ISim User Guide

UG660 (v 11.3) September 16, 2009

XILINX®

Xilinx Trademarks and Copyright Information

Xilinx is disclosing this user guide, manual, release note, and/or specification (the “Documentation”) to you solely for use in the development of designs to operate with Xilinx hardware devices. You may not reproduce, distribute, republish, download, display, post, or transmit the Documentation in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Xilinx expressly disclaims any liability arising out of your use of the Documentation. Xilinx reserves the right, at its sole discretion, to change the Documentation without notice at any time. Xilinx assumes no obligation to correct any errors contained in the Documentation, or to advise you of any corrections or updates. Xilinx expressly disclaims any liability in connection with technical support or assistance that may be provided to you in connection with the Information.

THE DOCUMENTATION IS DISCLOSED TO YOU “AS-IS” WITH NO WARRANTY OF ANY KIND. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE DOCUMENTATION, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT OF THIRD-PARTY RIGHTS. IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOSS OF DATA OR LOST PROFITS, ARISING FROM YOUR USE OF THE DOCUMENTATION.

© Copyright 2002-2009 Xilinx Inc. All Rights Reserved. XILINX, the Xilinx logo, the Brand Window and other designated brands included herein are trademarks of Xilinx, Inc. All other trademarks are the property of their respective owners. The PowerPC name and logo are registered trademarks of IBM Corp., and used under license. All other trademarks are the property of their respective owners.

Table of Contents

Xilinx Trademarks and Copyright Information ... 2
Chapter 1 Getting Started ... 11
ISim Overview ... 11
Simulation Libraries ... 11
Language Support ... 11
Feature Support ... 11
ISim Operating System Support ... 12
What's New in ISim 11.3 ... 12
What's New in ISim 11.1 and 11.2 ... 12
General Features ... 12
Command Line Enhancements ... 13
GUI Enhancements ... 13
Tcl Enhancements ... 13
Steps in Simulation ... 13
ISim Modes of Operation ... 13
Steps in a Simulation Overview ... 15
Step 1: Gathering Files and Mapping Libraries ... 15
Stimulus File ... 15
User Libraries ... 15
Step 2: Parsing and Elaborating the Design ... 15
Graphical User Interface Mode ... 15
Interactive Command Line Mode ... 15
Step 3: Simulating the Design ... 16
Graphical User Interface Mode ... 16
Interactive Command Line Mode ... 17
Step 4: Examining the Design ... 17
Step 5: Debugging the Design ... 17
Examining Error Messages ... 17
Examining Log Files ... 18
Using Tcl Simulation Commands ... 18
Debugging in the Graphical User Interface ... 18
Tutorial ... 18
Deprecated Features and Commands ... 18
Deprecated File Types ... 18
Deprecated Commands ... 18
Deprecated Command Line Options ... 19
Other Changes ... 19
Chapter 2 Exploring the ISim Graphical User Interface ... 21
Graphical User Interface Overview ... 21
Launching the GUI ... 21
Parts of the GUI ... 22
Design Objects and Icons ... 23
Design Hierarchy Icon ... 23
Design Object Icons ... 23
Arranging the Main Window ... 24
Arranging Windows ... 24
Using Window Commands ... 24
Using Drag and Drop ... 24
Hiding and Restoring Windows ... 24
Menu Commands ... 25
Toggle Icons ... 25
Expanding and Collapsing a Hierarchy ... 26
Clicking the Arrow ... 26
Menu Command ... 26
Using the Context Menu ... 26

Wave Window 26
Wave Window Overview 26
Wave Configuration File (.wcfg) 26
Signals/Buses in the Wave Configuration 27
Objects in the Wave Configuration 28
Wave Window Toolbar Icons 28
Working With Wave Configurations 29
Creating New Wave Configurations 29
To Create a Wave Configuration 29
Adding Signals to the Wave Configuration 29
In the GUI 29
Using Tcl 29
Saving Wave Configurations 30
Opening and Closing Wave Configurations 30
To Open a Wave Configuration File 30
To Close a Wave Configuration File 30
Instances and Processes Panel 30
Instances and Processes Panel Overview 30
Searching For Objects 31
Opening HDL Source Files 31
Objects Panel 31
Objects Panel Overview 31
Objects Panel Toolbar Icons 32
Searching For Objects 32
Using Show Drivers 32
Showing Display Elements 33
To Display all Child Elements 33
To Limit the Display of Child Elements 33
To Change the Preset Maximum Number of Child Elements 33
Selecting an Object in the Wave Window 33
Source Files Panel 33
Source Files Panel Overview 33
Opening the Source Code 33
Text Editor Window 34
Text Editor Window Overview 34
To Modify Source Files Safely 34
Opening HDL Source Files 34
Viewing HDL Source Files 34
Setting Breakpoints 35
Console Panel 35
Console Panel Overview 35
Breakpoints Panel 35
Breakpoints Panel Overview 35
Breakpoint Toolbar Icons 36
Search Results Panel 36
Search Results Panel Overview 36
Search Results Toolbar Commands 36
Find in Files Results Panel 36
Using Find in Files 36
Toolbar Commands and Shortcuts 37
ISim Toolbar Commands 37
Standard Toolbar 37
Edit Toolbar 37
Help Toolbar 37
Window Toolbar 37
View Toolbar 38
ISim Toolbar 38
Shortcuts 39
Keyboard Shortcuts 39

ISim Preferences 39
Setting ISim Preferences 39
ISE Text Editor Preferences 40
ISim Simulator Preferences 40
ISim Color Preferences 40
Chapter 3 VHDL Simulation 41
VHDL Simulation Overview 41
Running a Functional Simulation of a VHDL Design From the Command Line 41
Method 1: Using a Project File - Recommended 41
Method 2: Parsing the Files Using vhpcomp 42
Simulation 42
Running a Timing Simulation From the Command Line With a VHDL Design 43
Generating Timing Simulation Model 43
Method 1: Using a Project File - Recommended 43
Method 2: Using vhpcomp 44
Simulation 44
Library Mapping File 45
Search Order 45
Syntax 45
Example 45
Features/Limitations 46
Interactive Simulation in Command Line Mode 46
Chapter 4 Verilog Simulation 47
Verilog Simulation Overview 47
Running a Functional Simulation of a Verilog Design From the Command Line 47
Method 1: Using a Project File - Recommended 47
Method 2: Parsing the Files Using vlogcomp 48
Simulation 49
Running a Timing Simulation of a Verilog Design From the Command Line 49
Generating Timing Simulation Model 49
Method 1: Using a Project File - Recommended 49
Method 2: Parsing the Files Using vlogcomp 50
Simulation 51
Search Order for Instance of Verilog Design Units 51
Supporting Source Libraries 51
Library Location (-sourcelibdir) 51
Source File Extension (-sourcelibext) 51
Source File (-sourcelibfile) 52
Library Mapping File 52
Search Order 52
Syntax 53
Example 53
Features/Limitations 53
Predefined XILINX_SIM Macro for Verilog Simulation 53
Interactive Simulation in Command Line Mode 53
Chapter 5 Mixed Language Simulation 55
Mixed Language Simulation Overview 55
Restrictions on Mixed Language in Simulation 55
Key Steps in a Mixed Language Simulation 55
Instantiating Mixed Language Components 55
Instantiating a Verilog Module in a VHDL Design Unit 55
Instantiating a VHDL Module in a Verilog Design Unit 56
Mixed Language Binding and Searching 56
VHDL Instantiation Unit 57
Verilog Instantiation Unit 57
Mixed Language Boundary and Mapping Rules 57
General 57
Port Mapping 58
Generics (Parameters) Mapping 58

VHDL/Verilog Values Mapping ... 58
Chapter 6 Waveform Analysis ... 61
Before Analysis ... 61
Launching the ISim GUI ... 61
To Launch the GUI ... 61
What to Expect ... 61
Adding Signals to the Wave Configuration ... 61
In the GUI ... 61
Using Tcl ... 62
Adding Copies of Signals or Buses ... 62
Running a Simulation in ISim ... 62
Stopping a Simulation ... 62
To Stop a Running Simulation ... 63
Closing ISim ... 63
Customizing the Wave Configuration ... 63
Placing Cursors ... 63
To Place the Main Cursor ... 63
To Place the Secondary Cursor ... 63
To Move a Cursor ... 63
Setting Markers ... 64
Adding a Marker ... 64
Moving a Marker ... 64
Deleting a Marker ... 64
Adding Dividers ... 65
Adding Groups ... 65
Adding Virtual Buses ... 65
Renaming Objects ... 66
Changing the Display Names ... 66
Changing the Radix ... 67
To Set the Default Radix ... 67
To Change an Individual Radix ... 67
Reversing Bus Bit Order ... 67
Navigating the Wave Configuration ... 67
Expanding and Collapsing a Hierarchy ... 67
Clicking the Arrow ... 67
Menu Command ... 68
Using the Context Menu ... 68
Zooming In and Out ... 68
Using Commands ... 68
Using the Middle Mouse Wheel ... 69
Displaying the Floating Ruler ... 69
Displaying Waveform Values With Markers ... 69
To Display Waveform Values With Markers ... 70
Displaying Waveform Values at Signal Transitions ... 70
Measuring Time with Cursors ... 70
To Measure Time With Cursors ... 70
Measuring Time with Markers ... 71
Using Go To Time ... 71
To Go To a User-Specified Time ... 71
To Go to Time 0 or Latest Time ... 72
Using Show Drivers ... 72
Printing Wave Configurations ... 72
To Print Preview ... 72
To Print ... 72
Using Custom Colors ... 73
Creating Custom Color Schemes ... 73
Changing Signal Colors ... 73
Chapter 7 Saving and Opening Simulation Results ... 75
Saving the Results ... 75

Saving a Database to a WDB File ... 75
Saving a Wave Configuration to a WCFG File ... 75
Relationship Between WDB and WCFG ... 75
Opening a Live Simulation ... 75
Opening a Waveform Database ... 76
Opening a Wave Configuration ... 76
Opening a Static Simulation ... 76
Opening a Wave Configuration and Waveform Database ... 76
Opening a Waveform Database Only ... 77
Chapter 8 Debugging ... 79
Source Level Debugging Overview ... 79
Stepping ... 79
Stepping Through a Simulation ... 79
Breakpoints ... 80
Setting Breakpoints ... 80
Debugging Your Design Using Breakpoints ... 80
Deleting Breakpoints ... 81
To Remove a Breakpoint ... 81
To Remove All Breakpoints ... 81
Chapter 9 Writing Activity Data for Power Consumption ... 83
Writing Activity Data of the Design ... 83
Activity File Types ... 83
To Write Activity Files for Power ... 84
Chapter 10 Using Tcl Simulation Commands ... 85
Simulation Command Overview ... 85
Simulation Command Entry Methods ... 85
Simulation Command Summary ... 85
Entering Simulation Commands ... 86
To Enter Commands in GUI Mode ... 87
To Enter Commands in Interactive Command Line Mode ... 87
To Enter Commands in Non-Interactive Batch Mode ... 88
Aliasing Simulation Commands ... 88
Command Line Conventions ... 88
Tcl Commands ... 89
bp Command ... 89
Options ... 89
describe Command ... 90
Options ... 90
dump Command ... 91
help Command ... 91
Options ... 91
isim condition Command ... 92
Options ... 92
isim force Command ... 93
Options ... 94
isim get arraydisplaylength Command ... 94
isim get radix Command ... 95
isim get userunit Command ... 95
isim ltrace Command ... 95
Options ... 95
isim ptrace Command ... 96
Options ... 96
isim set arraydisplaylength Command ... 96
Options ... 96
isim set radix Command ... 97
Options ... 97
isim set userunit Command ... 98
Options ... 98
ntrace Command ... 98

Options 98
onerror Command 100
Options 100
put Command 100
Options 101
quit Command 103
Options 103
restart Command 103
resume Command 104
run Command 104
Options 104
saif Command 105
Options 105
scope Command 105
Options 106
sdfanno Command 106
Options 106
show Command 107
Options 107
step Command 111
test Command 111
Options 111
vcd Command 113
Options 113
wave add Command 114
Options 115
Appendix Reference 117
Simulation Executable Commands 117
Simulation Executable Command Overview 117
Running Simulation From the Command Line 117
Executable Command Summary 117
Project Files 118
Tcl Commands 118
ISim Simulation Executable 118
ISim Simulation Executable Overview and Syntax 118
ISim Simulation Executable Options 118
ISim Simulation Executable Examples 120
fuse 120
fuse Overview and Syntax 120
fuse Options 121
fuse Examples 123
Using Precompiled HDL 123
Using HDL Source 124
Using the Command File Option 124
vlogcomp 125
vlogcomp Overview and Syntax 125
vlogcomp Options 125
vlogcomp Examples 126
vhpcomp 127
vhpcomp Overview and Syntax 127
vhpcomp Options 127
vhpcomp Examples 128
Third-Party Command Equivalency 128
Third-Party Simulation Command Support Overview 128
Third-Party Compiler Commands 129
Compiler Command Compatibility Table 129
Third-Party Tcl Commands 129
Simulation Tcl Command Compatibility Table 129
HDL Language Support 131

VHDL Language Support (a to m) 131
VHDL Language Support (n to z) 136
Verilog Language Coverage 139
Verilog Language Support 139
Behavioral Statement Constructs 139
Continuous Assignment Statements 139
Procedural Blocks and Assignments 139
Parallel and Sequential Bloc 139
Statements 140
Timing Control Statements 140
Conditional Statements 140
Case Statements 140
Looping Statements 140
Task Enable Statements 140
Compiler Directive Constructs 141
Compiler Directive Constructs 141
Declaration Constructs 141
Module Parameter Declarations 141
Type Declarations 141
Net Variable Types 142
Strengths 142
Delays 142
Declaration Lists 142
Declaration Assignments 143
Declaration Ranges 143
Function Declarations 143
Task Declarations 143
Block Item Declarations 143
Expression Constructs 144
Concatenations 144
Function Calls 144
Expressions 145
Primaries 145
Expresson Left-Side Values 145
Operators 145
Numbers 146
Strings 146
General Constructs 146
Attributes 146
Comments 147
Identifiers 147
Identifier Branches 148
White Space 148
Primitive and Module Instance Constructs 148
Primitive Instantiations 148
Primitive Strengths 149
Primitive Terminals 149
Primitive Gate and Switch Types 149
Module Instantiations 149
Generated Instantiation 150
Source Text Constructs 150
Library Source Text 151
Configuration Source Text 151
Module and Primitive Source Text 151
Module Parameters and Ports 151
Module Items 152
Specify Function Constructs 152
Specify Block Declarations 152
Specify Path Declarations 152

Specify Block Terminals....152
Specify Path Delays....153
System Timing Check Commands....154
System Timing Check Command Arguments....154
System Timing Check Event Definitions....155
System Task and Function Constructs....155
Display System Tasks....155
File I/O Tasks....155
Timescale Tasks....157
Simulation Control Tasks....157
PLA Modeling Tasks....157
Stochastic Analysis Tasks....157
Simulation Time Functions....158
Conversion Functions....158
Probabilistic Distribution Functions....158
Command Line Input....158
Value Change Dump (VCD) Files....159
UDP Declaration and Instantiation Constructs....159
UDP Declaration....159
UDP Ports....159
UDP Body....160
UDP Instantiation....160

XILINX®

Chapter 1

Getting Started

ISim Overview

Xilinx® ISim is a Hardware Description Language (HDL) simulator that enables you to perform functional and timing simulations for VHDL, Verilog and mixed VHDL/Verilog designs.

Simulation Libraries

The Xilinx simulation device libraries are precompiled, and are updated automatically when updates are installed. Do not run the Simulation Library Compilation Wizard (Compxlib) to compile libraries for use with ISim.

Language Support

ISim supports the following languages.

Language	Support
VHDL	IEEE-STD-1076-2000
Verilog	IEEE-STD-1364-2001
SDF	Xilinx [NetGen] generated SDF Files
VITAL	VITAL-2000
Mixed VHDL/Verilog	Yes
VHDL FLI/VHPI	No
Verilog PLI	No
SystemVerilog	No
Other Assertion-Based Languages	No

Feature Support

ISim supports the following features.

