>
</tbody>
</table>

### Documentation changes

<table>
<thead>
<tr>
<th>What</th>
<th>Description</th>
<th>Where (select a link)</th>
<th>ModelSim release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macro discussion moved to Tcl chapter</td>
<td>Macro topic in Tips and Techniques moved to Tcl chapter</td>
<td>&quot;Macros (DO files)&quot; (UM-377)</td>
<td>5.5d</td>
</tr>
<tr>
<td>Technical support chapter removed</td>
<td>Tech support information now accessed on the web</td>
<td><a href="http://www.model.com/support">www.model.com/support</a></td>
<td>5.5a</td>
</tr>
<tr>
<td>New Foreign Language Interface Reference manual</td>
<td>new manual provides detailed documentation of FLI including code examples</td>
<td>FLI Reference Manual</td>
<td>5.5</td>
</tr>
<tr>
<td>FLI chapter removed</td>
<td>replaced by FLI reference manual</td>
<td></td>
<td>5.5</td>
</tr>
<tr>
<td>new chapter on waveform comparison</td>
<td>describes new waveform comparison feature</td>
<td>Chapter 11 - Waveform Comparison</td>
<td>5.5</td>
</tr>
<tr>
<td>new tutorial on waveform comparison</td>
<td>practice using the new waveform comparison feature</td>
<td>ModelSim Tutorial</td>
<td>5.5</td>
</tr>
</tbody>
</table>
# GUI changes in version 5.5

This section identifies differences between the version 5.3/5.4 GUI and the 5.5 GUI.

<table>
<thead>
<tr>
<th>Main window changes</th>
<th>UM-442</th>
</tr>
</thead>
<tbody>
<tr>
<td>Menu bar and toolbar</td>
<td>UM-442</td>
</tr>
<tr>
<td>File menu</td>
<td>UM-443</td>
</tr>
<tr>
<td>Edit menu</td>
<td>UM-444</td>
</tr>
<tr>
<td>Design menu</td>
<td>UM-445</td>
</tr>
<tr>
<td>View menu</td>
<td>UM-446</td>
</tr>
<tr>
<td>Project menu</td>
<td>UM-446</td>
</tr>
<tr>
<td>Compare menu</td>
<td>UM-447</td>
</tr>
<tr>
<td>Options menu</td>
<td>UM-447</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signals window changes</th>
<th>UM-448</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source window changes</td>
<td>UM-449</td>
</tr>
<tr>
<td>Edit menu</td>
<td>UM-449</td>
</tr>
<tr>
<td>Options menu</td>
<td>UM-449</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Wave window changes</th>
<th>UM-450</th>
</tr>
</thead>
<tbody>
<tr>
<td>Menu bar and toolbar</td>
<td>UM-450</td>
</tr>
<tr>
<td>Edit menu</td>
<td>UM-450</td>
</tr>
<tr>
<td>Compare menu</td>
<td>UM-451</td>
</tr>
<tr>
<td>Bookmark menu</td>
<td>UM-451</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Coverage_summary window changes</th>
<th>UM-452</th>
</tr>
</thead>
</table>
Main window changes

The most obvious change in the version 5.5 Main window is the addition of the workspace. See "Workspace" (UM-152) for full details.

Menu bar and toolbar

The Main window toolbar in version 5.5 has not changed from version 5.3 / 5.4. The Main window menu bar has new Project and Compare menus. See the following pages for additional menu changes.
File menu

As shown below, the version 5.5 Main window File menu contains two additions. See "The Main window menu bar" (UM-154) for complete menu option details.
The graphic below shows the new menu command for importing (adding) a source file to a project. See "Step 2 — Add files to the project" (UM-27) for details.

**5.3 / 5.4**

- New Folder
- New Source
- Import Source
- New Project

**5.5**

- Compile Order...
- Compile All
- Add file to Project
- Sort by Alphabetical Order

**Edit menu**

See "The Main window menu bar" (UM-154) for complete menu option details.

**5.5**

- Copy
- Paste
- Select All
- Unselect All
- Find...
- Breakpoint(s)...

**new selection**

**new 5.5 context menu accessed via right mouse button on the Project page in the Workspace**

**new Project menu**
Design menu

See "The Main window menu bar" (UM-154) for complete menu option details.

### 5.3 / 5.4

- Browse Libraries...
- Create a New Library...
- View Library Contents...
- FPGA Library Manager...
- Compile...
- Compile Project
- Load Design...
- End Simulation...

### 5.5

- Browse Libraries...
- Create a New Library...
- Import Library...
- Compile...
- Load Design...
- End Simulation...

---

**5.5**

- Compile Order...
- Compile All
- Add file to Project
- Sort by Alphabetical Order

---

replaced in 5.5 by the Designs page in the Workspace

new 5.5 context menu accessed via right mouse button on the Project page
View menu

See "The Main window menu bar" (UM-154) for complete menu option details.

5.5

Project menu

The Project menu is new in version 5.5. See "What are projects?" (UM-22) for details.

5.5
Compare menu

The Compare menu is new in version 5.5. See *Chapter 11 - Waveform Comparison* for details on waveform comparisons. See also “The Main window menu bar” (UM-154) for complete menu option details.

**5.5**

Options menu

See "The Main window menu bar" (UM-154) for complete menu option details. See also “What are projects?” (UM-22) for details on Project operations.

**5.3 / 5.4**

no equivalent in 5.5; all Project editing is done from the Project page in the Workspace
Signals window changes

The menus accessed from the Signals menu bar are the same in version 5.5 as they were in version 5.3 / 5.4. However, the context menu (accessed with a right mouse click in the Signals window) has changed. See "Setting signal breakpoints" (UM-192) for complete details on this context menu.
Source window changes

Edit menu

See "The Source window menu bar" (UM-196) for complete menu option details.

5.5

Options menu

See "The Structure window menu bar" (UM-205) for complete menu option details.

5.5
Wave window changes

Menu bar and toolbar

The version 5.5 Wave window menu bar has two new menus, and the toolbar has four new icons. See “The Wave window menu bar” (UM-214) for complete menu and toolbar option details.

Edit menu

See “The Wave window menu bar” (UM-214) for complete menu option details.
Compare menu

The Compare menu is new in version 5.5. See Chapter 11 - Waveform Comparison for details on waveform comparisons. See also "The Wave window menu bar" (UM-214) for complete menu option details.

5.5

Bookmark menu

The Bookmark menu is new in version 5.5. See "Saving zoom range and scroll position with bookmarks" (UM-236) for details on bookmarks. See also "The Wave window menu bar" (UM-214) for complete menu option details.

5.5
Coverage_summary window changes

The coverage_summary window has been enhanced to show line misses and exclusions below the summary information.