Feature	Support
Incremental Compilation	Yes
Source Code Debugging	Yes
SDF Annotation	Yes
VCD Generation	Yes
SAIF Support	Yes
Hard IP - MGT, PPC, PCIe®, etc	Yes
Multi-threading	Yes

ISim Operating System Support

ISim supports the operating systems as specified in the *Xilinx ISE Design Suite: Installation, Licensing, and Release Notes* with one exception: no native support in Windows XP/Vista 64-bit.

Note ISim may be run in 32-bit mode on Windows XP/Vista 64-bit Operating Systems. Refer to the *Xilinx ISE Design Suite: Installation, Licensing, and Release Notes* for information on running Xilinx® tools in 32-bit mode.

What's New in ISim 11.3

- Changes to time measurement using markers, cursors, and the floating ruler.
- Design Unit name displayed in the Instances and Processes panel.
- Ability to go to the source code at the line where the design unit is instantiated from the Instances and Processes panel or Search Results panel.
- Ability to set the default radix in the Preferences dialog box.
- Improvements to Zoom function shortcut keys, and ability to zoom to marker from the context menu in the Wave window.
- New shortcut key sequence (Ctrl+W) to add objects to the Wave window.
- Changes to viewing and opening Waveform Database file and Wave Configuration file in the static simulator (for example, **`isimgui -view WDB.wdb`**).
- Ability to control the font size and style of the Console panel in the Preferences dialog box.
- Ability to clear the Console panel using context-menu command.

What's New in ISim 11.1 and 11.2

General Features

- A FLEXnet Publisher license is required for full version of ISim. ISim Lite version is available when no FLEXnet Publisher license is detected. For more information about software licensing, see the *Xilinx ISE Design Suite: Installation, Licensing, and Release Notes*.
- A new static waveform viewer is available to open static simulation.
- Multi-processing is turned on by default in HDL compiler (fuse), which reduces compile time up to 2X.
- Switching Activity Interface format (SAIF) files can be generated for power estimation using the XPower Analyzer tool.

Command Line Enhancements

- Ability to launch ISim graphical user interface for command-line users.
- Ability to set the Verilog timescale at elaboration time (fuse -timescale)
- Ability to change the default time precision for VHDL designs (fuse -timeprecision_vhdl)
- Ability to change the top-level generic/parameters at runtime (fuse -generic_top)
- Support for Verilog source libraries.
- Support for $value$plusarg and $test$plusarg system tasks (<sim_executable> -testplusarg)
- A built-in macro called XILINX_ISIM defined for Verilog users

GUI Enhancements

- ISim graphical user interface (GUI) improvements:
  - Crisp, new look and feel
  - Improved scrolling
  - Improved zoom capability
  - New ruler for measuring
  - Support for virtual buses
  - Ability to add copies of signals
- Ability to find a text string in a set of files.
- Ability to save and reload wave configuration
- Ability to load multiple wave configurations per simulation session

Tcl Enhancements

- Ability to turn on line-by-line analysis for debugging (isim ltrace)
- Ability to control the behavior following a failed Tcl simulation command (onerror)
- Ability to start simulation without reloading the design (restart)
- Forcibly (without needing confirmation) exit the waveform GUI (quit -f)
- Ability to add simulation objects or blocks to wave configuration (wave add)

Steps in Simulation

ISim Modes of Operation

There are three modes of operation available in ISim:

- Graphical User Interface
- Interactive Command Line
- Non-Interactive Batch

Mode of Operation	Features	How ISim Is Invoked
Graphical User Interface	Graphical view of simulation data. Menu commands, context commands, and toolbar buttons are used to run simulation, and examine and debug data. Also, Tcl commands entered at Console prompt are used to run simulation, and examine and debug data. For information about working with the graphic user interface, see Graphical User Interface Overview.	• From ISE® - Run a simulation process on your design, such as, Simulate Behavioral Model. • From Command Prompt - Run simulation executable with `-gui` switch, for example, `my_sim.exe -gui`. • From Command Prompt - Run simulation executable with `-gui` switch and `-view <file.wcfg>` to open wave configuration file, and open a previous simulation. **Note** The GUI can also be opened in read-only mode using `isimgui.exe –view <wcfg_file>.wcfg`
Interactive Command Line	No interaction with graphical user interface. Commands run at command prompt. After simulation executable run, Tcl prompt opens in which simulation Tcl commands are entered to examine and debug data. For more information, see VHDL Simulation Overview, Verilog Simulation Overview or Mixed Language Simulation Overview.	From Command Prompt - 1. Run command to generate simulation executable. For example, `fuse -prj my_prj.prj tb -L unisims_ver -L userlib -o my_sim.exe` 2. Run simulation executable, for example, `my_sim.exe`.
Non-Interactive Batch	No interaction with graphical user interface. A single command sequence is run, and all actions are controlled through use of command switches and the contents of a batch file containing Tcl commands. For more information, see ISim Simulation Executable Overview and Syntax.	From Command Prompt - 1. Run command to generate simulation executable. For example, `fuse -prj my.prj tb -L mylib -L yourlib -o my_sim.exe` 2. Create a file with Tcl commands to run. 3. Run simulation executable with the `-tclbatch` switch. For example, `my_sim.exe -tclbatch cmd.tcl`.

Steps in a Simulation Overview

The basics steps for simulating your design in ISim are as follows:

- Step 1: Gathering Files and Mapping Libraries
- Step 2: Parsing and Elaborating the Design
- Step 3: Simulating the Design
- Step 4: Examining the Design
- Step 5: Debugging the Design

Step 1: Gathering Files and Mapping Libraries

The files needed to run a simulation in ISim are the following:

- design files, including stimulus file
- any user libraries
- any other miscellaneous data files

Stimulus File

Include an HDL-based test bench as the stimulus file. Create or edit your test bench using any of the following means:

- **Text Editor -** Create or edit an HDL test bench in any text editor.
- **Language Templates -** Use a template to populate the file correctly, such as those available with the ISE® software. For more information, see "Using the Language Templates" in ISE Help.
- **Third-party tool -** Create or edit an HDL test bench in any vendor-provided tool.

User Libraries

Depending on which use mode is used to launch ISim, there are two different methods available to add user libraries:

- If launching ISim from Project Navigator, define user libraries in the ISE software. See "Working with VHDL Libraries" in ISE Help for details.
- If using ISim standalone, Interactive Command mode, or non-Interactive mode, set the library mapping file to point to the user logical/physical libraries.

Step 2: Parsing and Elaborating the Design

Prior to running a simulation, ISim must parse the code into one or more libraries, and then elaborate the design components upon which the design depends. The simulation executable is generated during this step.

Graphical User Interface Mode

For users wishing to launch the ISim graphical user interface, there are two options:

- The design is parsed and design components are elaborated for you when you invoke ISim from the ISE® software. For details, see "Simulation from ISE Software" in Step 3: Simulating the Design.
- The design is parsed and elaborated manually at the command line, as described in the next section. And then the generated simulation executable is invoked with **`-gui`** to launch graphical user interface.

Interactive Command Line Mode

There are two steps in the interactive command-line mode: (1) creating a project file, and (2) using **`fuse`** to parse the design using the project file, elaborate the design, and generate the simulation executable.

Creating a Project File

The project file (PRJ) is used with the **fuse** command to provide a list of all the files associated with a design. The PRJ file contains the language, library name and the design file.

1. Create a text file, and assign it a `.prj` file extension.
2. In the project file starting from the first line, enter library and source file information as follows:

   ```
   verilog|vhdl <library_name> {<file_name_1>.v|.vhd}
   verilog|vhdl <library_name> {<file_name_2>.v|.vhd}
   .
   verilog|vhdl <library_name> {<file_name_n>.v|.vhd}
   ```

   where:

   - *verilog*|*vhdl* indicates that the source is a Verilog or VHDL file. Include either `verilog` or `vhdl`.
   - <*library_name*> indicates the library that a particular source on the given line should be compiled. `work` is the default library.
   - <*file_name*> is the source file or files associated with the library. More than one Verilog source can be specified on a given line. One VHDL source can be specified on a given line.

Using fuse

Use the **`fuse`** command to parse the design using the PRJ, elaborate the design, and generate the simulation executable. For example:

```
fuse -prj my_project.prj work.top work.glbl -o my_sim.exe
```

For more information about the fuse command syntax and available switches, see fuse Overview

Make sure that this step was run successfully. If not, see "Examining Error Messages" and "Examining Log Files" in Step 5: Debugging the Design.

Step 3: Simulating the Design

Once design compilation and elaboration are complete, the next step is to run the simulation executable, and simulate the design.

For information about running the ISim in read-only mode, see Opening a Static Simulation.

Graphical User Interface Mode

Simulation at the Command Line

From the last step, a simulation executable was generated (**`x.exe`** (default) or **`my_sim.exe`** (user specified). Now, run the simulation executable with the **`-gui`** switch , for example, **`my_sim.exe -gui`**. This command launches the ISim graphical user interface (GUI). The simulation executable command does not start the simulation. In order to start the simulation, use one of the Run simulation commands described in Running a Simulation in ISim. You will also need to add signals to the wave configuration. See Launching the ISim GUI for details.

Optionally, you can also invoke the simulation executable, launch the GUI, and run simulation with a Tcl file by leveraging the **`-tclbatch`** option, for example, **`my_sim.exe -gui -tclbatch my_sim.tcl`**. You can use the wave add command (example: **`wave add /`** to add all signals at top level) in your `my_sim.tcl` file to automatically trace the signals and display the signals in the GUI upon launch.

Simulation from the ISE software

Parsing, elaboration, and running the simulation executable command are all run in the background when you run one of the following processes in the ISE® software.

- **Simulate Behavioral Model -** See "Performing Behavioral Simulation" in ISE Help for details
- **Simulate Post-Place & Route Model -** See "Performing Timing Simulation" in ISE Help for details.

These ISE processes launch the ISim GUI with the top-level signals being traced by default. Optionally, user can specify custom Tcl files inside ISE to control the signals that would get traced upon launch of ISim GUI. The simulator will run for the time specified under the ISE simulation process property "Simulation Run Time". See "Simulation Properties" in ISE Help for details.

To run for an additional time, use one of the Run simulation commands described in Running a Simulation in ISim. For more information, see Launching the ISim GUI.

Interactive Command Line Mode

Run the simulation executable, for example, **`my_sim.exe`**. When the Tcl prompt displays, type in the run command.

Optionally, you can also invoke the simulation executable with a Tcl file by leveraging the **`-tclbatch`** option, for example, **`my_sim.exe -tclbatch my_sim.tcl`**.

Make sure that this step was run successfully. If not, see "Examining Error Messages" and "Examining Log Files" in Step 5: Debugging the Design.

Step 4: Examining the Design

Once the design is simulated, it is more than likely that you will need to debug the design in order to ensure that it meets the design specification.

Initial examination of the simulation results can be done in two ways:

1. Viewing the signal interactions in the Wave window.
2. Viewing or querying the results in the Console panel or the Tcl prompt. See Simulation Command Overview.

Results can be saved. For more information, see Saving the Results.

You can also view and examine simulation results in a read-only static simulator. For more information, see Opening a Static Simulation.

Step 5: Debugging the Design

If problems are encountered, debugging will be necessary to identify the root cause and resolution for the problems. ISim provides a variety of ways to debug the design.

Examining Error Messages

First, look at the error messages to see if there are any errors in the design. Error messages appear in the ISE® software Console (GUI mode) and the log files discussed in the next section. Look for messages with one of the following prefixes:

- **HDL Compiler -** Indicates an error during the parsing or static elaboration step. If an error occurs during parsing and elaboration, and this step was not run successfully, the problem can be an HDL compiler issue. Enter **`fuse -v 1`** to dump information that may help identify the problem. Also, a `fuse.log` file contains a list of error messages, and errors also appear in the ISE software Console (in ISE Integration Mode).
- **Simulator -** Indicates an error during executable code generation or simulation. See Step 3: Simulating the Design.

Use the file name and line number in the message to locate the problem.

Examining Log Files

Examining the available log files may also provide helpful clues about design errors.

- **fuse.log -** Log file containing output produced by fuse during the compilation and elaboration step.
- **isim.log -** Log file containing output produced by simulation executable during the simulation step. This file does not disclose any design data, and is safe to share with Xilinx® Technical Support if you report a problem.
- **isimcrash.log -** Log file generated when the tool encounters an unexpected error or condition. This is generated inside `isim/<simulation_executable> .sim` directory. Provide this file to Xilinx Technical Support for further assistance. This file also does not disclose any design data, and is safe to share with Xilinx Technical Support if you report a problem.

Using Tcl Simulation Commands

Several simulation commands are available to assist you with debugging. These commands can be run at the command line Tcl prompt, or in the Console panel of the ISim interface.

- isim ptrace on
- isim ltrace on
- dump
- show
- isim force
- bp
- onerror

For more commands, see Simulation Command Overview.

Debugging in the Graphical User Interface

For the debug strategies when using the GUI, see Source Level Debugging Overview.

Tutorial

For a tutorial on how to use ISim, go to the *ISim In-Depth Tutorial*.

This tutorial demonstrates how to use ISim for design simulation and debugging. To access this tutorial from the ISim software, select **Help > Tutorial**.

Deprecated Features and Commands

Deprecated File Types

Xilinx Waveform (.xwv) file, which was open in the static waveform viewer, has been changed for the Wave Database (.wdb) file. Older designs should be rerun in the ISE® 11 software, and simulated in ISim to open and save a WDB file.

Deprecated Commands

Following are commands that have been deprecated in this release. If used, they will be ignored by ISim.

Deprecated Command	Command to Use Instead
The executable named **isimwave**	**isimgui -view** <*wcfg_file*>**.wcfg** or **isimgui –view** <*wdb*>**.wdb** (Linux) **isimgui.exe -view** <*wcfg_file*>**.wcfg** or **isimgui.exe –view** <*wcfg_file*>**.wdb** (Windows) See Opening a Static Simulation.

Deprecated Tcl Command	Command to Use Instead
`dump -p`	`show child`
`dump -p` *<process_name>*	`scope` *<process_name>* `dump`
`isim batch <on/off>`	None
`show status`	None
`stop`	In command line mode, press **Ctrl+C**. In GUI mode, press the Break toolbar button ( ).

Deprecated Command Line Options

Following are the command options that have been deprecated in this release. The recommended syntax is provided.

Deprecated fuse Command Option	Syntax to Use Instead
`fuse -top` as in `fuse -top yourtop -top glbl -top thirdtop -work yourlib`	The top module now referenced as *<library_name>.<top_name>* as in `yourlib.yourtop yourlib.glbl yourlib.thirdtop` See fuse command.
`fuse -work` as in `fuse -top yourtop -top glbl -top thirdtop -work yourlib`	The library is now referenced as *<library_name>.<top_name>* as in `fuse work.yourtop work.glbl yourlib.thirdtop`

Other Changes

- The default simulation timescale has been changed from `1 fs/ 1 fs` to `1 ns/1 ps`.
- In 11.3, the **Simulation > Stop** menu command and Stop toolbar button were removed. Use another method to close ISim.

Chapter 2

Exploring the ISim Graphical User Interface

Graphical User Interface Overview

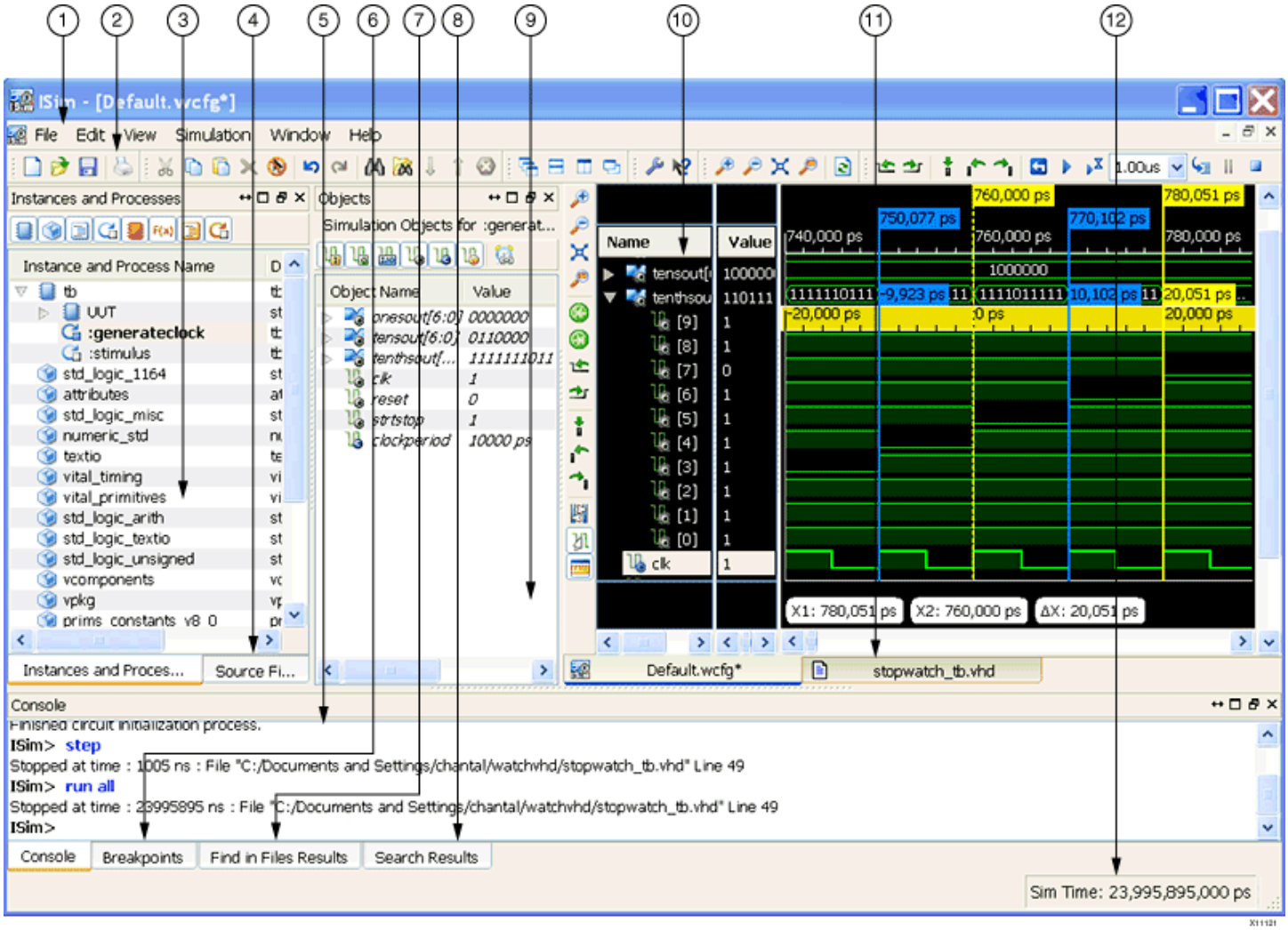
The ISim graphical user interface (GUI) consists of the main window, which contains panels, the Workspace, toolbars, and the status bar. In the main window, you can view the simulatable parts of the design, you can add and view signals in the wave configuration, you can utilize commands to run simulation, examine the design, and debug as necessary.

Launching the GUI

The ISim GUI is launched when you run the simulation executable, either from the ISE® software, or from the command line. For details, see Step 3: Simulating the Design from the Steps in Simulation.

To close the ISim software, select **File > Exit**. You will be prompted to save your waveform configuration before closing.

Parts of the GUI



#	Part of the GUI	Description
1	Menu Commands	Provides access to most operations available in the software. Some operations are available in context menu only.
2	ISim Toolbar Commands	Provides access to frequently used commands.
3	Instances and Processes Panel	Displays the block (instance and process) hierarchy associated with the current simulation.
4	Source Files Panel	Displays the list of all the files associated with the design.
5	Console Panel	Displays messages generated by the simulator. You can enter simulation Tcl commands at the prompt.
6	Breakpoints Panel	Displays a list of all breakpoints currently set in the design.
7	Find in Files Results Panel	Displays the results that match a text string in a set of files. See Using Find in Files.
8	Search Results Panel	Displays the results that match the criteria from a search.

#	Part of the GUI	Description
9	Objects Panel	Displays the simulation objects associated with the block selected in the Instances and Processes Panel.
10	Wave Window	Displays the wave configuration, which consists of a list of signals and buses, their waveforms, and any wave objects, such as dividers, cursor or markers. The Wave window can display more than one wave configuration.
11	Text Editor Window	Displays read-only HDL files.
12	Status Bar	Displays a brief description for a menu command or toolbar button that your cursor is placed over, and Simulation Time.

Design Objects and Icons

Design Hierarchy Icon

Design entities/modules are displayed in a design hierarchy in the Instances and Processes Panel.

VHDL Entity

VHDL Package

VHDL Block

VHDL Process

Verilog Module

Verilog Task or Function

Verilog Block

Verilog Process

Design Object Icons

The following design objects are displayed in the Objects Panel and in the Wave window.

Signals

Input Port

Output Port

InOut, Bidirectional Port

Internal Signal

Constants, parameters, and generics

Variable

Linkage Signal

Buffer Signal

Buses

Input Bus

Output Bus

InOut, Bidirectional Bus

Internal Bus

Constant, Parameter and Generics Bus

Variable Bus

Linkage Bus

Buffer Bus

Arranging the Main Window

Arranging Windows

You can move windows, panels and the toolbar around in the interface using one of the following techniques.

Using Window Commands

The Window menu commands are available for the Wave window and Text Editor window only.

Cascade overlaps all the windows with the active window on top.

Tile Horizontally arranges the windows horizontally as non-overlapping tiles.

Tile Vertically arranges the windows vertically as non-overlapping tiles.

Float floats the active window.

Dock restores the floated window to its former location in the interface. For floating window only.

Using Drag and Drop

For other parts of the interface, like panels and the main window toolbar, drag and drop enables you to move the object to a new location.

1. Click and hold the header for the Panel you wish to move.
2. Move the panel to a new location.

   A gray box will indicate where the panel will be placed.
3. Release the mouse button to place the panel to the new location.

Hiding and Restoring Windows

Many of the parts of the main window can be hidden from view, and restored again.

Note To restore windows to their default locations, select **View > Restore Default Layout**.

Menu Commands

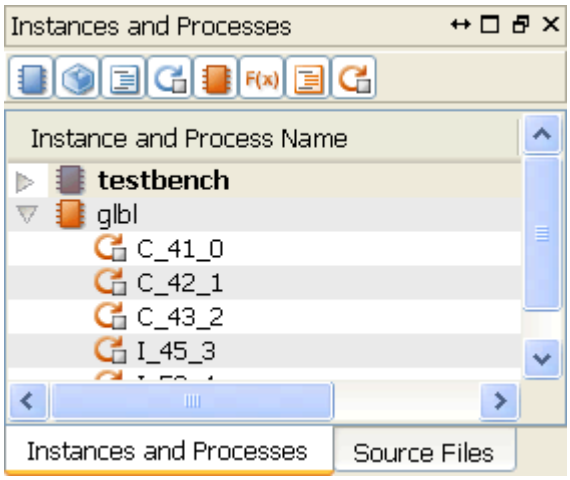
The View menu commands enable you to hide the main window panels, toolbar, and status bar.

- **View > Panels -** hides and restores the ISim panels:
  - Console
  - Search Results
  - Source Files
  - Breakpoints
  - Objects
  - Instances and Processes
  - Find in Files Results
- **View > Toolbar -** hides and restores the ISim toolbars:
  - **Standard**
  - **Edit**
  - **View**
  - **ISim**
  - **Window**
  - **Help**
- **View > Status Bar -** hides and restores the status bar located at the bottom of the main window.

Standard minimize, maximize and close commands apply to the Wave window and Text Editor window using the upper right-hand icons.

Toggle Icons

The toggle icons are available at the top right of the ISim window panels, for example, the Instances and Processes panel and the Objects panel.



With these commands, you can hide, restore, float, and dock the panel.

Toggle Slide Out - minimizes the panel. Also used to restore the pane by hovering over the pane's name at the edge of the window, and clicking the minimize button.

Toggle Maximize - maximizes the panel.

Click again to restore the panel size.

Toggle Float - floats the panel.

Click again to restore the floated window to its former location.

Close - closes the panel from view.

To restore the panel, select **View > Panels** and select the pane to restore to view.

Expanding and Collapsing a Hierarchy

You can expand and collapse a hierarchy in any window or panel with objects in nested groups using one of the following methods.

Clicking the Arrow

- Click the arrow to expand the hierarchy. One level can be expanded at a time.

- Click the arrow to collapse the hierarchy.

Menu Command

1. Select an object.
2. Select **Edit > Wave Objects >**
   - **Expand -** Expands the hierarchy object that is selected. One level can be expanded at a time.
   - **Collapse -** Collapses the hierarchy of the object selected.

Using the Context Menu

1. Select an object.
2. Right click, and select the applicable command from the context menu
   - **Expand -** Expands the hierarchy object that is selected. One level can be expanded at a time.
   - **Collapse -** Collapses the hierarchy of the object selected.

Wave Window

Wave Window Overview

The Wave window displays signals, buses and their waveforms. Each tab in the Wave window shows a wave configuration, which consists of a list of signals and buses, their properties, and any added wave objects, such as dividers, cursors, and markers. In the user interface, the signals and buses in the wave configuration are being traced during simulation, and therefore, the wave configuration is used to examine the simulation results. The design hierarchy and the signal transitions are not part of the wave configuration but rather are stored in a separate database (.wdb) file.

Wave Configuration File (.wcfg)

A wave configuration comprises a list of signals, their properties, such as color and radix value, and other wave objects, such as dividers, group, markers and cursors. A wave configuration is completely customizable, and you can add or remove signals and other wave objects at any time when the simulation isn't actively running.

The initial file called `Default.wcfg` is not saved until you save the file. Stored in the wave configuration file is the list of signals, their properties and wave objects.

Multiple wave configurations can be created and simulated, and the wave configurations can be saved separately.

For information about saving the Wave Configuration, see Saving the Results.

Signals/Buses in the Wave Configuration

The signals and buses in the Wave window can be one of the following design objects with the corresponding icon.

Signals

Input Port

Output Port

InOut, Bidirectional Port

Internal Signal

Constants, parameters, and generics

Variable

Linkage Signal

Buffer Signal

Buses

Input Bus

Output Bus

InOut, Bidirectional Bus

Internal Bus

Constant, Parameter and Generics Bus

Variable Bus

Linkage Bus

Buffer Bus

Objects in the Wave Configuration

Cursors - The main cursor and secondary cursor in the wave configuration are used to pinpoint a time (main cursor) and to measure time (main and secondary cursors together). The cursors form the focal point for various navigation activities. See Placing Cursors.

- **Main Cursor -** The main cursor is a solid line that intersects the waveform, and the value at that intersection is displayed in the Value column for each waveform. The cursor is the current simulation time while simulation is running, with the time displayed directly above the cursor.
- **Secondary Cursor -** The secondary cursor is a dotted line used with the main cursor to identify a time range. The time range can be used with zoom and print to focus on the area.

Markers - A marker is used to mark a particular time for future reference. A marker is a vertical line intersecting the waveform. A marker enables you to display the signal value where the marker intersects the waveform. The time of the marker is displayed at the top of the line. In addition, a series of markers can be used to jump the cursor forward or back for quick analysis of value change. See Adding Markers and Displaying Waveform Values With Markers.

Hollow/Filled-in Circle - When placing or moving cursors and markers, the Snap to Transition button assists with placing the cursor/marker more precisely on a signal transition. When placing or moving a cursor or marker, the mouse displays a hollow circle when hovering between transitions. The mouse displays a filled in circle when hovering over a signal's transition.

Dividers - A divider is a visual separator of signals in the wave configuration. See Adding Dividers.

Groups - A group is a collection, or "folder", to which signals and buses can be added in the wave configuration as a means of organizing a set of related signals. A group displays the icon and group name. The group itself displays no waveform data but can be expanded to show its contents or collapsed to hide them. See Adding Groups.

Virtual Buses - A virtual bus is a grouping, or "folder", to which logic scalars and arrays can be added. A virtual bus displays the icon and virtual bus name. The virtual bus displays a bus waveform, which is comprised of the signal waveforms in the vertical order that they appear under the virtual bus, flattened to a one-dimensional array. See Adding Virtual Buses.

Wave Window Toolbar Icons

The toolbar icons available with the Wave window provide easy access to frequently used commands.

Zoom Out decreases the size of the viewed objects.

Zoom In increases the size of the viewed objects.

Zoom to Full View zooms out to display the entire view in the active window.

Zoom to Cursors displays the waveforms such that the two cursors are at the left and right edge of the display. If the secondary is off, the command centers the display around the main cursor without changing the magnification level.

Go To Time 0 moves the cursor and focus to O time.

Go To Latest Time moves the cursor and focus to the end of simulation.

Go To Next Transition moves the main cursor to next transition.

Go To Previous Transition moves the main cursor to the previous transition.

Adds a marker at the position of the main cursor to the Waveform area.

Moves the main cursor to the nearest marker to the left of the main cursor's current position.

Moves the main cursor to the nearest marker to the right of the main cursor's current position.

Swaps the main and secondary cursors, if both are set.

Snap to Transition Mode moves the cursor to a transition when you place the cursor close to the transition. This mode can be switched on or off.

Displays and hides the floating ruler that can be moved to the desired location in the Wave window.

Working With Wave Configurations

Creating New Wave Configurations

You can create any number of Wave Configurations in the current session. The Wave Configuration stores the list of signals, their properties and any wave objects that were added.

To Create a Wave Configuration

1. Select **File > New**.

   The New dialog box opens.
2. Select **Wave Configuration** from the list.
3. Click **OK**.

A new untitled wave configuration is opened. The new wave configuration is empty until you add signals.

If more than one wave configuration is open, use the wave configuration tab to locate a particular wave configuration, or select **Window > Next** or **Window > Previous** to navigate through open wave configurations.

Adding Signals to the Wave Configuration

You can populate the Wave window with the signals from your design using menu commands or drag and drop capabilities in the graphical user interface, or using Tcl commands in the Console panel.

Note Changes to the wave configuration, including creating the wave configuration or adding signals to it, do not become permanent until you save the WCFG file. For more information, see Saving the Results.

In the GUI

1. In the Instances and Processes panel, expand the design hierarchy, and select an item.

   The objects that correspond to the selected instance or process displays in the Objects panel.
2. In the Objects panel, select one or more objects.
3. Use one of the following methods to add objects to the Wave Configuration:
   - Right-click, and select **Add to Wave Window** from the context menu.
   - Drag and drop the objects from the Objects panel to the Name column of the Wave window.
   - In the Console, use the wave add command as described below.

Using Tcl

1. Although optional, you may wish to first identify the objects you want to add by exploring the design hierarchy in the Instances and Processes panel and Objects panel, as described above, or by entering the scope command in the Console panel.
2. In the Console panel, enter the wave add command to enter an individual object or a group of objects.

Saving Wave Configurations

The current wave configuration can be saved. If you have multiple wave configurations open, each can be saved to a unique name for later viewing.

To Save a Wave Configuration

Select **File > Save**, press **Crtl+S**, or click the Save button .

The current simulation wave configuration is saved.

Note Use **File > Save As** to assign a different name to the wave configuration.

Opening and Closing Wave Configurations

To Open a Wave Configuration File

There are a number of ways to open a wave configuration file. Go to:

- Opening a Live Simulation
- Opening a Static Simulation

To Close a Wave Configuration File

To close the wave configuration and all its windows, select **File > Close**.

Instances and Processes Panel

Instances and Processes Panel Overview

The Instances and Processes panel displays the block (instance and process) hierarchy associated with the wave configuration open in the Wave window. Instantiated and elaborated entities/modules are displayed in a tree structure, with entity components being entities, processes, tasks and blocks.