![Coverage_summary window](image-url)
License Agreement

IMPORTANT – USE OF THIS SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS

CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE SOFTWARE

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

END USER LICENSE AGREEMENT

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

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

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

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

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

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

5. LIMITED WARRANTY.

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

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

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

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

8. INFRINGEMENT.

8.1 Mentor Graphics will defend or settle, at its option and expense, any action brought against you alleging that Software infringes a patent or copyright in the United States, Canada, Japan, Switzerland, Norway, Israel, Egypt, or the
European Union. Mentor Graphics will pay any costs and damages finally awarded against you that are attributable to the claim, provided that you: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the claim; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the claim.

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

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

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

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

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

11. RESTRICTED RIGHTS NOTICE. Software has been developed entirely at private expense and is commercial computer software provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the U.S. Government or a U.S. Government subcontractor is subject to the restrictions set forth in the license agreement under which Software was obtained pursuant to DFARS 227.7202-3(a) or as set forth in subparagraphs (c)(1) and (2) of the Commercial
12. THIRD PARTY BENEFICIARY. For any Software under this Agreement licensed by Mentor Graphics from Microsoft or other licensors, Microsoft or the applicable licensor is a third party beneficiary of this Agreement with the right to enforce the obligations set forth in this Agreement.

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

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

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

Rev. 03/00
Index


Symbols

- .main clear command CR-39
- .wave.tree interrupt command CR-40
- .wave.tree zoomfull command CR-41
- .wave.tree zoomin command CR-42
- .wave.tree zoomlast command CR-43
- .wave.tree zoomout command CR-44
- .wave.tree zoomrange command CR-45
- ’delayed CR-25

Numerics

- 64-bit ModelSim using with 32-bit FLI apps UM-108

A

- abort command CR-46
- Absolute time CR-18
- Accelerated packages UM-47
- add button command CR-47
- add list command CR-49
- add wave command CR-58
- add_menu command CR-52
- add_menucb command CR-54
- add_menuitem simulator command CR-55
- add_separator command CR-56
- add_submenu command CR-57
- alias command CR-62
- architecture simulator state variable UM-398
- argc simulator state variable UM-398

Arrays

- indexes CR-15
- slices CR-15

AssertFile .ini file variable UM-388

AssertionFormat .ini file variable UM-388

Assertions

- selecting severity that stops simulation UM-260

B

Base (radix)