There are three columns displayed in this panel. The first column displays the instance, process, and static task/function in a tree structure showing the block hierarchy of the design. The second column displays the names of the design units (Verilog module or VHDL entity (architecture)) corresponding to the instance, static task/function, or process from the first column. The third column displays the type of the instance, static task/function, or process.

Following is an explanation of the icon used for the items displayed in this panel:

VHDL Entity

VHDL Package

VHDL Block

VHDL Process

Verilog Module

Verilog Task or Function

Verilog Block

Verilog Process

To expand a hierarchy in order to display its components, click the arrows or use the **Expand** context menu commands. See Expanding and Collapsing a Hierarchy.

To sort the information in this panel according to the data in one of the columns, click the column title, such as Design Unit.

To hide or restore the panel, select **View > Panel > Instances and Processes**.

Searching For Objects

You can search for objects in the design using the Search command, which is available in the Instances and Processes panel and Objects panel. Search criteria includes a text string, and/or an object-type filter.

To Search For Objects

1. Place the cursor in the Objects panel or the Instances and Processes panel.
2. Right-click and select **Search** from the context menu.
3. In the Search dialog box, enter a text string. You can use an asterisk (*) as a wildcard symbol.
4. Select the object type you are searching for.
5. Click **Match case** if applicable.
6. Click **OK**.

Objects that match the search criteria are displayed in the Search Results Panel.

Opening HDL Source Files

In ISim, you can open HDL source files for viewing in the text editor.

To Open HDL Source Files

1. In the Instances and Processes panel, Objects panel, or the Source Files panel, select an item.
2. Double-click the item, or right-click and select **Go To Source** from the context menu.

The HDL source file associated with that object opens in the text editor in read-only mode.

You can also open a file using the **File > Open** menu command. In the Open dialog box, change Files of type file to Verilog or VHDL, select the file, and click **Open**. This method does not open the files in read-only mode.

Objects Panel

Objects Panel Overview

The Objects panel displays all simulation objects (ports, signals, variables, constants, parameters, and generics) associated with the selected instances and processes in the Instances and Processes panel.

The top of the panel displays which instance/process is selected in the Instances and Processes panel whose objects and their values are listed in the Objects panel.

The table columns are defined as follows:

- **Object Name -** Displays the name of the simulation object, accompanied by the symbol which represents the type of object it is.
- **Value -** The value of the simulation object at the current simulation time or at the main cursor, as determined by the Sync Time toolbar icon.
- **Data Type -** Displays the data type of the corresponding simulation object, logic or an array.

Objects Panel Toolbar Icons

Hides/displays input ports.

Hides/displays output ports.

Hides/displays inout, bidirectional ports.

Hides/displays internal signals.

Hides/displays constants, parameters, and generics.

Hides/displays variables.

Toggles the Sync Time feature on and off. When toggled on, Objects panel values are based on main cursor in the Wave window. When toggled off, values are the values at the Sim Time displayed in the Status Bar (simulation end time).

Searching For Objects

You can search for objects in the design using the Search command, which is available in the Instances and Processes panel and Objects panel. Search criteria includes a text string, and/or an object-type filter.

To Search For Objects

1. Place the cursor in the Objects panel or the Instances and Processes panel.
2. Right-click and select **Search** from the context menu.
3. In the Search dialog box, enter a text string. You can use an asterisk (*) as a wildcard symbol.
4. Select the object type you are searching for.
5. Click **Match case** if applicable.
6. Click **OK**.

Objects that match the search criteria are displayed in the Search Results Panel.

Using Show Drivers

You can use the Show Driver command to display the driver for a change in signal, or object value. This command is used to determine the cause of a value change, which will help determine if circuit connections are correct. ISim displays the signal, or object, and its one or more drivers in the Console panel.

The Show Driver command is available for probing objects in the following areas:

- Objects panel
- Wave window
- Console panel (`show driver` command)

To Show Drivers

1. Select an object, or signal.
2. Select **Edit > Wave Objects > Show Drivers**, or select **Show Drivers** from the right-click menu.

In the Console, the drivers for the object, or signal are listed. When there is no driver, a message in the Console indicates that there is no driver.

Note Running this command is the same as running `show driver` at the Console prompt.

Showing Display Elements

In the Objects panel, you can control whether or not to limit a preset maximum number of child elements displayed for every composite object. You can change the preset maximum number using the Preferences dialog box.

To Display all Child Elements

1. Right-click anywhere in the object list in the Objects panel.
2. Right click, and select **Show All Elements**.

The number of children displayed in the object hierarchy is updated immediately.

To Limit the Display of Child Elements

1. Right-click anywhere in the object list in the Objects panel.
2. Select **Limit Elements**.

To Change the Preset Maximum Number of Child Elements

Set the preference settings as follows:

1. Select **Edit > Preferences**.
2. In the Preferences dialog box, select **ISim Simulator**.
3. Select **Limit the maximum number of elements displayed to**.
4. Enter a number.
5. Click **Apply**, and **OK**.

The number of children displayed in the object hierarchy is updated immediately.

Selecting an Object in the Wave Window

Follow this procedure to highlight signals for an object in the Objects panel.

To Select an Object in the Wave Window

1. Select an object in the Objects panel.
2. Right click, and select **Select in Wave Window**.

The signal associated with the object is highlighted.

Source Files Panel

Source Files Panel Overview

The Source Files panel displays the list of all the files associated with the design. The list of files is provided by the **`fuse`** command during design parsing and elaboration, which is run in the background for GUI users. The HDL source files are available for quick access to the read-only source code.

Opening the Source Code

1. Select a file in the list.
2. Click the Go To Source Code button .

Note You can also use the **Go To Source Code** command from the context menu, or simply double-click a file.

The selected file opens in read-only mode in the Text Editor window.

Text Editor Window

Text Editor Window Overview

In ISim, the text editor window is available for easy access to the underlying HDL source files. Basic steps available are:

- Opening HDL source files
- Viewing HDL source files
- Setting breakpoints to source files for debugging.

Your HDL files should *never* be edited in the text editor in ISim. Editing files could result in a conflict with the files in your ISE® software project.

To Modify Source Files Safely

Follow these steps for modifying source files safely. Note that HDL files should only be edited outside of ISim to avoid design conflicts.

1. In ISE, open the source file in ISE Text Editor, or a third-party text editor.
2. Make the appropriate edits.
3. Run the design through ISE tools to update the design.
4. Simulate the design.

Opening HDL Source Files

In ISim, you can open HDL source files for viewing in the text editor.

To Open HDL Source Files

1. In the Instances and Processes panel, Objects panel, or the Source Files panel, select an item.
2. Double-click the item, or right-click and select **Go To Source** from the context menu.

The HDL source file associated with that object opens in the text editor in read-only mode.

You can also open a file using the **File > Open** menu command. In the Open dialog box, change Files of type file to Verilog or VHDL, select the file, and click **Open**. This method does not open the files in read-only mode.

Viewing HDL Source Files

In ISim, you can view the underlying HDL source files in the Text Editor window in order for verification of the design.

Standard text editor operations are available to assist you with viewing your HDL files, such as Zoom, and Find. Information about specific operations is available in the ISE Text Editor Help.

Your HDL files should *never* be edited in the text editor in ISim. Editing files could result in a conflict with the files in your ISE® software project.

To View Source File Contents

1. Open an HDL source file.
2. Use available features, such as scrolling, find, layout preferences, to assist you with viewing the file contents.

   Your HDL files should *never* be edited in the text editor in ISim. Editing files could result in a conflict with the files in your ISE project.

Note If edits are made to HDL files in ISim, do not save the changes.

Setting Breakpoints

When working with ISim, you can set breakpoints in executable lines in your HDL file as follows. This allows you to run your code continuously until the source code line with the breakpoint is reached, as described in Using Breakpoints to Debug Your Design.

Note You can only set breakpoints on lines with executable code.

To Set a Breakpoint

1. Select **View > Breakpoint > Toggle Breakpoint**, or click the Toggle Breakpoint toolbar button.
2. In the HDL file, click a line of code just to the right of the line number.

Note Alternatively, you can right-click a line of code, and select **Toggle Breakpoint**. You can also click a breakpoint to remove it.

Once the procedure is completed, a simulation breakpoint icon appears next to the line of code.

Note If you place a breakpoint on a line of code that is not executable, the breakpoint is not added.

A list of all breakpoints is available in the Breakpoints panel.

Console Panel

Console Panel Overview

The Console panel enables you to view a log of messages generated by ISim, and to enter standard and ISim-specific Tcl commands at the command prompt.

- **Messages -** All messages generated by the ISim including errors, warnings and informational messages are displayed in the Console panel. The Console panel also echoes all simulator commands that were invoked from the graphical controls in the ISim interface.
- **Simulation commands -** The command prompt enables you to enter simulation Tcl commands, and to view the command dump (or print-out) in the Console panel. See Entering Simulation Commands for more information.

A number of right-click menu commands are available to help manage the contents of the Console panel. The commands include cut, copy and paste, Clear Console, among others.

Breakpoints Panel

Breakpoints Panel Overview

The Breakpoints panel displays a list of all breakpoints currently set in the design. A breakpoint is a user-determined stopping point in the source code used for debugging the design with ISim. For more information, see Debugging Your Design Using Breakpoints.

See also Setting Breakpoints and Deleting Breakpoints.

For each breakpoint set in your source files, the list in the Breakpoints panel identifies the file location, file name and line number. You can delete a selection, delete all breakpoints, and go to the source code from the Breakpoint panel toolbar icons or context menu.

Breakpoint Toolbar Icons

Deletes the selected line from the Breakpoint panel, and deletes the breakpoint from the HDL source file.

Deletes all breakpoints from the HDL source files.

Open the HDL source file in the text editor with the breakpoint in focus.

Search Results Panel

Search Results Panel Overview

The Search Results panel displays the results that match the search criteria from the Search command. The results display the icon for the object type being displayed, and the location of the object in the design.

Search Results Toolbar Commands

The following functions are available in the Search Results panel.

Clears the contents of the Search Results panel.

Adds the signal associated with the selected search result to the wave configuration in the Wave window.

Opens the HDL source file in the text editor at the line where the design unit is defined.

Opens the HDL source file in the text editor at the line where the design unit is instantiated.

Stops the search.

Find in Files Results Panel

Using Find in Files

You can find a text string in a set of files as follows.

To Use Find in Files

1. Select **Edit > Find in Files,** or click the Find Text in Files toolbar button .
2. In the Find in Files dialog box, specify the text to find, set the parameters for your search, and click **Find**.
3. In the Find in Files Results panel, do any of the following:
   - To clear all results from the panel, click the Clear All toolbar button .
   - To open the file that contains the find result in the Workspace, select a find result, and click the Show Current Result toolbar button .

     Note Alternatively, you can double-click the find result to open the file.
   - To view the next find result, click the Show Next Result toolbar button .
   - To view the previous find result, click the Show Previous Result toolbar button .
   - To stop the currently running Find in Files search, click the Stop Job toolbar button .
   - To save your Find in Files search results to a CSV file, click the Save Results as a Text File toolbar button .

Toolbar Commands and Shortcuts

ISim Toolbar Commands

The toolbars available in the ISim main window consists of many functionally different toolbars. The main window toolbar icons are located near the top of the user interface.

Standard Toolbar

The Standard toolbar provides convenient access to frequently used File menu commands. To show/hide the Standard toolbar, select **View > Toolbars > Standard.**

New opens the New dialog box and lets you select the type of file you wish to create.

Open opens the Open dialog box and lets you browse through your directories and select a file to open. The file displays in the appropriate application or editor.

Save saves the active file to disk and overwrites the previously saved version. If a file is not saved previously, the Save As dialog box opens and lets you save the active file to disk.

Print opens the Print dialog box and let you print the active file.

Edit Toolbar

The Editor toolbar provides convenient access to frequently used Edit menu commands. To show/hide the Editor toolbar, select **View > Toolbars > Edit**.

Cut removes the selected text or object from the Workspace and places it in the clipboard.

Copy copies the selected text or object from the Workspace and places it in the clipboard.

Paste inserts the text or an object from the clipboard into the active window.

Delete deletes the item in focus in the Workspace window.

Unselect All unselects everything in the active window.

Undo reverses your latest operation. This button works only when there is something to undo.

Redo reverses your latest "undone" operation.

Find searches for a text string in the active window.

Help Toolbar

The Help toolbar provides convenient access to frequently used Help menu commands. To show/hide the Help toolbar, select **View > Toolbars > Help**.

Support and Services displays the Xilinx® Support page in the default Web browser.

What's This help activates. After clicking this button, select a menu item or toolbar button for which you want help.

Window Toolbar

The Window toolbar provides convenient access to frequently used Window menu commands. To show/hide the Window toolbar, select **View > Toolbars > Window**.

Cascade overlaps all the windows with the active window on top.

Tile Horizontally arranges the windows horizontally as non-overlapping tiles.

Tile Vertically arranges the windows vertically as non-overlapping tiles.

Float floats the active window.

View Toolbar

The View toolbar provides convenient access to frequently used View menu commands. To show/hide the View toolbar, select **View > Toolbars > View**.

Zoom In increases the size of the viewed objects.

Zoom Out decreases the size of the viewed objects.

Zoom to Full View zooms out to display the entire view in the active window.

Zoom to Cursors displays the waveforms such that the two cursors are at the left and right edge of the display. If the secondary is off, the command centers the display around the main cursor without changing the magnification level.

Refresh cleans up the display of the file in focus.

ISim Toolbar

The ISim toolbar provides convenient access to frequently used ISim commands. To show/hide the ISim toolbar, select **View > Toolbars > ISim**.

Go To Previous Transition moves the main cursor to the previous transition.

Go To Next Transition moves the main cursor to next transition.

Adds a marker at the position of the main cursor to the Waveform area.

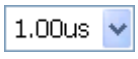
Moves the main cursor to the nearest marker to the left of the main cursor's current position.

Moves the main cursor to the nearest marker to the right of the main cursor's current position.

Resets the simulation time to zero.

Runs simulation until there are no more events, a stop command is issued or a break point is encountered.

Runs simulation for the amount of time specified (Run For).



Specifies the amount of time the simulation runs when the Run For button is clicked. The arrow enables you to select a previously used value from a list.

Steps through the simulation to the next line of HDL code.

Forces a running simulation to stop immediately. Simulation can be restarted using one of the run commands.

Ends the current simulation, leaving the simulation data open.

Shortcuts

Keyboard Shortcuts

Shortcut	Menu Command
F1	Help Topics (Help menu)
Shift+F1	What's This Help
F3	Find Next (Edit menu)
F5	Refresh (View menu)
F6	Zoom Full View (View menu)
F7	Zoom Out (View menu)
F8	Zoom In (View menu)
F11	Step (Simulation menu)
Delete	Delete (Edit menu)
Ctrl+N	New (File menu)
Ctrl+O	Open (File menu)
Ctrl+S	Save (File menu)
Ctrl+P	Print (File menu)
Ctrl+Z	Undo (Edit menu)
Ctrl+Y	Redo (Edit menu)
Ctrl+X	Cut (Edit menu)
Ctrl+C	Copy (Edit menu)
Ctrl+V	Paste (Edit menu)
Ctrl+F	Find (Edit menu)
Ctrl+G	Go To (Edit menu)
Ctrl+H	Replace (Edit menu)
Ctrl+A	Select All (Edit menu)
Ctrl+Tab	Next (Window menu)
Ctrl+Shift+Tab	Previous (Window menu)
Ctrl+ Mouse Wheel	Zooms in and out
Shift+ Mouse Wheel	Zooms left and right
Mouse Wheel	Scrolls up and down

ISim Preferences

Setting ISim Preferences

The preference settings enable you to view and change the settings for ISim.

To Set Preferences

1. Select **Edit > Preferences**.

2. In the left pane of the Preferences dialog box, click a category to view.
   - ISE Text Editor
   - ISim Simulator
3. Make the necessary setting changes.
4. Click the **Apply** button after changes are made.
5. When done viewing and editing the preferences, click **OK**.

The Preference settings are saved and are effective immediately in your session of ISim.

ISE Text Editor Preferences

The preference setting associated with ISE Text Editor controls the behavior of HDL files open in ISim only. For more information about the preference settings, see the ISE Text Editor Help.