- specifying in List window UM-177

Batch_mode command CR-63

Batch-mode simulations UM-418, UM-419

bd (breakpoint delete) command CR-64

Binary radix

- mapping to std_logic values CR-22

bookmark add wave command CR-65

bookmark delete wave command CR-66

bookmark goto wave command CR-67

bookmark list wave command CR-68

bookmarks UM-236

bp (breakpoint) command CR-69

Break

- on signal value CR-298

Break on assertion UM-260

BreakOnAssertion .ini file variable UM-388

Breakpoints

- continuing simulation after CR-199
- deleting CR-64
- setting CR-69

Time-based breakpoints in when statements CR-300

viewing CR-69

breakpoints

- deleting with the mouse UM-199
- enabling and disabling UM-201

- setting file-line breakpoints UM-199
- setting signal breakpoints UM-192

- setting with the mouse UM-199

viewing in the Source window UM-195

Bus contention checking UM-422

- configuring CR-76
- disabling CR-77
- enabling CR-75

Bus float checking UM-422

- configuring CR-79
- disabling CR-80
- enabling CR-78

Busse, user-defined UM-149

Button Adder (add buttons to windows) UM-263

Buttons

- adding to the Main window button bar CR-47

C

- cd (change directory) command CR-72
- Cell libraries UM-90
- change command CR-73
- change_menu_cmd command CR-74
- check contention add command CR-75
- check contention config command CR-76
check contention off command CR-77
check float add command CR-78
check float config command CR-79
check float off command CR-80
check stable off command CR-81
check stable on command CR-82
checkpoint command CR-83
Checkpoint/restore UM-416
CheckpointCompressMode .ini file variable UM-388
CheckSynthesis .ini file variable UM-386
clear differences UM-314
clocked comparison UM-297, UM-302, UM-306
Code Coverage
  coverage report command UM-294
  coverage_summary window UM-286
  enabling code coverage UM-286, UM-294
  excluding lines/files UM-287, UM-291
  invoking code coverage with vsim UM-54
  miss and exclusion details UM-287
  saving coverage reports UM-288
  Tcl preference variables UM-294
Command reference UM-18
CommandHistory .ini file variable UM-388
Command-line mode UM-418
Commands
  compare commands UM-317
  graphic interface commands UM-271
  VSIM Tcl commands UM-370
commands
  .main clear CR-39
  .wave.tree interrupt CR-40
  .wave.tree zoomfull CR-41
  .wave.tree zoomin CR-42
  .wave.tree zoomlast CR-43
  .wave.tree zoomout CR-44
  .wave.tree zoomrange CR-45
  abort CR-46
  add button CR-47
  add list CR-49
  add wave CR-58
  add_menu CR-52
  add_menucb CR-54
  add_menuitem CR-55
  add_separator CR-56
  add_submenu CR-57
  alias CR-62
  batch_mode CR-63
  bd (breakpoint delete) CR-64
  bookmark add wave CR-65
  bookmark delete wave CR-66
  bookmark goto wave CR-67
  bookmark list wave CR-68
  bp (breakpoint) CR-69
  cd (change directory) CR-72
  change CR-73
  change_menu_cmd CR-74
  check contention add CR-75
  check contention config CR-76
  check contention off CR-77
  check float add CR-78
  check float config CR-79
  check float off CR-80
  check stable off CR-81
  check stable on CR-82
  checkpoint CR-83
  compare annotate CR-87, CR-90
  compare clock CR-88
  compare close CR-93
  compare delete CR-92
  compare info CR-94
  compare list CR-95
  compare open CR-105
  compare options CR-96
  compare region CR-84
  compare reload CR-99
  compare savediffs CR-102
  compare saverules CR-103
  compare see CR-104
  compare start CR-101
  configure CR-109
  coverage clear CR-112
  coverage reload CR-113
  coverage report CR-114
  dataset alias CR-115
  dataset clear CR-116
  dataset close CR-117
  dataset info CR-118
  dataset list CR-119
  dataset open CR-120
  dataset rename CR-121
  delete CR-122
  describe CR-123
  disable_menu CR-125
  disable_menuitem CR-126
  disablebp CR-124
  do CR-127
  down CR-128
  drivers CR-130
  dumplog64 CR-131
  echo CR-132
  edit CR-133
  enable_menu CR-135
enable_menuitem CR-136
enablebp CR-134
environment CR-137
examine CR-138
exit CR-141
find CR-142
force CR-144
getactivecursortime CR-147
getactivemarkertime CR-148
help CR-149
history CR-150
lecho CR-151
left CR-152
log CR-154
lshift CR-156
lsublist CR-157
macro_option CR-158
modelsim CR-159
next CR-160
noforce CR-161
nolog CR-162
notation conventions CR-10
notepad CR-164
noview CR-165
nowhen CR-166
onbreak CR-167
onElabError CR-168
onerror CR-169
pause CR-170
play CR-171
power add CR-172
power report CR-173
power reset CR-174
printenv CR-175
profile clear CR-176
profile interval CR-177
profile off CR-178
profile on CR-179
profile option CR-180
profile report CR-181
property list CR-184
property wave CR-185
pwd CR-186
quietly CR-187
quit CR-188
radix CR-189
record CR-190
report CR-191
restart CR-193
restore CR-195
resume CR-196
right CR-197
run CR-199
search CR-201
searchlog CR-203
seetime CR-205
shift CR-206
show CR-207
splitio CR-208
status CR-209
step CR-210
stop CR-211
tb (traceback) CR-212
toggle add CR-213
toggle report CR-214
toggle reset CR-215
transcribe CR-216
transcript CR-217
TreeUpdate CR-306
tssi2mti CR-218
up CR-219
variables referenced in CR-18
vcd add CR-221
vcd checkpoint CR-222
vcd comment CR-223
vcd dumpports CR-224
vcd dumpportsall CR-226
vcd dumpportsflush CR-227
vcd dumpportslimit CR-228
vcd dumpportsoff CR-229
vcd dumpportson CR-230
vcd file CR-231, CR-233
vcd flush CR-235
vcd limit CR-236
vcd off CR-237
vcd on CR-238
vcom CR-240
vdel CR-246
vdir CR-247
vgencomp CR-248
view CR-250
virtual count CR-252
virtual define CR-253
virtual delete CR-254
virtual describe CR-255
virtual expand CR-256
virtual function CR-257
virtual hide CR-260
virtual log CR-261
virtual nohide CR-263
virtual nolog CR-264
virtual region CR-266
virtual save CR-267
virtual show CR-268
virtual signal CR-269
virtual type CR-272
vlib CR-273
vlog CR-274
vmake CR-282
vmap CR-283
vsim CR-284
vsimDate CR-296
WaveActivateNextPane CR-306
WaveRestoreCursors CR-306
WaveRestoreZoom CR-306
when CR-298
where CR-302
wlf2log CR-303
write format CR-305
write list CR-307
write preferences CR-308
write report CR-309
write transcript CR-310
write tssi CR-311
write wave CR-313
Comment characters in VSIM commands CR-10
compare
add clock UM-303
add region UM-305
add signals UM-301
by signal UM-301
clear differences UM-314
clocked UM-297, UM-302, UM-306
commands UM-317
differences UM-297, UM-304, UM-307
difference markers UM-311
differences UM-315
end UM-313
graphical interface UM-299
icons UM-312
limit count UM-308
list window display UM-316
menu UM-312
modify clock UM-303
options UM-308
pathnames UM-311
preference variables UM-317
reference dataset UM-299
reference region UM-305
reload UM-315
rules UM-315
run UM-313
save differences UM-314
show differences UM-314
signal options UM-302
specify dataset UM-299
specify when expression UM-304
start UM-312
startup wizard UM-312
tab UM-300
test dataset UM-299
test region UM-305
timing differences UM-311
tolerance UM-304, UM-307
tolerances UM-297
values UM-311
verilog matching UM-308
VHDL matching UM-308
wave window display UM-310
waveforms UM-295
write report UM-314
compare annotate command CR-87, CR-90
compare by region UM-305
compare clock command CR-88
compare close command CR-93
compare commands UM-317
compare delete command CR-92
compare info command CR-94
compare list command CR-95
compare open command CR-105
compare options command CR-96
compare region command CR-84
compare reload command CR-99
compare savediffs command CR-102
compare saverules command CR-103
compare see command CR-104
compare simulations UM-131
compare start command CR-101
compare waveforms UM-131, UM-222
comparison modes UM-297
comparison wizard UM-312
Compilation and Simulation
Verilog UM-68–UM-119
VHDL UM-51–UM-62
Compiler directives UM-99
IEEE Std 1364-2000 UM-99
XL compatible compiler directives UM-99
Compiling
invoking the VHDL compiler UM-53
locating source errors UM-245
setting default options UM-246
setting options in projects UM-31
setting order in projects UM-30
Verilog
  incremental compilation UM-70
  optimizing performance UM-83
  XL `uselib compiler directive UM-75
  XL compatible options UM-73
Verilog compile options UM-248
Verilog designs CR-274
  optimizing performance CR-275
VHDL UM-53
VHDL compile options UM-246
VHDL designs CR-240
  at a specified line number (-line <number>) CR-241
  selected design units (-just eapbc) CR-241
  standard package (-s) CR-243
with the graphic interface UM-244
with VITAL packages UM-62
Component declaration
generating VHDL from Verilog UM-128
  with vgencomp UM-128
concatenation
  directives CR-20
  of signals CR-20, CR-269
Concatenation of signals CR-20, CR-269
ConcurrentFileLimit .ini file variable UM-389
configuration simulator state variable UM-398
Configurations
  simulating CR-284
  configure command CR-109
Constants
  displaying values of CR-123, CR-138
context menus
  coverage_source window UM-291
  described UM-148
  Library page UM-41
  Signal window UM-192
  Structure pages UM-134
continuous comparison UM-297, UM-304
convert real to time UM-66
convert time to real UM-65
coverage clear command CR-112
coverage reload command CR-113
coverage report command CR-114, UM-294
coverage_summary window UM-286
  link to Dataflow window UM-166
  Wave window UM-233