ISim Simulator Preferences

Use the ISim Properties page to set some basic ISim preferences. To access this page, select **Edit > Preferences**, and select **ISim Simulator** in the left pane.

Draw Waveform Shadow

Shows/hides the shadow background for signals in the Wave window.

Limit the maximum number of elements display to

Sets limit for number of children elements to display for objects in the Object window. See Showing Display Elements.

Default Radix

Sets the default radix value displayed in the wave configuration, the Objects panel, and the Console panel. See Changing the Radix.

Console text font

The window to the right of the field shows example text for the specified font. Clicking the Change button opens a dialog box that enables you to specify the font used in the Console tab.

ISim Color Preferences

Use the Colors page to set your color preferences for displaying the waveform. For more information about changing colors, see Creating a Custom Color Scheme.

Click **Apply** to apply all changes.

Current Color Scheme

Displays the default color scheme and any custom schemes you have created.

New

Click this button to create a new scheme. Enter the new name in the Current Color Scheme field and edit the colors in the scheme table.

Delete

Click this button to delete the custom scheme that you have selected.

You can edit the color of this scheme

This table contains the color selection for each customizable item in the ISim main window. Click **Apply** to apply changes immediately.

Chapter 3

VHDL Simulation

VHDL Simulation Overview

Running a simulation from the command line involves three basic steps:

1. Parsing the design files
2. Generating the ISim simulation executable
3. Simulating the design

Go to the appropriate simulation topic for details:

- Running a Functional Simulation at the Command Line
- Running a Timing Simulation at the Command Line

Running a Functional Simulation of a VHDL Design From the Command Line

Method 1: Using a Project File - Recommended

Parsing the Design Files

Create a file called `<proj_name>.prj` and use the following syntax inside the project file:

```
vhdl <library_name> <file_name_1>.vhd
vhdl <library_name> <file_name_2>.vhd
.
vhdl <library_name> <file_name_n>.vhd
```

where:

- *<library_name>* indicates the library into which the source on the given line should be compiled. *<library_name>* is optional and is only needed when using a different library than the default `work`.
- *<file_name_1>.vhd* is the source file. Each line can contain only one VHDL source file.

Example of file called `project_name.prj`:

```
vhdl work top.vhd
vhdl mylib_for_testbench testbench.vhd
```

Note The top level file `testbench.vhd` contains the entity named testbench.

Generating the ISim Simulation Executable - Running fuse

The HDL linker, fuse, performs static elaboration of a design in terms of parsed nodes, generates object code for each unique module instance, and links the generated object codes with the ISim simulation engine library to create an ISim simulation executable.

Syntax:

fuse {[*<library_name>*.]*<top_name>*}**-prj** *<prj_file_name>.prj* **-o** *<output_file_name>*

where:

- *{[<library_name>.]<top_name>}* is one or more libraries and top level design unit names. *<library_name>* is optional and is only needed when using a different library than the default `work`. If specified, the HDL file that contains the top level has to be compiled in the related library name as referred in the project file. For example, design unit name of the test bench file.
- **-o** switch is optional for defining the isim executable name. Without **-o**, the default name for the isim executable is `x.exe`.

Example fuse command with project file:

```
fuse mylib_for_testbench.testbench -prj project_name.prj
```

See fuse Overview and Syntax for more information about the fuse command.

Method 2: Parsing the Files Using vhpcomp

Parsing the Design Files

Syntax:

vhpcomp [**-work**] *<library_name>* *<file_name>*.vhd

where

- **-work** is optional and is only needed when trying to specify a different library than the default *work*.
- *<library_name>* indicates the library into which the source specified by*<file_name>* should be compiled. There can be multiple VHDL file names per line.

Example:

```
vhpcomp suba.vhd subb.vhd
```

See vhpcomp Overview and Syntax for more information about the command.

Generation of the ISim Simulation Executable - Running fuse

Syntax:

fuse {[*<library_name>*.]*<top_name>*} **-o** *<output_file_name>*

where:

- *{[<library_name>.]<top_name>}* is one or more libraries and top level design unit names. *<library_name>* is optional and is only needed when using a different library than the default `work`. If specified, the HDL file that contains the top level has to be compiled in the related library name as referred in the project file. For example, design unit name of the test bench file.

  Note Using `glbl` as *top_name* is mandatory if behavioral design instantiates UNISIM primitives.
- **-o** switch enables you to define the simulation executable name (for example, **my_sim.exe**). Without **-o**, the default name for the simulation executable is `x.exe`.

Example:

```
fuse work.topunit work.glbl -o my_sim.exe
```

See fuse Overview and Syntax for more information about the fuse command.

Simulation

After the compilation and the ISim simulation executable generation, the final stage is simulation. To run the simulation, run the executable file generated by fuse.

Command	Behavior
x.exe (default name) or user-defined executable, such as **my_sim.exe**	Design is simulated, when complete, Tcl prompt opens for Tcl command entry.
x.exe -gui or **my_sim.exe -gui**	Design is simulated, when complete, ISim GUI is launched. GUI Commands and Tcl commands can be used to analyze design, rerun simulation, etc.
x.exe -tclbatch <tcl_file_name> or **my_sim.exe -tclbatch <tcl_file_name>**	Design is simulated, and Tcl commands specified in Tcl file are run, with the final command being **quit** command.

See ISim Simulation Executable Overview and Syntax for more information about the command.

Running a Timing Simulation From the Command Line With a VHDL Design

Generating Timing Simulation Model

Before launching a timing simulation, a timing simulation model and delay file for back-annotation are required. Use the NetGen tool to generate these files. See "Generating Gate-Level Netlist (Running NetGen)" in the *Synthesis and Simulation Design Guide*.

Method 1: Using a Project File - Recommended

Compilation of the Files

Create a file called `<proj_name>.prj` and use the following syntax inside the project file:

```
vhdl <library_name> <your_testbench>.vhd
vhdl <library_name> <topleve_timesim>.vhd
.
vhdl <library_name> <file_name_n>.vhd
```

- *<library_name>* indicates the library that the source on the given line should be compiled. The default library name is *work*.
- *<your_testbench>.vhd* is the stimulus file.
- *<topleve_timesim>.vhd* is the timing simulation model generated by NetGen (see "Generating Timing Simulation Model").
- *<file_name_n>.vhd* are any additional source files required by the testbench (auxiliary testbench files, etc.)

Generation of the ISim Simulation Executable - Running fuse

The HDL linker, fuse, performs static elaboration of a design in terms of parsed nodes, generates object code for each unique module instance, and links the generated object codes with the ISim simulation engine library to create an ISim simulation executable.

Syntax:

```
fuse {[<library_name>.]<top_name>}-prj <prj_file_name>.prj  -o <output_file_name>
```

where:

- *{[<library_name>.]<top_name>}* is one or more libraries and top level design unit names. The library name is optional and if not specified it is defaulted to `work`. If specified, the HDL file that contains the top level has to be compiled in the related library name as referred in the project file. For example, design unit name of the test bench file.

  Note Using `glbl` as *top_name* is necessary.

- **`-prj`** *<proj_file_name>.prj* is optional in fuse during timing simulation.
- **`-o`** switch is optional. Without **`-o`**, the default name for the simulation executable is `x.exe`.

```
fuse topunit work.glbl -prj mydesign.prj -o my_sim.exe
```

See fuse Overview and Syntax for more information about the fuse command.

Method 2: Using vhpcomp

Parsing of the Files

Syntax:

`vhpcomp` `[`**`-work`** *<library_name>* `]`*<file_name>*`.vhd`

`vhpcomp -work` *<library_name>*<*file_name*>`.vhd`

where

- **`-work`** is optional and is only needed when trying to specify a different library than the default `work`.
- *<library_name>* indicates the library into which the source specified by*<file_name>* should be compiled. There can be multiple VHDL file names per line.

See vhpcomp Overview and Syntax for more information about the command.

Generation of the ISim Simulation Executable - Running fuse

Syntax:

`fuse` `{[`*<library_name>.* `]<top_name> }`**`-o`** *<output_file_name>*

where:

- *{[<library_name>.]<top_name>}* is one or more libraries and top level design unit names. For example, design unit name of the test bench file. Include a library name is optional. The default name work is assumed when library name is not specified. Examples: `work.topunit, work.glbl, mylib.glbl`.
- **`-o`** switch is optional. Without **`-o`**, the default name for the simulation executable is `x.exe`.

See fuse Overview and Syntax for more information about the fuse command.

Simulation

After the compilation and ISim simulation executable generation, the final stage is simulation. The ISim simulation executable generated by the fuse command is run to effect simulation.

If you would like Tcl commands contained in a file to be executed after the simulation has begun, use the **`-tclbatch`** switch.

It is also possible to instruct the simulator to use the correct timing delays from the SDF file.

Syntax:

```
<executable_name>.exe -tclbatch <tcl_file_name> -sdfmin|-sdftyp|-sdfmax
<<anno_point>=sdf_file>.sdf>
```

where

- *<executable_name>.exe* is the simulation executable called x.exe unless otherwise specified with the **`fuse -o`** switch.
- **`-sdfmin|-sdftyp|-sdfmax`** is the type of delay (minimum, typical, or maximum) that ISim should use.
- *<anno_point>* is the point of hierarchy where you want to annotate the delay. This is usually UUT.
- *<sdf_file>* is the filename of the SDF file you want to annotate.

See ISim Simulation Executable Overview and Syntax for more information about the command.

Library Mapping File

Note The following information is intended for advanced users.

The ISim HDL compile programs, vhpcomp, vlogcomp and fuse, use the `xilinxsim.ini` configuration file to learn the definitions and physical locations of VHDL and Verilog logical libraries.

Search Order

The compilers attempt to read `xilinxsim.ini` from these locations in the following order.

1. `$XILINX/vhdl/hdp/<platform>`.
2. User file specified through the **`-initfile`** switch in **`vlogcomp`**, **`vhpcomp`** or **`fuse`**. If **`-initfile`** is not specified, "`xilinxsim.ini`" in the current working directory is searched for.

Syntax

The `xilinxsim.ini` file has the following format:

```
<logical_library1> = <physical_dir_path1>
<logical_library2> = <physical_dir_path2>
.
.
<logical_libraryn> = <physical_dir_pathn>
```

Example

The following is an example of a `xilinxsim.ini` file:

VHDL

```
std=C:/libs/vhdl/hdp/
stdieee=C:/libs/vhdl/hdp/ieee
work=C:/work
```

Verilog

```
unisims_ver=$XILINX/rtf/verilog/hdp/nt/unisims_ver
xilinxcorelib_ver=C:/libs/verilog/hdp/nt/xilinxcorelib_ver
mylib=./mylib
work=C:/work
```

Features/Limitations

The `xilinxsim.ini` file has the following features and limitations:

- There must be no more than one library/path per line specified inside the `xilinxsim.ini` file.
- If the directory corresponding to the physical path does not exist, **`vhpcomp`** or **`vlogcomp`** creates it when the compiler first tries to write to it.
- You can describe the physical path in terms of environment variables. The environment variable must start with $ character.
- The default physical directory for a logical library is `isim/<logical_library_name>`.
- All comments in this file must start with '--'.

Interactive Simulation in Command Line Mode

When a simulation is run in command line mode, a Tcl prompt opens and you can enter simulation Tcl commands, which enable you to run simulation, analyze the design, and debug the design. For more information about Simulation Commands, see Simulation Command Overview. For tips on how to enter commands, see Entering Simulation Tcl Commands.

XILINX®

Chapter 4

Verilog Simulation

Verilog Simulation Overview

Running a simulation from the command line involves three basic steps:

1. Parsing the design files
2. Generating the ISim simulation executable
3. Simulating the design

Go to the appropriate simulation topic for details:

- Running a Functional Simulation at the Command Line
- Running a Timing Simulation at the Command Line

Running a Functional Simulation of a Verilog Design From the Command Line

In a functional (behavioral) simulation of a Verilog design, the steps must be followed if UNISIM primitives are used:

- Compile `$XILINX/Verilog/src/glbl.v` to library `work`.
- Specify **`work.glbl`** as one of the *<library_name>.<top_name>* in **`fuse`**.
- Specify **`-L unisims_ver`** in **`fuse`**.

Method 1: Using a Project File - Recommended

Parsing the Design Files

Create a file called `<proj_name>.prj` and use the following syntax inside the project file:

```
verilog <library_name> {<file_name_1>.v} {[-d <macro>] [-i <include_path>]}
verilog <library_name> {<file_name_2>.v} {[-d <macro>] [-i <include_path>]}
.
verilog <library_name> {<file_name_n>.v} {[-d <macro>] [-i <include_path>]}
```

where:

- *verilog* indicates that the source is a Verilog file.
- *<library_name>* indicates the library into which the source on the given line should be compiled. More than one Verilog source can be specified on a given line.
- *[-d <macro>]* defines one or more macro located in the location specified in *[-i <include_path>]*.

Example:

```
verilog work top.v testbench.v
```

Generating the ISim Simulation Executable - Running fuse

The HDL linker, fuse, performs static elaboration of a design in terms of parsed nodes, generates object code for each unique module instance, and links the generated object codes with the ISim simulation engine library to create an ISim simulation executable.

Syntax:

```
fuse {[<library_name>.]<top_name>}-prj <prj_file_name>.prj  -L <Verilog_library>
-o <output_file_name>
```

where:

- *{[<library_name>.]<top_name>}* is one or more libraries and top level design unit names. The library name is optional and if not specified it is defaulted to `work`. If specified, the HDL file that contains the top level has to be compiled in the related library name as referred in the project file. For example, design unit name of the test bench file.

  Note Using `glbl` as *top_name* is mandatory if behavioral design instantiates UNISIM primitives.

- **`-L`** *<Verilog_library>* must include **`unisims_ver`** if behavioral simulation relies on UNISIM primitives, in addition to other Xilinx libraries that apply to the design, such as **`unimacro_ver`** and **`xilinxcorelib_ver`**.
- **`-o`** switch is optional. Without **`-o`**, the default name for the simulation executable is `x.exe`.

Example:

```
fuse work.test_bench work.glbl -prj mydesign.prj -L unisims_ver -L unimacro_ver
-L xilinxcorelib_ver -o test_bench.exe
```

See fuse Overview and Syntax for more information about the fuse command.

Method 2: Parsing the Files Using vlogcomp

Parsing the Design Files

Syntax:

```
vlogcomp [-work <library_name>] <file_name>.v {[-d <macro>] [-i <include_path>]
```

where

- **`-work`** is optional and is only needed when trying to specify a different library than the default `work`.
- *<library_name>* indicates the library into which the source specified by*<file_name>*should be compiled. More than one Verilog source can be specified on a given line.
- **`-d`***<macro>* defines one or more macro located in the location specified in *-i <include_path>*.

Example:

```
vlogcomp suba.v subb.v
```

See vlogcomp Overview and Syntax for more information about the vlogcomp command.

Generation of the ISim Simulation Executable - Running fuse

Syntax:

```
fuse {[<library_name>.]<top_name>} {-L <Verilog_library>} -o <output_file_name>
```

where:

- *{[<library_name>.]<top_name>}* is one or more top level of design unit names can be specified. For example, design unit name of the test bench file. One top name must be `glbl`. Including a library name is optional. The default library name `work` is assumed when a library name is not specified.

  Note For more information about the global module, see "Global Set/Reset (GSR) and Global Tristate (GTS) in Verilog" in the *Synthesis and Simulation Design Guide*.

- **`-L`** *<Verilog_library>* must include **`unisim`** in addition to other Xilinx libraries that apply to the design, such as **`unimacro`** and **`xilinxcorelib`**.
- **`-o`** switch is optional. Without **`-o`**, the default name for the simulation executable is `x.exe`.

Example:

```
fuse work.test_bench work.glbl -L unisims_ver -L unimacro_ver -L xilinxcorelib_ver
-o test_bench.exe
```

See fuse Overview and Syntax for more information about the fuse command.

Simulation

After the compilation and the ISim simulation executable generation, the final stage is simulation. To run the simulation, run the executable file generated by fuse.