D

Dataflow window UM-166
dataset
  specifying with the environment command CR-137
dataset alias command CR-115
Dataset Browser UM-136
dataset clear command CR-116
dataset close command CR-117
dataset info command CR-118
dataset list command CR-119
dataset open command CR-120
dataset rename command CR-121
datasets UM-131, UM-296
  managing UM-136
  reference UM-299
  restrict dataset prefix display UM-137
  simulator time resolution UM-132
  specifying for compare UM-299
  test UM-299
DatasetSeparator .ini file variable UM-389
Declarations
  hiding implicit with explicit declarations CR-245
Default compile options UM-246
DefaultForceKind .ini file variable UM-389
DefaultRadix .ini file variable UM-389
DefaultRestartOptions variable UM-389, UM-395
Defaults
  restoring UM-382
  window arrangement UM-148
Delay
  detecting infinite zero-delay loops UM-424
  interconnect CR-287, CR-292, UM-79
  modes for Verilog models UM-90
  SDF files UM-319
  specifying stimulus delay UM-191
DelayFileOpen .ini file variable UM-389
delete command CR-122
deleting library contents UM-40
collapse deltas in the List window UM-174
detecting infinite zero-delay loops UM-424
DelayFileOpen .ini file variable UM-389
delete command CR-122
deleting library contents UM-40
collapse deltas in the List window UM-174
DelayFileOpen .ini file variable UM-389
delete command CR-122
deleting library contents UM-40
collapse deltas in the List window UM-174
DelayFileOpen .ini file variable UM-389
delete command CR-122
deleting library contents UM-40
collapse deltas in the List window UM-174
DelayFileOpen .ini file variable UM-389
delete command CR-122
deleting library contents UM-40
collapse deltas in the List window UM-174
DelayFileOpen .ini file variable UM-389
delete command CR-122
deleting library contents UM-40
Delta
  referencing simulator iteration as a simulator state variable UM-398
  cycles UM-424
  simulator state variable UM-398
Dependent design units UM-53
describe command CR-123
Descriptions of HDL items UM-202
Design hierarchy
  viewing in Structure window UM-204
Design library
assigning a logical name UM-43  
creating UM-39  
for VHDL design units UM-53  
mapping search rules UM-44  
resource type UM-38  
working type UM-38  
Design stability checking UM-423  
Design units UM-38  
adding Verilog units to a library CR-274  
report of units simulated CR-309  
viewing hierarchy UM-149  
Directories  
  mapping libraries CR-283  
  moving libraries UM-45  
disable_menu command CR-125  
disable_menuitem command CR-126  
disablebp command CR-124  
do command CR-127  
DO files (macros)  
  error handling UM-378  
  passing parameters to UM-377  
  Tcl source command UM-379  
Do files, see macros CR-127  
documentation UM-19  
DOPATH environment variable UM-383  
down command CR-128  
drivers command CR-130  
dumplog64 command CR-131  
dumpports tasks  
  VCD files UM-335  

E  

echo command CR-132  
edit command CR-133  
Editing  
  in notepad windows UM-162, UM-403  
  in the Main window UM-162, UM-403  
  in the Source window UM-162, UM-403  
EDITOR environment variable UM-383  
enable_menu command CR-135  
enable_menuitem command CR-136  
enablebp command CR-134  
end comparison UM-313  
ENDFILE function UM-58  
ENDLINE function UM-58  
Entities  
  selecting for simulation CR-294  
  entity simulator state variable UM-398  
  Environment  
  displaying or changing pathname CR-137  
  environment command CR-137  
  Environment variables UM-383  
  accessed during startup UM-34  
  for locating license file UM-408  
  referencing from ModelSim command line UM-385  
  referencing with VHDL FILE variable UM-385  
  setting before compiling or simulating UM-383  
  setting in Windows UM-384  
  specify transcript file location with TranscriptFile UM-390  
  specifying library locations in modelsim.ini file UM-386  
  specifying UNIX editor CR-133  
  used in Solaris linking for FLI UM-105  
  using with location mapping UM-425  
  variable substitution using Tcl UM-369  
  viewing current names and values with printenv CR-175  

Errors during compilation, locating UM-245  
  event order  
    changing in Verilog CR-274  
  Event order issues UM-78  
  examine command CR-138  
  exclusion filter UM-287  
  exit command CR-141  
  Explicit .ini file variable UM-387  
  Expression Builder UM-269, UM-306  
    specify when expression UM-304, UM-306, UM-307  
  Expression_format CR-19  
  Extended identifier UM-126  

F  

F8 function key UM-165, UM-405  
  file-line breakpoints UM-199  
find command CR-142  
Finding  
  a cursor in the Wave window UM-234  
  a marker in the List window UM-182  
Finding names, and searching for values UM-147  
FLEXlm license manager UM-407–UM-413  
  administration tools for Windows UM-413  
  license server utilities UM-412  
force command CR-144  
  force command defaults UM-394  
Foreign language interface  
  tracing UM-118  
format file
Wave window UM-213
format list CR-305
format wave CR-305

G

GenerateFormat .ini file variable UM-389
Generics
assigning or overriding values with -g and -G CR-285
examing generic values CR-138
Generics, VHDL UM-122
get_resolution() VHDL function UM-63
getactivemsurtorm time command CR-147
getactivemarkertime command CR-148
Graphic interface UM-143–UM-273
UNIX support UM-15
graphical interface
waveform comparison UM-299
GUI_expression_format CR-19
GUI expression builder UM-269
syntax CR-23

H

halting waveform drawing CR-40
Hazard .ini file variable (VLOG) UM-388
Hazards
  event order issues UM-78
HDL item UM-19
help command CR-149
Hierarchical profile UM-279
history command CR-150
History shortcuts CR-12, UM-403
hm_entity UM-357
HOME environment variable UM-383

I

ieee .ini file variable UM-386
IEEE libraries UM-47
IEEE std 1076 UM-16, UM-51
IEEE std 1364 UM-16, UM-68
IgnoreError .ini file variable UM-389
IgnoreFailure .ini file variable UM-389
IgnoreNote .ini file variable UM-389
IgnoreVitalErrors .ini file variable UM-387
IgnoreWarning .ini file variable UM-389
Implicit operator, hiding with vcom -explicit CR-245
Incremental compilation
  automatic UM-71
  manual UM-71
  with Verilog UM-70
Indexing signals, memories and nets CR-15
init_signal_spy UM-64
initial dialog box
  turning on/off UM-382
Initialization sequence UM-35
Installation
  locating the license file UM-408
Instantiation in mixed-language design
  Verilog from VHDL UM-126
  VHDL from Verilog UM-129
Instantiation label UM-205
Interconnect delay CR-292
Interconnect delays UM-79, UM-330
Iteration_limit
  detecting infinite zero-delay loops UM-424
IterationLimit .ini file variable UM-389

K

Keyboard shortcuts
  List window UM-182, UM-402
  Main window UM-403
  Main windows UM-162
  Source window UM-403
  Wave window UM-238, UM-401

L

lecho command CR-151
left command CR-152
Libraries
  64-bit and 32-bit in same library UM-48
  alternate IEEE libraries UM-47
  creating design libraries CR-273, UM-39
  design library types UM-38
  design units UM-38
  ieee_numeric UM-47
  ieee_synopsis UM-47
  listing contents CR-247
  lock file, unlocking CR-242, CR-278
  mapping from the command line UM-44
  mapping hierarchy UM-393
  mapping search rules UM-44
  mapping with the GUI UM-43
  moving UM-45
  naming UM-43
Index

predefined UM-46
rebuilding ieee_numeric UM-47
refreshing library images CR-243, CR-279, UM-47
resource libraries UM-38
setting up for groups UM-421
std UM-46
verilog UM-72, UM-123
VHDL library clause UM-46
working libraries UM-38
working with contents UM-40
libraries
modelsim_lib UM-63
library simulator state variable UM-398
Licensing
License variable in .ini file UM-390
locating the license file UM-408
using the FLEXlm license manager UM-407
list CR-49
List window UM-170
waveform comparison UM-316
LM_LICENSE_FILE environment variable UM-383
lmdown license server utility UM-412
lmgrd license server utility UM-412
lmremove license server utility UM-413
lmreread license server utility UM-413
lmstat license server utility UM-412
lmutil license server utility UM-412
Locating source errors during compilation UM-245
Location maps
referencing source files UM-425
Locked memory
under HP-UX 10.2 UM-427
under Solaris UM-428
LockedMemory .ini file variable UM-390
log command CR-154
Log file
nolog command CR-162
virtual log command CR-261
virtual nolog command CR-264
Logic Modeling
SmartModel
command channel UM-350
SmartModel Windows
lmcwin commands UM-351
memory arrays UM-352
lshift command CR-156
lsublist command CR-157

M

macro_option command CR-158
MacroNestingLevel simulator state variable UM-398
Macros (DO files)
creating from a saved transcript UM-153
depth of nesting, simulator state variable UM-398
DO files (macros) UM-377
error handling UM-378
parameter as a simulator state variable (n) UM-398
parameter total as a simulator state variable UM-398
passing parameters to UM-377
startup macros UM-394
Macros (do files)
executing CR-127
executing at breakpoints CR-70
forcing signals or nets CR-144
passing parameters to CR-127
relative directories CR-127
shifting parameter values CR-206
Main window UM-151
Mapping Verilog states in mixed designs UM-124
math_complex package UM-47
math_real package UM-47
Memory
enabling shared memory on Sun/Solaris UM-428
locked memory under HP-UX 10.2 UM-427
modeling in VHDL UM-430
Menus
customizing menus and buttons UM-148
Dataflow window UM-167
List window UM-171
Main window UM-154
Process window UM-185
Signals window UM-188
Source window UM-196
Structure window UM-205
tearing off or pinning menus UM-148
Variables window UM-208
Wave window UM-214
Messages
echoing CR-132
redirect UM-390
turning off assertion messages UM-394
turning off warnings from arithmetic packages UM-394
MG C_LOCATION_MAP variable UM-383
Miss and Exclusion details UM-287
Mixed-language simulation UM-121
MODEL_TECH environment variable UM-383
MODEL_TECH_TCL environment variable UM-383
Modeling memory in VHDL UM-430
ModelSim
  custom setup with daemon options UM-410
  license file UM-408
modelsim command CR-159
ModelSim commands CR-29—CR-304
  comments in commands CR-10
MODELSIM environment variable UM-383
modelsim.ini
  default to VHDL93 UM-395
  hierarchial library mapping UM-393
  opening VHDL files UM-395
  to specify a startup file UM-394
  turning off arithmetic warnings UM-394
  turning off assertion messages UM-394
  using environment variables in UM-393
  using to create a transcript file UM-393
  using to define force command default UM-394
  using to define restart command defaults UM-395
  using to delay file opening UM-395
MODELSIM_TCL environment variable UM-383
Mouse shortcuts
  Main window UM-162, UM-403
  Source window UM-403
  Wave window UM-238, UM-401
MPF file
  loading from the command line UM-32
MTI_TF_LIMIT environment variable UM-383
Multiple drivers on unresolved signal UM-383
multiple simulations UM-131

N
n simulator state variable UM-398
Name case sensitivity for VHDL and Verilog CR-16
Names
  alternative signal names in the List window (-label) CR-50
  alternative signal names in the Wave window (-label) CR-59
Negative pulses
  driving an error state CR-293, UM-82
negative timing checks UM-95
Nets
  adding to the Wave and List windows UM-191
  applying stimulus to CR-144
  displaying drivers of CR-130
  displaying in Dataflow window UM-166
  displaying values in Signals window UM-187
  examining values CR-138
  forcing signal and net values UM-190
  saving values as binary log file UM-191
  viewing waveforms UM-210
New features UM-437
Next and previous edges, finding UM-238, UM-402
Next and previous edges, searching for CR-153, CR-198
next command CR-160
No space in time literal UM-247
NoCaseStaticError.ini file variable UM-387
NoDebug.ini file variable (VCOM) UM-387
NoDebug.ini file variable (VLOG) UM-388
noforce command CR-161
nolog command CR-162
NoOthersStaticError.ini file variable UM-387
notepad command CR-164
Notepad windows, text editing UM-162, UM-403
noview command CR-165
NoVital.ini file variable UM-387
NoVitalCheck.ini file variable UM-387
Now simulator state variable UM-398
now simulator state variable UM-398
  special considerations UM-399
nowhen command CR-166
numeric_bit package UM-47
numeric_std package UM-47
NumericStdNoWarnings.ini file variable UM-390

O
onbreak command CR-167
onElabError command CR-168
onerror command CR-169
Operating systems supported UM-15
Optimize for std_logic_1164 UM-248
Optimize_1164.ini file variable UM-387
Optimizing Verilog with -fast UM-83
  without source UM-89
order of event
  changing in Verilog CR-274

P
Packages
  standard UM-46
textio UM-46
  vital_memory UM-47
packages
  util UM-63
Parameters, using with macros UM-377
pathnames UM-311
PathSeparator .ini file variable UM-390
pause command CR-170
Performance
  enabling shared memory UM-428
  improving for Verilog simulations UM-83
  improving on HP-UX 10.2 UM-427
  improving on Sun/Solaris UM-428
Performance Analyzer UM-275
  %parent field UM-281
  commands UM-284
  getting started UM-277
  hierarchical profile UM-279
  in(%) field UM-280
  interpreting data UM-277
  name field UM-280
  profile report command UM-283
  ranked profile UM-281
  report option UM-283
  setting preferences UM-284
  statistical sampling UM-276
  under(%) field UM-280
  view_profile command UM-279
  view_profile_ranked command UM-280
  viewing results UM-278
play command CR-171
PLI UM-101
PLIOBJS environment variable UM-384
Popup
  toggling Waveform popup on/off UM-211, UM-230, UM-311
port driver data
  capturing UM-341
Ports
  VHDL and Verilog UM-123
Postscript
  saving a waveform in UM-239
power add command CR-172
power report command CR-173
power reset command CR-174
Precedence
  of variables UM-397
Pre-compiled libraries
  optimizing with -fast UM-89
preference variables
  waveform compare UM-317
Preferences
  performance analyzer preferences UM-284
printenv command CR-175
printing
  comparison differences UM-315
Process window UM-184
Process without a wait statement UM-247
Processes
  displayed in Dataflow window UM-166
  values and pathnames in Variables window UM-207
profile clear command CR-176
profile interval command CR-177
profile off command CR-178
profile on command CR-179
profile option command CR-180
profile report command CR-181, UM-283
Project files
  modelsim.ini
    MODELSIM environment variable UM-383
  override mapping for work directory with vcom CR-244
  override mapping for work directory with vlog CR-279
  modelsim.mpf
    project definition UM-22
projects
  accessing from the command line UM-32
  adding files to UM-27
  changing compile order UM-30
  compiling the files UM-28
  creating UM-25
  customizing settings UM-30
  differences in 5.5 UM-23
  loading a design UM-29
  setting compiler options in UM-31
property list command CR-184
property wave command CR-185
'protect compiler directive CR-242, CR-277, UM-420
Pulse error state CR-293, UM-82
pwd command CR-186
Q
Quiet .ini file variable (VCOM) UM-387
Quiet .ini file variable (VLOG) UM-388
quietly command CR-187
quit command CR-188
R
Radix
  changing in Signals, Variables, Dataflow, List, and
  Wave windows CR-189
  of signals in Wave window CR-60
  specifying in List window UM-177
specifying in Signals window UM-190
to examine CR-139
radix command CR-189
Ranked profile UM-281
real type
  converting to time UM-66
Rebuilding supplied libraries UM-47
Reconstruct RTL-level design busses UM-139
record command CR-190
Records
  changing values of UM-207
Redirect messages
  TranscriptFile UM-390
reference region UM-305
reference signals UM-296
Refreshing library images CR-243, CR-279, UM-47
Register variables
  adding to the Wave and List windows UM-191
displaying values in Signals window UM-187
saving values as binary log file UM-191
viewing waveforms UM-210
report command CR-191
RequireConfigForAllDefaultBinding variable UM-387
Resolution UM-54, UM-63
Resolution .ini file variable UM-390
resolution simulator state variable UM-398
resolution, simulator time CR-288
Resource library UM-38
Restart UM-157, UM-160, UM-220
restart command CR-193
restart command defaults UM-395
restore command CR-195
Restoring defaults UM-382
resume command CR-196
right command CR-197
run command CR-199
RunLength .ini file variable UM-390