Command	Behavior
x.exe (default name) or user-defined executable, such as **my_sim.exe**	Design is simulated, when complete, Tcl prompt opens for Tcl command entry.
x.exe -gui or **my_sim.exe -gui**	Design is simulated, when complete, ISim GUI is launched. GUI Commands and Tcl commands can be used to analyze design, rerun simulation, etc.
x.exe -tclbatch <tcl_file_name> or **my_sim.exe -tclbatch <tcl_file_name>**	Design is simulated, and Tcl commands specified in Tcl file are run, with the final command being **quit** command.

See ISim Simulation Executable Overview and Syntax for more information about the command.

Running a Timing Simulation of a Verilog Design From the Command Line

In a timing simulation of a Verilog design, the following rules must be followed:

- Compile `$XILINX/Verilog/src/glbl.v` to library `work`.
- Specify **`work.glbl`** as one of the *<library_name>.<top_name>* in **`fuse`**.
- Specify **`-L simprims_ver`** in **`fuse`**.

Generating Timing Simulation Model

Before launching a timing simulation, a timing simulation model and delay file for back-annotation are required. Use the NetGen tool to generate these files. See "Generating Gate-Level Netlist (Running NetGen)" in the *Synthesis and Simulation Design Guide*.

Method 1: Using a Project File - Recommended

Compilation of the Files

Create a file called `<proj_name>.prj` and use the following syntax inside the project file:

```
verilog <library_name> {<file_name>.v} {[-d <macro>] [-i <include_path>]}
```

where

- *verilog* indicates that the source is a Verilog file. More than one Verilog source can be specified on a given line.
- *<library_name>* is the target library into which the source(s) on that particular line should be compiled.
- *[-d <macro>]* allows you to define one or more macro of the path specified by *[-i <include_path>]*. These options are optional.

Example 1:

```
verilog work top.v testbench.v
verilog work glbl.v
verilog work top_timesim.v
```

Generation of the ISim Simulation Executable - Running fuse

The HDL linker, fuse, performs static elaboration of a design in terms of parsed nodes, generates object code for each unique module instance, and links the generated object codes with the ISim simulation engine library to create an ISim simulation executable.

Syntax:

```
fuse {[<library_name>.]<top_name>} -prj <prj_file_name>.prj {-L
<verilog_library_name> }-o <output_file_name>
```

where:

- *{[<library_name>.]<top_name>}* is one or more top level of design unit names can be specified. For example, design unit name of the test bench file. One top name must be `glbl`. Including a library name is optional. The default library name `work` is assumed when a library name is not specified.
- **`-L`** *<Verilog_library>* must include **`simsprims_ver`** in addition to other Xilinx libraries that apply to the design.
- **`-o`** switch is optional. Without **`-o`**, the default name for the simulation executable is `x.exe`.

Example:

```
fuse work.testbench work.glbl -prj design.prj -L simprims_ver -o isim.exe
```

See fuse Overview and Syntax for more information about the fuse command.

Method 2: Parsing the Files Using vlogcomp

Parsing the Files

Syntax:

```
vlogcomp [-work <library_name>]<file_name>.v
```

where

- **`-work`** is optional and is only needed when trying to specify a different library than the default `work`.
- *<library_name>* indicates the library into which the source specified by*<file_name>* should be compiled. More than one Verilog source can be specified on a given line.

Example:

```
vlogcomp top_testbench.v top_timesim.v
```

See vlogcomp Overview and Syntax for more information about the vlogcomp command.

Generation of the ISim Simulation Executable - Running fuse

Syntax:

```
fuse {[<library_name>.]<top_name>} {-L <Verilog_library>} -o <output_file_name>
```

where:

- *{[<library_name>.]<top_name>}* is one or more top level of design unit names can be specified. For example, design unit name of the test bench file. One top name must be `glbl`. Including a library name is optional. The default library name `work` is assumed when a library name is not specified.
- **`-L`** *<Verilog_library>* must include **`simsprims_ver`** in addition to other Xilinx libraries that apply to the design.
- **`-o`** switch is optional. Without **`-o`**, the default name for the simulation executable is `x.exe`.

Example:

```
fuse work.textbench work.glbl -L simprims_ver -o timesim.exe
```

See fuse Overview and Syntax for more information about the fuse command.

Simulation

After the compilation and ISim simulation executable generation, the final stage is simulation. The ISim simulation executable generated by the fuse command is run to effect simulation.

If you would like Tcl commands contained in a file to be executed after the simulation has begun, use the **-tclbatch** switch.

It is also possible to instruct the simulator to use the correct timing delays from the SDF file.

Syntax:

```
<executable_name>.exe -tclbatch <tcl_file_name> -sdfmin|-sdftyp|-sdfmax
<<anno_point>=sdf_file>.sdf>
```

where

- *<executable_name>.exe* is the simulation executable called x.exe unless otherwise specified with the **fuse -o** switch.
- **-sdfmin|-sdftyp|-sdfmax** is the type of delay (minimum, typical, or maximum) that ISim should use.
- *<anno_point>* is the point of hierarchy where you want to annotate the delay. This is usually UUT.
- *<sdf_file>* is the filename of the SDF file you want to annotate.

See ISim Simulation Executable Overview and Syntax for more information about the command.

Search Order for Instance of Verilog Design Units

The HDL linker, fuse, uses the following search order in order to search and bind instantiated Verilog Design Units in a design.

1. Library specified by ‘uselib directive.
2. Libraries specified on the command line with **-lib|-L** switch.
3. Library of the parent design unit.
4. Logical work library.

Supporting Source Libraries

The compiler arguments listed below support source libraries in the same manner as Verilog-XL. See the vlogcomp Options or the fuse Options for a description of each argument.

In order to use this feature, the following command options will need to be passed to the vlogcomp Verilog compiler.

Library Location (-sourcelibdir)

Note **-sourcelibdir** provides functionality that is similar to the **-y** switch in Verilog-XL.

After the source files on the command line have been compiled, if there are any unresolved references to modules, the compiler searches the source libraries for resolution. During this search, the compiler attempts to match the *name* of any unresolved instantiated design unit with a file of the *same name* in the specified **-sourcelibdir** directory. If such a file exists, compiler analyzes that file. Also note that by default the compiler ignores any files with extensions such as .v, .h, etc, unless **-sourcelibext** is also used.

```
-sourcelibdir <library_first> -sourcelibdir <library_second>
```

Source File Extension (-sourcelibext)

Note **-sourcelibext** provides functionality that is similar to the **+libext+** switch in Verilog-XL.

This command line argument may be used in conjunction with **-sourcelibdir** when the source library files have extensions.

```
-sourcelibext .v
```

Source File (-sourcelibfile)

Note **-sourcelibfile** provides functionality similar to the **-v** switch in Verilog-XL.

ISim also enables you to provide a source Verilog library file that contains definitions of all the unresolved modules.

```
-sourcelibfile ./library/lib_abc.v
```

Examples

The following examples demonstrate the use of these command options.

Vlogcomp

```
vlogcomp -work mywork1 file1.v -sourcelibdir mydir/cells
```

The compiler searches for unresolved cells inside directory `mydir/cells`. For example, if `file1.v` instantiates DFF and DMUX, which are unresolved, then the compiler would look for files with names DFF and DMUX inside directory `mydir/cells`. Files DFF and DMUX should define modules DFF and DMUX.

Fuse

```
fuse –prj test.prj test -sourcelibfile ./mylib1/lib_abc.v -sourcelibfile
./mylib1/lib_cde.v
```

where `test.prj` contains:

```
verilog work test.v
```

The compiler uses files given by the **-sourcelibfile** options for modules used in `test.v`. It analyzes the modules and elaborates the `test` design

Project File

```
fuse –prj test.prj test
```

where `test.prj` contains:

```
verilog work test.v -sourcelibdir ./mylib1 –sourcelibdir ./mylib2 -sourcelibext .v
```

For every unresolved module with name *modulename* instantiated in file `test.v`, the compiler looks up files with name `modulename.v` inside the directories `./mylib1` and `./mylib2`, in that order.

Library Mapping File

Note The following information is intended for advanced users.

The ISim HDL compile programs, vhpcomp, vlogcomp and fuse, use the `xilinxsim.ini` configuration file to learn the definitions and physical locations of VHDL and Verilog logical libraries.

Search Order

The compilers attempt to read `xilinxsim.ini` from these locations in the following order.

1. `$XILINX/vhdl/hdp/<platform>`.
2. User file specified through the **`-initfile`** switch in **`vlogcomp`**, **`vhpcomp`** or **`fuse`**. If **`-initfile`** is not specified, "`xilinxsim.ini`" in the current working directory is searched for.

Syntax

The `xilinxsim.ini` file has the following format:

```
<logical_library1> = <physical_dir_path1>
<logical_library2> = <physical_dir_path2>
.
.
<logical_libraryn> = <physical_dir_pathn>
```

Example

The following is an example of a `xilinxsim.ini` file:

VHDL

```
std=C:/libs/vhdl/hdp/
stdieee=C:/libs/vhdl/hdp/ieee
work=C:/work
```

Verilog

```
unisims_ver=$XILINX/rtf/verilog/hdp/nt/unisims_ver
xilinxcorelib_ver=C:/libs/verilog/hdp/nt/xilinxcorelib_ver
mylib=./mylib
work=C:/work
```

Features/Limitations

The `xilinxsim.ini` file has the following features and limitations:

- There must be no more than one library/path per line specified inside the `xilinxsim.ini` file.
- If the directory corresponding to the physical path does not exist, **`vhpcomp`** or **`vlogcomp`** creates it when the compiler first tries to write to it.
- You can describe the physical path in terms of environment variables. The environment variable must start with $ character.
- The default physical directory for a logical library is `isim/<logical_library_name>`.
- All comments in this file must start with '--'.

Predefined XILINX_SIM Macro for Verilog Simulation

XILINX_ISIM is a Verilog predefined macro specific to ISim, and the value of this macro is '1'. These predefined macros are used to perform tool specific functions or sometimes just to identify which tool to use in a design flow.

```
module  isim_predefined_macro;
integer fp;
initial
begin
‘ifdef XILINX_ISIM
    $display("XILINX_ISIM defined");
    fp = $fopen("ISIM.dat");
‘else
    $display("XILINX_ISIM not defined");
    fp = $fopen("other.dat");
‘endif
$fdisplay (fp, "results");
end
endmodule
```

Interactive Simulation in Command Line Mode

When a simulation is run in command line mode, a Tcl prompt opens and you can enter simulation Tcl commands, which enable you to run simulation, analyze the design, and debug the design. For more information about Simulation Commands, see Simulation Command Overview. For tips on how to enter commands, see Entering Simulation Tcl Commands.

XILINX®

Chapter 5

Mixed Language Simulation

Mixed Language Simulation Overview

Note The following information is intended for advanced users.

ISim supports mixed language project files and mixed language simulation. This enables you to include Verilog modules in a VHDL design, and vice versa. Some restrictions do apply:

Restrictions on Mixed Language in Simulation

- Mixing VHDL and Verilog is restricted to the module instance or component only. A VHDL design can instantiate Verilog modules and a Verilog design can instantiate VHDL components. Any other mix use of VHDL and Verilog is not supported.
- A Verilog hierarchical reference cannot refer to a VHDL unit nor can a VHDL expanded/selected name refer to a Verilog unit.
- Only a small subset of VHDL types, generics and ports are allowed on the boundary to a Verilog module. Similarly, a small subset of Verilog types, parameters and ports are allowed on the boundary to VHDL design unit.
- Component instantiation-based default binding is used for binding a Verilog module to a VHDL design unit. Specifically, configuration specification, direct instantiation and component configurations are not supported for a Verilog module instantiated inside a VHDL design unit. More Info

Key Steps in a Mixed Language Simulation

- Instantiate mixed language components. See Instantiating a Verilog Module in a VHDL Design Unit or Instantiating a VHDL Module in a Verilog Design Unit
- Optionally, specify the search order for VHDL entity or Verilog modules in the design libraries of a mixed language project.

  Use the **`fuse -L`** option to specify the binding order of a VHDL entity or a Verilog module in the design libraries of a mixed language project. Note that the library search order specified by **`-L`** is used for binding Verilog modules to other Verilog modules as well. More Info
- Run the simulation.

Instantiating Mixed Language Components

Instantiating a Verilog Module in a VHDL Design Unit

In a mixed language design, you can instantiate a Verilog module in a VHDL design unit.

To Instantiate a Verilog Module in a VHDL Design Unit

1. Declare a VHDL component with the same name as the Verilog module (respecting case sensitivity) that you want to instantiate.

   For example,

```
COMPONENT MY_VHDL_UNIT PORT (
Q : out  STD_ULOGIC;
D : in   STD_ULOGIC;
C : in   STD_ULOGIC );
END COMPONENT;
```

2. Use named association to instantiate the Verilog module.

   For example,

```
UUT : MY_VHDL_UNIT PORT MAP(
Q => O,
D => I,
C => CLK);
```

To ensure that you are correctly matching port types, see port mapping rules in Mixed Language Boundary and Mapping Rules.

Since Verilog is case sensitive, named associations and the local port names that you use in the component declaration *must* match the case of the corresponding Verilog port names.

Instantiating a VHDL Module in a Verilog Design Unit

In a mixed language design, you can instantiate a VHDL module in a Verilog design unit.

To Instantiate a VHDL Module in a Verilog Design Unit

Instantiate the VHDL entity as if it were a Verilog module.

For example,

```
module testbench ;
wire in, clk;
wire out;
FD FD1(
.Q(Q_OUT),
.C(CLK);
.D(A);
);
```

Mixed Language Binding and Searching

Note The following information is intended for advanced users.

When you instantiate a VHDL component or a Verilog module, the fuse linker first searches for a unit of the same language as that of the instantiating design unit. If a unit of the same language is not found, fuse searches for a cross language design unit in the libraries specified in the **`-lib`** option. The search order is the same as the order of appearance of libraries on fuse command line. For more information about Verilog library search order, see Search Order for Instance of Verilog Design Units.

Note When using the ISE® software, the library search order is specified automatically. No user intervention is necessary or possible.

VHDL Instantiation Unit

When a VHDL design instantiates a component, the fuse linker treats the component name as a VHDL unit and searches for it in logical library "*work*". If a VHDL unit is found, fuse binds it and the search stops. If fuse cannot find a VHDL unit, it treats the case preserved component name as a Verilog module name and continues to search as follows :

- Performs a case sensitive search for a Verilog module in the user specified list of unified logical libraries in the user specified search order. The first one matching the name is picked and the search stops.
- If case sensitive search is not successful, performs a case insensitive search for a Verilog module in the user specified list of unified logical libraries in the user specified search order. If a unique binding is found for any one library, the search stops.

Verilog Instantiation Unit

When a Verilog design instantiates a component, the fuse linker treats the component name as a Verilog unit and searches for a Verilog module in the user specified list of unified logical libraries in the user specified order. If found, fuse binds it and the search stops. If fuse cannot find a Verilog unit, it treats the name of the instantiated module as a VHDL entity name and continues the search as follows:

- Performs a case insensitive search for an entity with the same name as the instantiated module name in the user specified list of unified logical libraries in the user specified order. The first one matching the name is picked and the search stops.
- Performs a case sensitive search for a VHDL design unit name constructed as an extended identifier in the user specified list of unified logical libraries in the user specified order. If a unique binding is found for any one library, the name is picked and the search stops.

Note For a mixed language design, the port names used in a named association to a VHDL entity instantiated by a Verilog module are always treated as case insensitive. Also note that you cannot use a defparam statement to modify a VHDL generic.

Mixed Language Boundary and Mapping Rules

Note The following information is intended for advanced users.

General

The following restrictions apply to the boundaries between VHDL and Verilog design units/modules.

- The boundary between VHDL and Verilog is enforced at design unit level.
- A VHDL design is allowed to instantiate one or more Verilog modules.
- Instantiation of a Verilog UDP inside a VHDL design is not supported.
- A Verilog design can instantiate a VHDL component corresponding to a VHDL entity only. Instantiation of a VHDL configuration in a Verilog design is not supported.