troubleshooting UM-331
Verilog
  $sdf_annotate system task UM-324
  optional conditions UM-328
  optional edge specifications UM-327
  rounded timing values UM-329
  SDF to Verilog construct matching UM-325
Verilog SDF annotation UM-324

VHDL
  Resolving errors UM-323
  SDF to VHDL generic matching UM-322
search command CR-201
Searching
  binary signal values in the GUI CR-22
  for values and finding names in windows UM-147
List window
  signal values, transitions, and names CR-19, CR-128, CR-219, UM-180
  next and previous edge in Wave window CR-152, CR-197
Verilog libraries UM-72
waveform
  signal values, edges and names CR-152, CR-197, UM-202, UM-206, UM-231
searchlog command CR-203
searchLog simulator command UM-141
setetime command CR-205
Shared memory
  improving performance on Sun/Solaris UM-428
shift command CR-206

Shortcuts
  command history CR-12, UM-403
  command line caveat CR-12, UM-403
  List window UM-182, UM-402
  Main window UM-403
  Main windows UM-162
  Source window UM-403
text editing UM-162, UM-403
Wave window UM-238, UM-401
show command CR-207
show differences UM-314
Show source lines with errors UM-247
Show source .ini file variable (VCOM) UM-387
Show source .ini file variable (VLOG) UM-388
Show_VitalChecksWarning .ini file variable UM-387
Show_Warning1 .ini file variable UM-387
Show_Warning2 .ini file variable UM-387
Show_Warning3 .ini file variable UM-387
Show_Warning4 .ini file variable UM-387
Show_Warning5 .ini file variable UM-387
signal breakpoints UM-192

S

clearing SDF .ini file variable UM-387, UM-388
elements of data UM-387
errors and warnings UM-321
instance specification UM-320
interconnect delays UM-330
mixed VHDL and Verilog designs UM-330
specification with the GUI UM-321

ModelSim SE User’s Manual
Signal spy UM-64
Signal transitions searching for UM-234
Signals
  adding to a WLF file UM-191
  adding to the Wave and List windows UM-191
  alternative names in the List window (-label) CR-50
  alternative names in the Wave window (-label) CR-59
  applying stimulus to CR-144, UM-190
  combining into a user-defined bus UM-149
  creating a signal log file CR-154
  displaying drivers of CR-130
  displaying environment of CR-137
  displaying in Dataflow window UM-166
  displaying values in Signals window UM-187
  examining values CR-138
  finding CR-142
  indexing arrays CR-15
  pathnames in VSIM commands UM-64
  referencing in the hierarchy UM-64
  saving values as binary log file UM-191
  selecting signal types to view UM-189
  specifying force time CR-145
  specifying radix of in List window CR-50
  specifying radix of in Wave window CR-60
  specifying radix of signal to examine CR-139
  viewing waveforms UM-210
Signals window UM-187
Simulating
  applying stimulus to signals and nets UM-190
  batch mode UM-418
  command-line mode UM-418
  comparing simulations UM-131, UM-295
  mixed Verilog and VHDL Designs
    compilers UM-122
    libraries UM-122
    Verilog parameters UM-123
    Verilog state mapping UM-124
    VHDL and Verilog ports UM-123
    VHDL generics UM-122
  optimizing Verilog performance CR-275
  saving simulations CR-154, CR-289, UM-131, UM-421
  saving waveform as a Postscript file UM-239
  setting default run length UM-260
  setting iteration limit UM-260
  setting time resolution UM-252
  specifying design unit CR-284
  specifying the time unit for delays CR-18
  speeding-up with Performance Analyzer UM-275
  stepping through a simulation CR-210
  stopping simulation in batch mode CR-300
Verilog
  delay modes UM-90
  event order issues UM-78
  hazard detection UM-79
  optimizing performance UM-83
  resolution limit UM-77
  XL compatible simulator options UM-79
VHDL UM-54
  invoking code coverage UM-54
  viewing results in List window UM-170
  with the graphic interface UM-250
  with VITAL packages UM-62
Simulation and Compilation
  Verilog UM-68–UM-119
  VHDL UM-51–UM-62
  simulator resolution
    returning as a real UM-63
    simulator time resolution (vsim -t) CR-288
    simulator version CR-289, CR-296
    simultaneous events in Verilog
      changing order CR-274
      sizetf callback function UM-112
    sm_entity UM-347
  SmartModels
    creating foreign architectures with sm_entity UM-347
    invoking SmartModel specific commands UM-350
    lmcwin commands UM-351
    memory arrays UM-352
    Verilog interface UM-353
    VHDL interface UM-346
Software updates UM-20
  software version UM-159
Sorting
  sorting HDL items in VSIM windows UM-148
Source code
  source code security UM-420
  Source directory, setting from source window UM-196
  Source files
    referencing with location maps UM-425
Source window UM-195
  Specify path delays CR-293, UM-82
  specify when expression UM-306
  Speeding-up the simulation UM-275
  splitio command CR-208
Stability checking
  disabling CR-81
  enabling CR-82
Standards supported UM-16
Startup
alternate to startup.do (vsim -do) CR-285
environment variables access during UM-34
files accessed during UM-33
macro in the modelsim.ini file UM-390
startup macro in command-line mode UM-418
using a startup file UM-394
Startup .ini file variable UM-390
Startup macros UM-394
Status bar
Main window UM-162
status command CR-209
std .ini file variable UM-386
std_developerskit .ini file variable UM-386
Std_logic
mapping to binary radix CR-22
std_logic_arith package UM-47
std_logic_signed package UM-47
std_logic_unsigned package UM-47
StdArithNoWarnings .ini file variable UM-390
STDOUT environment variable UM-384
step command CR-210
Stimulus
applying to signals and nets UM-190
stop command CR-211
Structure window UM-204
Support UM-20
Symbolic link to design libraries (UNIX) UM-44
synopsys .ini file variable UM-386
system calls
VCD UM-334
Verilog UM-92
System initialization UM-33
system tasks
VCD UM-334
Verilog UM-92