Port Mapping

The following rules and limitations for port mapping are used in mixed language projects.

- Supported VHDL port types:
  - IN
  - OUT
  - INOUT

  Note Buffer and linkage ports of VHDL are not supported.
- Supported Verilog port types:
  - INPUT
  - OUTPUT
  - INOUT

  Note Connection to bi-directional pass switches in Verilog are not supported.
- Unnamed Verilog ports are not allowed on mixed design boundary.
- The following table shows supported VHDL and Verilog data types for ports on the mixed language design boundary:

VHDL Port	Verilog Port
bit	net
std_ulogic	net
std_logic	net
bit_vector	vector net
std_ulogic_vector	vector net
std_logic_vector	vector net

Note Verilog output port of type reg is supported on the mixed language boundary. On the boundary, an output reg port is treated as if it were an output net (wire) port.

Note Any other type found on mixed language boundary is considered an error.

Generics (Parameters) Mapping

Following VHDL generic types (and their Verilog equivalents) are supported.

- integer
- real
- string
- boolean

Note Any other generic type found on mixed language boundary is considered an error.

VHDL/Verilog Values Mapping

Verilog states are mapped to std_logic and bit as shown in the following table.

Verilog	std_logic	bit
Z	'Z'	'0'
0	'0'	'0'
1	'1'	'1'
X	'X'	'0'

Note Verilog strength is ignored. There is no corresponding mapping to strength in VHDL.

VHDL type bit is mapped to Verilog states in the following table.

bit	Verilog
'0'	0
'1'	1

VHDL type std_logic is mapped to Verilog states in the following table.

std_logic	Verilog
'U'	X
'X'	X
'0'	0
'1'	1
'Z'	Z
'W'	X
'L'	0
'H'	1
'-'	X

XILINX®

Chapter 6

Waveform Analysis

Before Analysis

Launching the ISim GUI

To Launch the GUI

The ISim GUI is launched when you run the simulation executable, either from the ISE® software, or from the command line. For details, see Step 3: Simulating the Design from the Steps in Simulation.

To launch the ISim graphic user interface in read-only mode in order to view or analyze the data from a previous simulation, see Opening a Static Simulation for more information.

What to Expect

Simulation from ISE software

When you launch ISim from the ISE® software, a wave configuration with top-level signals is displayed. Design data is populated in other areas of the GUI, such as the Objects panel, and the Instances and Processes panel. You can then proceed to add additional signals, or run the simulation in ISim.

Simulation at the Command Line

When you launch ISim from the command line by running the simulation executable with the `-gui` switch (Graphical User Interface Mode), an empty wave configuration is displayed. Design data is populated in other areas of the GUI, such as the Objects Panel, and the Instances and Processes panel. You must add signals to the wave configuration before you run the simulation in ISim.

Adding Signals to the Wave Configuration

You can populate the Wave window with the signals from your design using menu commands or drag and drop capabilities in the graphical user interface, or using Tcl commands in the Console panel.

Note Changes to the wave configuration, including creating the wave configuration or adding signals to it, do not become permanent until you save the WCFG file. For more information, see Saving the Results.

In the GUI

1. In the Instances and Processes panel, expand the design hierarchy, and select an item.

   The objects that correspond to the selected instance or process displays in the Objects panel.
2. In the Objects panel, select one or more objects.
3. Use one of the following methods to add objects to the Wave Configuration:
   - Right-click, and select **Add to Wave Window** from the context menu.
   - Drag and drop the objects from the Objects panel to the Name column of the Wave window.
   - In the Console, use the wave add command as described below.

Using Tcl

1. Although optional, you may wish to first identify the objects you want to add by exploring the design hierarchy in the Instances and Processes panel and Objects panel, as described above, or by entering the scope command in the Console panel.
2. In the Console panel, enter the wave add command to enter an individual object or a group of objects.

Adding Copies of Signals or Buses

You can add copies of the same signal/bus in a wave configuration for comparing waveforms. Copies of the same signal /bus can be placed anywhere in the wave configuration, such as in groups, or virtual buses.

To Add a Copy of a Signal or Bus

1. Select a signal or bus in the wave configuration in the Wave window.
2. Select **Edit > Copy** , or press **Ctrl+C**.

   The signal name is copied to the clipboard.
3. Select **Paste** command or **Ctrl+V**.

The signal/bus is now copied to the wave configuration. You can move the signal/bus using drag and drop as needed.

Running a Simulation in ISim

Simulation, the process of verifying the logic and timing of a design, can be run from ISim using functions in the interface or at the command line.

To Run a Simulation From the ISim GUI

The following GUI menu commands can be used to run simulation.

- **Simulation > Restart** - Stops simulation and sets simulation time back to 0. Use the Run All, Run For or Step command to run the simulation over again without reloading the design. See restart Tcl command.
- **Simulation > Run All** - Runs simulation until all events are executed. You can also use the Run Tcl command with the *all* option.
- **Simulation > Run** - Runs simulation for 100ns or for specified amount of time in the toolbar. Time and time unit are entered in the Value box in the toolbar. You can also use the run Tcl command with a length and unit specified.
- **Simulation > Step** - Runs simulation for one executable HDL instruction at a time. See Stepping a Simulation. See also step Tcl command.

In addition, you can run simulation until a specific point in your HDL source code is reached. To do so, use breakpoints and the Run All command. See Source Level Debugging Overview.

Note The current simulation time is displayed on the status bar in the lower right corner.

Stopping a Simulation

While running a simulation for any length of time, you can stop a simulation using the Break command, which leaves the simulation session open.

To close the session of ISim, see Closing ISim.

To Stop a Running Simulation

You can stop a running simulation using the Break command as follows:

- Select **Simulation > Break**.
- Click the Break toolbar button .
- Enter **Ctrl+C** at the command line *only*.

The simulator stops at the next executable HDL line. The line at which the simulation stopped is displayed in the text editor.

Note This behavior applies to designs that have not been compiled with the **`-nodebug`** switch.

The simulation can be resumed at any time by using the Run All, Run, or Step commands. See Running a Simulation in ISim for details.

Closing ISim

You can terminate a simulation and close the ISim session.

To Close ISim

- Select **File > Exit**.
- Enter the **`quit -f`** command in the Console panel at the prompt.
- Click the **X** at the top-right corner of the main window.

The simulation terminates and the session of ISim closes.

Customizing the Wave Configuration

Placing Cursors

The main cursor and secondary cursor in the Wave window enable you to display and measure time, and they form the focal point for various navigation activities.

To Place the Main Cursor

With a single click in the Wave window, the main cursor is placed at the location of the mouse click.

To Place the Secondary Cursor

Place the secondary cursor as follows:

- Click and hold the waveform, and drag either left or right.

  This sets the secondary cursor, with the initial click, and the main cursor, when you finish dragging.
- Select **Shift** and click the mouse in the waveform.

  If the secondary cursor is not already on, this action sets the secondary cursor to the present location of the main cursor and places the main cursor at the location of the mouse click.

  Note To preserve the location of the secondary cursor while positioning the main cursor, hold the **Shift** key while clicking.

Note When placing the secondary cursor by dragging, you must drag a minimum distance before the secondary cursor will appear.

To Move a Cursor

A cursor can be moved by hovering over the cursor until you see the grab symbol , and clicking and dragging the cursor to the new location.

As you drag the cursor in the Wave window, you will see a hollow or filled-in circle if the Snap to Transition toolbar button is selected, which is the default behavior.

- A filled-in circle indicates that you are hovering over the selected signal's waveform transition.
- A hollow circle indicates that you are hovering between transitions in the selected signal's waveform.

Setting Markers

Adding a Marker

You can add one or more markers to navigate through the waveform, and to display the waveform value at a particular time. Markers are added to the wave configuration at the location of the main cursor.

To Add a Marker

1. Place the main cursor at the time where you want to add the marker by clicking in the Wave window at the desired time or on the desired transition.
2. Select **Edit > Markers > Add Marker**, or click the Add Marker toolbar button .

A marker is placed at the cursor, or slightly offset if a marker already exists at the location of the cursor. The time of the marker is displayed at the top of the line.

Moving a Marker

After you add a marker, you can move the marker to another location in the waveform using the drag and drop method.

To Move a Marker

1. Click marker label (at the top of the marker) and drag it to the desired location.

   The drag symbol indicates that the marker can be moved around.

   As you drag the marker in the Wave window, you will see a hollow or filled-in circle if the Snap to Transition toolbar button is selected, which is the default behavior.

   - A filled-in circle indicates that you are hovering over a transition of the selected signal's waveform or over another marker. For markers, the filled-in circle is white.
   - A hollow circle indicates that you are hovering between transitions in the selected signal's waveform.
2. Release the mouse key to drop the marker to the new location.

The marker is moved to the new location.

Deleting a Marker

You can delete one or all markers with one command.

To Delete Markers

1. Right-click over a marker.
2. Perform one of the following functions:
   - Select **Delete Marker** from the context menu to delete a single marker.
   - Select **Delete All Markers** from the context menu to delete all markers.

Note The **Delete** key can also be used to delete a selected marker.

The marker or markers are removed from the waveform.

The undo command (**Edit > Undo**) can be used to reverse a marker deletion.

Adding Dividers

You can add a divider to your wave configuration to create a visual separator of signals.

To Add a Divider

1. In a Name column of the Wave window, click a signal below which you wish to add a divider.
2. Select **Edit > New Divider**, or right-click and select **New Divider** from the context menu.

A divider is added to the wave configuration. The change is visual and nothing is added to the HDL code.

The new divider will be saved with the wave configuration file when the file is saved next.

Following changes can be made to a divider:

- Dividers can be named. See Renaming Objects.
- Dividers can be moved to another location in the waveform, by dragging and dropping the divider name.

To delete a divider, highlight the divider, and click the **Delete** key, or right-click and select **Delete** from the context menu.

Adding Groups

You can add a group to your wave configuration, which is a collection, or "folder", to which signals and buses can be added in the wave configuration as a means of organizing a set of related signals. The group itself displays no waveform data but can be expanded to show its contents or collapsed to hide them.

To Add a Group

1. In a wave configuration, select one or more signals or buses you wish to add to a group.

   Note A group can also comprise dividers, virtual buses, and other groups.
2. Select **Edit > New Group**, or right-click and select **New Group** from the context menu.

A group that contains the selected signal or bus is added to the wave configuration. A group is represented with the icon. The change is visual and nothing is added to the HDL code.

You can move other signals or buses to the group by dragging and dropping the signal/bus name.

The new group and its nested signals/buses will be saved the next time you save the wave configuration file.

Following changes can be made to a group:

- Groups can be renamed. See Renaming Objects.
- Groups can be moved to another location in the Name column by dragging and dropping the group name.

To remove a group, highlight a group and select **Edit > Wave Objects > Ungroup**, or right-click and select **Ungroup** from the context menu. Signals/buses formerly in the group will be placed at the top level in the wave configuration hierarchy.

Caution! The **Delete** key will remove the group *and* its nested signals and buses from the wave configuration.

Adding Virtual Buses

You can add a virtual bus to your wave configuration, which is a grouping, or "folder", to which logic scalars and arrays can be added. The virtual bus displays a bus waveform, which is comprised of the signal waveforms in the vertical order that they appear under the virtual bus, flattened to a one-dimensional array.

To Add a Virtual Bus

1. In a wave configuration, select one or more signals or buses you wish to add to a virtual bus.
2. Select **Edit > New Virtual Bus**, or right-click and select **New Virtual Bus** from the context menu.

A virtual bus, which contains the selected signal(s) or bus(es), is added to the wave configuration. A virtual bus is represented with the icon. The change is visual and nothing is added to the HDL code.

You can move other signals or buses to the virtual bus by dragging and dropping the signal/bus name.

The new virtual bus and its nested signals/buses will be saved the next time you save the wave configuration file.

Following changes can be made to a virtual bus:

- Virtual buses can be renamed. See Renaming Objects.
- Move the virtual bus to another location in the waveform by dragging and dropping the virtual bus name.

To remove a virtual bus, and ungroup its contents, highlight the virtual bus, and select **Edit > Wave Objects > Ungroup**, or right-click and select **Ungroup** from the context menu.

Caution! The **Delete** key will remove the virtual bus *and* its nested signals and buses from the wave configuration.

Renaming Objects

You can rename any object in the Wave window, such as signals, dividers, groups, and virtual buses.

To Rename an Object

1. Select the object name in the Name column.
2. Right click, and select **Rename** from the context menu.
3. Replace the name with a new one.
4. Press **Enter** or click outside the name to make the name change take effect.

You can also double-click the object name and then enter a new name.

The change is effective immediately. Object name changes in the wave configuration do not affect those in the design source code.

Changing the Display Names

You can display the full hierarchical name (long name), the simple signal/bus name (short name), or a custom name for each signal. The signal/bus name displays in the Name column of the wave configuration.

If the name is hidden:

- Expand the Name column until you see the entire signal name.
- Use the scroll bar in the Name column to view the name.

To Change the Display Name

1. Select one or more signal or bus names. Use **Shift**+ click or **Ctrl**+ click to select many signal names.
2. Right-click, and select **Name >**
   - **Long** to display the full hierarchical name.
   - **Short** to display the name of the signal or bus only.
   - **Custom** to display the custom name given to the signal when renamed. See Renaming Objects.

The name changes immediately according to your selection.

Changing the Radix

To Set the Default Radix

The default radix controls the bus radix displayed in the wave configuration, Objects panel, and the Console panel. To change the default radix from the default binary:

1. Select **Edit > Preferences**.
2. In the Preferences dialog box, click **ISim Simulator** in the left pane.
3. Select a radix from the Default Radix field drop-down list.
4. Click **Apply**, and click **OK**.

To Change an Individual Radix

You can change the radix of an individual signal (HDL object) in the Object panel as follows:

1. Right-click on a bus in the Objects panel.
2. Select **Radix**, and the desired format from submenu menu:
   - **Binary**
   - **Hexadecimal**
   - **Unsigned Decimal**
   - **Signed Decimal**
   - **Octal**
   - **ASCII**

Note Changes to the radix of an item in the Objects panel do not apply to values in the Wave window or the Console panel. To change the radix of an individual signal (HDL object) in the Wave window, use the Wave window context menu. To change the radix in the Console panel, use the isim set radix Tcl command.

Reversing Bus Bit Order

You can reverse the bus bit order in the wave configuration in order to switch between MSB-first and LSB-first signal representation.

To Reverse a Bus' Bit Order

1. Select a bus.
2. Right click and select **Reverse Bit Order**.

The bus bit order is reversed. The Reverse Bit Order command is marked to show that this is the current behavior.

Navigating the Wave Configuration

Expanding and Collapsing a Hierarchy

You can expand and collapse a hierarchy in any window or panel with objects in nested groups using one of the following methods.

Clicking the Arrow

- Click the arrow to expand the hierarchy. One level can be expanded at a time.

- Click the arrow to collapse the hierarchy.

Menu Command

1. Select an object.
2. Select **Edit > Wave Objects >**
   - **Expand -** Expands the hierarchy object that is selected. One level can be expanded at a time.
   - **Collapse -** Collapses the hierarchy of the object selected.

Using the Context Menu

1. Select an object.
2. Right click, and select the applicable command from the context menu
   - **Expand -** Expands the hierarchy object that is selected. One level can be expanded at a time.
   - **Collapse -** Collapses the hierarchy of the object selected.

Zooming In and Out

Use the zoom functions to view your wave configuration in the Wave window as needed.

Using Commands

Zooming Behavior	Command	Shortcut Key
Zoom in	Select **View > Zoom > In**, or click the Zoom In toolbar button .	• Press **F8**. • Press **Ctrl**, and in the wave configuration, draw a line *down* to the *left*. **Note** The middle-mouse button can be used in place of the Ctrl key.
Zoom out	Select **View > Zoom > Out**, or click the Zoom Out toolbar button .	• Press **F7**. • Press **Ctrl**, and in the wave configuration, draw a line *up* and to the *right*. **Note** The middle-mouse button can be used in place of the Ctrl key.
Zoom to display the entire waveform in the window	Select **View > Zoom > To Full View**, or click the Zoom to Full toolbar button .	• Press **F6**. • Press **Ctrl**, and in the wave configuration, draw a line *up* and to the *left*. **Note** The middle-mouse button can be used in place of the Ctrl key.
Zoom to an area you select	Not applicable.	Press **Ctrl**, and draw a box around an area of the wave configuration from the top left corner of the area to the bottom right. **Note** The middle-mouse button can be used in place of the Ctrl key.