T

_tab stops
in the Source window UM-203
_tb command CR-212
Tcl UM-361–UM-372
command separator UM-368
command substitution UM-367
command syntax UM-364
evaluation order UM-368
history shortcuts CR-12, UM-403
Man Pages in Help menu UM-159
relational expression evaluation UM-368
variable substitution UM-369
VSIM Tcl commands UM-370
Technical support UM-20
test region UM-305
test signals UM-296
Text and command syntax UM-18
Text editing UM-162, UM-403
TextIO package UM-51
alternative I/O files UM-59
containing hexadecimal numbers UM-58
dangling pointers UM-58
ENDFILE function UM-58
ENDLINE function UM-58
file declaration UM-55
implementation issues UM-57
providing stimulus UM-59
standard input UM-56
standard output UM-56
WRITE procedure UM-57
WRITE_STRING procedure UM-57
Time
handling negative timing constraints UM-95
setting the resolution UM-54, UM-77, UM-252
simulation time units CR-18
time resolution as a simulator state variable UM-398
time type
converting to real UM-65
timing differences UM-296, UM-311
TMPDIR environment variable UM-384
to_real VHDL function UM-65
to_time VHDL function UM-66
toggle add command CR-213
Toggle checking UM-423
toggle report command CR-214
toggle reset command CR-215
Toggle statistics
enabling CR-213
reporting CR-214
resetting CR-215
toggling Waveform popup on/off UM-211, UM-230,
UM-311
tolerance
leading edge UM-304, UM-307
trailing edge UM-304, UM-307
tolerances UM-297
Toolbar
Main window UM-160
Wave window UM-218
Tracing HDL items with the Dataflow window UM-168
transcribe command CR-216
Transcript
saving to a file UM-393

transcript command CR-217

Transcript file
saving UM-153
TranscriptFile variable in .ini file UM-390

Tree windows
VHDL and Verilog items in UM-149
viewing the design hierarchy UM-149

TreeUpdate command CR-306

Triggers, setting in the List window UM-174, UM-434
TSSI CR-311
in VCD files UM-341
tssi2mti command CR-218
type
converting real to time UM-66
converting time to real UM-65

U

Unbound Component UM-247
UnbufferedOutput .ini file variable UM-391
up command CR-219
UpCase .ini file variable UM-388
Updates UM-20
Use 1076-1993 language standard UM-246
Use clause
specifying a library UM-46
Use explicit declarations only UM-247
User-defined bus UM-138, UM-149
UserTimeUnit .ini file variable UM-391
util package UM-63

V

Values
describe HDL items CR-123
examine HDL item values CR-138
Values of HDL items UM-202
Variable settings report CR-18

Variables
environment variables UM-383
LM_LICENSE_FILE UM-383
loading order at ModelSim startup UM-33
personal preferences UM-382
precedence between .ini and .tcl UM-397
reading from the .ini file UM-392
setting environment variables UM-383
simulator state variables
current settings report UM-382
iteration number UM-398
name of entity or module as a variable UM-398
resolution UM-398
simulation time UM-398

Variables window UM-207

Variables, HDL
changing value of on command line CR-73
changing value of with the GUI UM-207
describing CR-123
examining values CR-138

Variables, Tcl CR-18
vcd add command CR-221
vcd checkpoint command CR-222
vcd comment command CR-223
dumpports command CR-224
dumpportsall command CR-226
dumpportsflush command CR-227
dumpportslimit command CR-228
dumpportsoff command CR-229
dumpportson command CR-230
vcd file command CR-231, CR-233
VCD files
adding items to the file CR-221
capturing port driver data CR-224, UM-341
capturing to WLF files CR-239
dumping variable values CR-222
dumpports tasks UM-335
flushing the buffer contents CR-235
from VHDL source to VCD output UM-338
inserting comments CR-223
specifying maximum file size CR-236
specifying the file name CR-231, CR-233
state mapping CR-231, CR-233
supported TSSI states UM-341
turn off VCD dumping CR-237
turn on VCD dumping CR-238
VCD system tasks UM-334
vcd flush command CR-235
vcd limit command CR-236
vcd off command CR-237
vcd on command CR-238
vcd2wlf command CR-239
vcom command CR-240
vdel command CR-246
vdir command CR-247
Verilog
capturing port driver data with -dumpports CR-232,
UM-341
cell libraries UM-90
compile options UM-248
compiler directives UM-99
compiling design units UM-69
compiling with XL `uselib compiler directive UM-75
component declaration UM-128
creating a design library UM-69
instantiation criteria in mixed-language design UM-126
instantiation of VHDL design units UM-129
library usage UM-72
mapping states in mixed designs UM-124
mixed designs with VHDL UM-121
parameters UM-123
SDF annotation UM-324
sdf_annotate system task UM-324
simulating UM-77
delay modes UM-90
event order issues UM-78
XL compatible options UM-79
simulation hazard detection UM-79
simulation resolution limit UM-77
SmartModel interface UM-353
source code viewing UM-195
standards UM-16
system tasks UM-92
XL compatible compiler options UM-73
XL compatible routines UM-118
XL compatible system tasks UM-95
verilog .ini file variable UM-386
Verilog PLI
64-bit support UM-118
callback reason argument UM-111
registering applications UM-101
support for VHDL objects UM-114
Verilog PLI/VPI UM-101–UM-119
compiling and linking PLI/VPI applications UM-104
debugging PLI/VPI code UM-118
specifying the PLI/VPI file to load UM-108
Verilog Procedural Interface UM-101
Veriuser .ini file variable UM-391
version
obtaining UM-159
obtaining with vsim command CR-289
obtaining with vsim<info> commands CR-296
vgencomp command CR-248
VHDL
compile options UM-246
compiling design units UM-53
creating a design library UM-53
delay file opening UM-395
dependency checking UM-53
field naming syntax CR-16
file opening delay UM-395
Hardware Model interface UM-356
instantiation from Verilog UM-129
instantiation of Verilog UM-122
library clause UM-46
mixed designs with Verilog UM-121
object support in PLI UM-114
simulating UM-54
SmartModel interface UM-346
source code viewing UM-195
standards UM-16
VITAL package UM-47
VHDL utilities UM-63, UM-64
get_resolution() UM-63
to_real() UM-65
to_time() UM-66
VHDL93 .ini file variable UM-387
view command CR-250
view_profile command UM-279
view_profile_ranked command UM-280
Viewing and saving waveforms CR-289, UM-131,
UM-214
Viewing design hierarchy UM-149
viewing library contents UM-40
virtual count commands CR-252
virtual define command CR-253
virtual delete command CR-254
virtual describe command CR-255
virtual expand commands CR-256
virtual function command CR-257
virtual hide command CR-260, UM-139
virtual log command CR-261
virtual nohide command CR-263
virtual nolog command CR-264
Virtual objects UM-138
virtual functions UM-139
virtual regions UM-140
virtual signals UM-138
virtual types UM-140
virtual region command CR-266, UM-140
Virtual regions
reconstruct the RTL Hierarchy in gate level design
UM-140
virtual save command CR-267, UM-139
virtual show command CR-268
virtual signal command CR-269, UM-138
Virtual signals
reconstruct RTL-level design busses UM-139
reconstruct the original RTL hierarchy UM-139
virtual hide command UM-139
virtual type command CR-272
VITAL
  compiling and simulating with accelerated VITAL packages UM-62
  compliance warnings UM-61
  obtaining the specification and source code UM-60
  VITAL 2000 library UM-47
  VITAL packages UM-61
vlib command CR-273
vlog command CR-274
vmake command CR-282
vmap command CR-283
VPI UM-101
VSIM build date and version CR-296
vsim command CR-284
VSIM commands
  searchLog UM-141
  vsimVersion CR-296

W

Warnings
  turning off warnings from arithmetic packages UM-394
wave
  adding CR-58
  Wave format file UM-213
  Wave log format (WLF) CR-289
  Wave log format (WLF) file UM-131
  of binary signal values CR-154
Wave window UM-210
  compare waveforms UM-310
  toggling Waveform popup on/off UM-211, UM-230, UM-311
  values column UM-311
WaveActivateNextPane command CR-306
Waveform
  halting drawing CR-40
Waveform Comparison UM-295
  add clock UM-303
  add region UM-305
  adding signals UM-301
  clear differences UM-314
  clocked comparison UM-297, UM-302, UM-306
  commands UM-317
  compare by region UM-305
  compare by signal UM-301
  compare commands UM-317
  compare menu UM-312
  compare options UM-308
  compare tab UM-300
  comparison method UM-309
  comparison method tab UM-306
  comparison modes UM-297
  comparison wizard UM-312
  continuous comparison UM-297, UM-304, UM-307
  dataset UM-296
  difference markers UM-311
  end UM-313
  features UM-296
  flattened designs UM-298
  graphical interface UM-299
  hierarchical designs UM-298
  icons UM-312
  introduction UM-296
  leading edge tolerance UM-304, UM-307
  limit count UM-308
  List window display UM-316
  modify clock UM-303
  pathnames UM-311
  preference variables UM-317
  printing differences UM-315
  reference dataset UM-299
  reference region UM-305
  reference signals UM-296
  reload UM-315
  rules UM-315
  run
    run comparison UM-313
    save differences UM-314
    show differences UM-314
    signal options UM-302
    specify when expression UM-304, UM-306, UM-307
    specifying a dataset UM-299
    start UM-312
    Tcl preference variables UM-317
    test dataset UM-299
    test region UM-305
    test signals UM-296
    timing differences UM-296, UM-311
    tolerances UM-297
    trailing edge tolerance UM-304, UM-307
    values column UM-311
    Verilog matching UM-308
    VHDL matching UM-308
    Wave window display UM-310
    write report UM-314
  waveform comparison CR-84, UM-131, UM-222
  Waveform popup UM-211, UM-230, UM-311
  Waveforms UM-131
saving UM-214
saving and viewing UM-421
saving as a .eps file UM-214
viewing UM-210
WaveRestore Cursors command CR-306
WaveRestore Zoom command CR-306
WaveSignalNameWidth .ini file variable UM-391
Welcome dialog
  turning on/off UM-382
  when command CR-298
  where command CR-302
Wildcard characters
  for pattern matching in simulator commands CR-17
Windows
  finding HDL item names UM-147
  opening from command line CR-250
  opening multiple copies UM-148
  opening with the GUI UM-156
  searching for HDL item values UM-147
  adding buttons UM-263
  coverage_source UM-290
  coverage_summary UM-286
Dataflow window UM-166
  tracing signals and nets UM-168
List window UM-170
  adding HDL items UM-175
  adding signals with a WLF file UM-191
  examining simulation results UM-179
  formatting HDL items UM-176
  locating time markers UM-147
  output file CR-307
  saving the format of CR-305
  saving to a file UM-183
  setting display properties UM-173
  setting triggers UM-174, UM-434
Main window UM-151
  adding user-defined buttons CR-47
  status bar UM-162
  text editing UM-162, UM-403
  time and delta display UM-162
  toolbar UM-160
Process window UM-184
  displaying active processes UM-184
  specifying next process to be executed UM-184
  viewing processing in the region UM-184
saving position and size UM-148
Signals window UM-187
  VHDL and Verilog items viewed in UM-187
Source window UM-195
  setting tab stops UM-203
  text editing UM-162, UM-403

viewing HDL source code UM-195
Structure window UM-204
  HDL items viewed in UM-204
  instance names UM-205
  selecting items to view in Signals window UM-187
  VHDL and Verilog items viewed in UM-204
  viewing design hierarchy UM-204
Variables window UM-207
  displaying values UM-207
  VHDL and Verilog items viewed in UM-207
Wave window UM-210
  adding HDL items UM-213
  adding signals with a WLF file UM-191
  changing display range (zoom) UM-234
  changing path elements CR-111, UM-391
  cursor measurements UM-234
  locating time cursors UM-147
  saving format file UM-213
  searching for HDL item values UM-231
  setting display properties UM-229
  using time cursors UM-233
  zoom options UM-235
  zooming UM-234
WLF file UM-191
  log command CR-154
  WLF files UM-296, UM-421
  wlf2log command CR-303
  Work library UM-38
  workspace UM-152
  write
    waveform comparison report UM-314
    write format command CR-305
    write list simulator command CR-307
    write preferences command CR-308
    write report command CR-309
    write transcript command CR-310
    write tssi command CR-311
    write wave command CR-313

Z

Zero-delay loop, detecting infinite UM-424
Zero-delay oscillation UM-424
Zoom
  from Wave toolbar buttons UM-235
  from Zoom menu UM-234
  options UM-235
  saving range with bookmarks UM-236
  with the mouse UM-235