Zooming Behavior	Command	Shortcut Key
Zoom to display the time range between the two cursors	1. Place the cursors in the waveform. 2. Select **View > Zoom > To Cursors**, or click the Zoom to Selected toolbar button.	Not applicable.

Using the Middle Mouse Wheel

Function	Command
Zooms in and out	Press **Ctrl** and move the Mouse Wheel
Scrolls left and right	Press **Shift** and move the Mouse Wheel
Scrolls up and down	Move the Mouse Wheel only

Displaying the Floating Ruler

The floating ruler assists with time measurements using a time base other than the absolute simulation time shown on the standard ruler at the top of the Wave window. A floating ruler can be displayed (or hidden) and moved to the desired location in the Wave window. The floating ruler's time base (time 0) is the secondary cursor, or, if there is no secondary cursor, the selected marker. The floating ruler button and the floating ruler itself are visible only when the secondary cursor (or selected marker) is present.

To Display the Floating Ruler

1. Either:
   - Place the secondary cursor.

   Or:
   - Select a marker.
2. Select **View > Floating Ruler**, or click the Floating Ruler button.

   You only need to follow this procedure the first time. The floating ruler will appear each time the secondary cursor is placed or marker is selected, as instructed in Step 1.

The floating ruler displays.

Select the command again to hide the floating ruler.

Displaying Waveform Values With Markers

Markers, which are lines that intersect the waveform at a particular time, can be used to navigate through the wave configuration and to display the signal and bus values in the Value column for each marker. Follow this procedure to jump the main cursor from marker to marker to display waveform values.

To Display Waveform Values With Markers

1. In the wave configuration in the Wave window, add one or more markers, as described in Adding a Marker.

   For a single marker, if the main cursor and marker occupy the same location, the Value column displays the signal and bus values. No further action is required.

   For multiple markers, continue.

2. Select **Edit > Markers > Next Marker**, or click the Next Marker toolbar button .

   The cursor advances through markers in the wave configuration.

3. Observe the values in the Value column at each marker.

4. Select **Edit > Markers > Previous Marker**, or click the Previous Marker toolbar button .

   The cursor moves backward through markers in the wave configuration.

5. Observe the values in the Value column at each marker.

Displaying Waveform Values at Signal Transitions

You can view signal values at each transition of a signal in your waveform using the Next Transition or Previous Transition commands. The starting point is the main cursor.

To Use Next and Previous Transition Commands

1. Select a signal.

   The starting point is the location of the cursor in the waveform.

2. To advance to the next transition, select **View > Cursors > Next Transition** or click the Next Transition icon .

   The marker advances to the next transition for the selected signal. The values for *all* signals at that time are displayed in the Value column.

3. Repeat step 2 as necessary.

4. To go back to the previous transition, select **View > Cursors > Previous Transition** or click the Previous Transition icon .

   The marker moves back to the previous transition for the selected signal. The values for *all* signals at that time are displayed in the Value column.

5. Repeat step 4 as necessary.

The cursor advances or moves back, and the signal values are updated.

Measuring Time with Cursors

Together, the main and secondary cursors in the wave configuration measure a range of time. In addition to measuring time, the time range also forms the focal point for zooming to cursors and printing a range.

To Measure Time With Cursors

A quick time measurement between one transition and another, possibly between two different signal waveforms, can be done with the following procedure.

Note The Snap to Transition button is on by default which will assist with placing the cursor more precisely on a signal transition because the cursor snaps to the transition when in close proximity.

1. Place the mouse on the first transition and press and hold the left mouse button.
2. Drag the mouse to the second transition.
3. Release the mouse button.

   These actions place the secondary cursor on the first transition, and the main cursor on the second transition.
4. Examine the values at the bottom of the wave configuration:
   - X1 represents the time of the main cursor.
   - X2 represents the time of the secondary cursor.
   - Delta X represents the time range between the cursors.

   If the floating ruler is displayed, the time values of the cursors display just above the floating ruler.
5. *Optional*. You can switch the cursors using the Swap Cursors icon .

The time range displays until you click in the wave configuration to place a new main cursor.

If you move the secondary cursor, as described in Placing Cursors, the time range updates automatically.

Measuring Time with Markers

When the floating ruler is displayed, you can view the time measurement between the floating ruler time base (at the selected marker), and the main cursor and markers in the waveform.

To Measure Time with Markers

1. Display the floating ruler using a selected marker as the time base.
2. Add additional markers.
3. Move markers to desired location within the waveform.

   The marker label above the floating ruler shows the time intervals between the selected marker and each of the new markers.

   The time base can be switched quickly from one marker to another simply by clicking another marker to select it.

You can also use a combination of cursors and markers for measuring time. To do so, use the secondary cursor as the time base, and the floating ruler displays the time measurement of the markers and main cursor against the secondary cursor.

Using Go To Time

The Go To Time function enables you to jump the main cursor to a particular time in the wave configuration. Two Go To Time toolbar button also enables you to jump to time 0 (zero) and the latest simulation time. All signal and bus values are updated for the location of the main cursor.

To Go To a User-Specified Time

With a wave configuration open:

1. Select **Edit > Go To**.

   The Go To Time entry box appears in the Wave window below the wave configuration.
2. In the Go To Time box, enter the time and time unit you wish to jump the cursor to, or select a time/time unit from the drop-down list, if available.
3. Press **Enter**.

To Go to Time 0 or Latest Time

With a wave configuration open:

- Select the Go To Time 0 toolbar button to move the cursor to time 0 in the wave configuration.
- Select the Go To Latest Time toolbar button to move the cursor to the latest time in the wave configuration.

Using Show Drivers

You can use the Show Driver command to display the driver for a change in signal, or object value. This command is used to determine the cause of a value change, which will help determine if circuit connections are correct. ISim displays the signal, or object, and its one or more drivers in the Console panel.

The Show Driver command is available for probing objects in the following areas:

- Objects panel
- Wave window
- Console panel (`show driver` command)

To Show Drivers

1. Select an object, or signal.
2. Select **Edit > Wave Objects > Show Drivers**, or select **Show Drivers** from the right-click menu.

In the Console, the drivers for the object, or signal are listed. When there is no driver, a message in the Console indicates that there is no driver.

Note Running this command is the same as running `show driver` at the Console prompt.

Printing Wave Configurations

You can print one wave configuration at a time, using the setting specified during print setup. The wave configuration always prints with a white background.

To Print Preview

1. Select **File > Print Preview**
2. In the Print Preview dialog box, make sure the wave configuration looks as expected.
3. Select **Print** or **Setup** to further customize the print options and layout, and to finally print. See “To Print”.
4. Select **Close** to close the Print Preview dialog box.

Print preview displays the wave configuration in black and white, or color as determined by your default printer. You can select another printer, if available, just prior to printing.

To Print

1. Select **File > Print**.
2. In the ISim Print Setup dialog box, specify the Page Orientation, Time Range, Fit Time Range To, and other display settings.

   Note Time Range is populated with the time range of the main and secondary cursor if both are placed in the wave configuration.
3. Click **OK**.
4. In the Print dialog box, select a printer.
5. Click **Print**.

The wave configuration prints according to the print settings.

Using Custom Colors

Creating Custom Color Schemes

You can modify the colors used in the wave configuration by creating your own waveform color scheme.

To Create a Custom Color Scheme

1. Select **Edit > Preferences**.
2. In the left pane, expand **ISim Simulator**, and select **Colors**.
3. In the Colors preference page, click **New**.
4. Enter a name for the color scheme.
5. Click the color box and select a color for the areas you wish to change the color.
6. Click **Apply** to set the new color scheme.
7. Click **OK**.

The custom waveform scheme takes effect immediately.

Changing Signal Colors

You can change the color for individual signals or buses to make them stand out for comparison purposes.

General color settings are part of the color scheme specified in the Colors Preferences page. Changing an individual signal/bus color overrides the general color setting for signal/bus waveforms.

To Change a Signal Color

1. Right-click on a signal or bus.
2. Select **Signal Color** and select a color.

The signal or bus waveform displays the new color selection.

Note When using custom colors, all logic values including special values, such as 'X' and 'Z', display in the same color.

XILINX®

Chapter 7

Saving and Opening Simulation Results

Saving the Results

The simulation results of the objects (VHDL signals, or Verilog reg or wire) being traced are automatically saved to the Waveform Database (WDB) file (*<filename>*`.wdb`) in the working directory. If you add objects to the Wave window and run the simulation, the design hierarchy for the complete design and the transitions for the added objects are automatically saved to the WDB file. The wave configuration settings, which include the signal order, name style, radix and color, among others, are saved to the wave configuration (WCFG) file upon demand.

Saving a Database to a WDB File

When ISim is launched from the ISE® software, the WDB file is named according to the name specified in the ISim Properties dialog box. When launched from the command line, the **`-wdb`** switch is used to specify a filename. When a simulation is run, results of the objects (VHDL signals, Verilog reg or wire) being traced are automatically saved to the WDB; no additional action is required.

If another simulation is run on the same design in the same working directory as a currently running simulation, the WDB filename for the new simulation is the name of the first simulation appended with an integer. The results of the first simulation are not overwritten. For a WDB file called `isim.wdb`, subsequent invocations of ISim result in WDB files `isim1.wdb`, `isim2.wdb`, etc.

Note The name of the database file cannot be changed once in a simulation.

Saving a Wave Configuration to a WCFG File

The WCFG stores the order and inclusion of simulation objects, their properties, and any added wave objects, such as dividers, and markers found in the Wave window. For more information, see Wave Window Overview.

When saving a WCFG file, select **File > Save**, and specify a file name for the .wcfg file.

You can open the wave configuration file to view results in the static simulator. For more information, see Opening a Static Simulation.

Relationship Between WDB and WCFG

When you save the wave configuration (WCFG) file, ISim automatically adds a reference in the file to the associated waveform database (WDB) file. There can be multiple WCFG files for a single WDB file.

Opening a Live Simulation

A live simulation consists of the following:

- a waveform database file, which contains all simulation data
- a wave configuration file, which contains the order and settings associated with objects in the wave configuration

For information about saving simulation results, see Saving the Results.

Opening a Waveform Database

A WDB is automatically opened when a simulation is run. For more information about running a simulation, see Step 3: Simulating the Design.

Opening a Wave Configuration

When a simulation is run from the ISE® software, a default wave configuration is automatically opened in ISim. You can open additional wave configurations. When running a simulation from the command prompt or using a batch script, a wave configuration is not opened by default when the GUI is launched, and you must open or create one.

To open a wave configuration in the ISim GUI:

1. Select **File > Open**.
2. Select .wcfg from the file type list.
3. Select the file you wish to open.
4. Click **OK**.

The wave configuration opens in the Wave window. Multiple wave configurations can be opened during one simulation session. Click the corresponding wave configuration tab to view the wave configuration.

To open a wave configuration from the command prompt:

Since a wave configuration is not open by default when you launch the GUI, you can include the **`-view`** switch to open an existing one. Run the simulation executable with the following syntax:

<*sim_exe*>**`.exe -gui -wdb`** <*wdb*>**`.wdb -view`** <*wcfg*>**`.wcfg`**

where

- **`-gui`** launches the ISim graphical user interface
- **`-wdb`** <*wdb*>**`.wdb`** specifies the filename where the simulation data will be stored
- **`-view`** <*wcfg*>**`.wcfg`** opens the specified waveform file in the ISim graphic user interface

The ISim GUI opens with a new database (a live simulation). If simulation objects from the WCFG correspond to those in the database, the wave configuration is pre-populated with data from the database.

For information about creating a new wave configuration, see Creating New Wave Configurations.

Opening a Static Simulation

A static read-only simulation consists of a wave configuration file, which contains the order and settings associated with objects that are displayed in the wave configuration, and a waveform database file, which contains the simulation data from a previously run live simulation. A wave configuration file references the waveform database for which it was created. A simulation cannot be performed in the static simulator. For information about opening a live simulation, see Opening a Live Simulation.

Opening a Wave Configuration and Waveform Database

Opening an Existing WCFG With Associated Waveform Database

If you have a waveform configuration (WCFG) file from a previous simulation run and you want to open the WCFG and its associated simulation data (WDB), use one of the following methods to open the WCFG and its associated WDB.

In a command prompt, either:

1. Run **`isimgui.exe`** to open the static simulator.
2. Select **File > Open**, select the **.wcfg** file type from the file filter list, and select the wave configuration (WCFG) file from a previous simulation.

Or:

1. Run **`isimgui.exe -view`** <*wcfg_file*>**`.wcfg`** to a WCFG file in the static simulator.

   where:

   - **`isimgui.exe`** is the application executable.
   - **`-view`** <*wcfg*>**`.wcfg`** opens the specified waveform file in the ISim graphical user interface.

The static simulator displays the wave configuration, with all signals previously traced, and the associated waveform database.

Opening an Existing WCFG and Non-Associated Waveform Database

You can load simulation data (WDB), and view a WCFG that is not associated with the database. This particular way of opening a static simulation is useful for cases when different views (as captured in a WCFG file) of the same simulation result (the transitions stored in the WDB file) need to be seen by different engineers in a team. ISim will issue a warning for any object name that is present in the WCFG but not found in the WDB file and will display only those objects that match.

In a command prompt:

1. Run **`isimgui.exe -open`** <*wdb*>**`.wdb -view`** <*wcfg*>**`.wcfg`**

   where:

   - **`isimgui.exe`** is the application executable.
   - **`-open`** <*wdb*>**`.wdb`** opens the specified waveform database file in the ISim graphic user interface.
   - **`-view`** <*wcfg*>**`.wcfg`** opens the specified waveform file in the ISim graphical user interface.

Opening a Waveform Database and a New Default WCFG

If you have simulation data (WDB) that you wish to analyze and you do not want to open a previous WCFG, use the following method to open the Waveform Database and a new default WCFG.

In a command prompt:

1. Run **`isimgui.exe -view`** <*wdb_file*>**`.wdb`**

   where:

   - **`isimgui.exe`** is the application executable.
   - **`-view`** <*wdb*>**`.wdb`** opens the specified waveform database file in the ISim graphical user interface.

The static viewer displays the data from the previous simulation and a new wave configuration file named as `Default.wcfg` that displays up to a maximum of 1000 objects from the WDB file in the Wave window. You can remove or add signals to the default WCFG file, and save the WCFG for future viewing.

Opening a Waveform Database Only

If you would like to open a WDB from a previous simulation and no WCFG, use this method:

In a command prompt, either:

1. Run **`isimgui.exe`** to open the static simulator.
2. Select **File > Open**, select the .wdb file type from the file filter list, and select the wave database (WDB) file from a previous simulation

Or:

1. Run **`isimgui.exe -open`** <*wdb_file*>**`.wdb`**

   where:

   - **`isimgui.exe`** is the application executable.
   - **`-open`** <*wdb*>**`.wdb`** opens the specified waveform database file in the ISim graphical user interface.

The static viewer displays the data from the previous simulation in the Objects panel, and the Instances and Processes panel. There is no waveform data open in the Wave window.

You can open an existing wave configuration using **File > Open**, or create a new wave configuration using **File > New**.

XILINX®

Chapter 8

Debugging

Source Level Debugging Overview

ISim enables you to debug your HDL source code to verify that the design is running as expected. Debugging is accomplished through controlled execution of the source code to determine where problems may be occurring.

Some strategies available for debugging in ISim are:

- **Step through the code line by line**

  For any design at any point in development, you can use the Step command to debug your HDL source code one line at a time to verify that the design is working as expected. After each line of code the Step command is run again to continue the analysis. For more information, see Stepping Through a Simulation.

- **Set breakpoints on the specific lines of HDL code, and run the simulation until a breakpoint is reached**

  In larger designs, it can be cumbersome to stop after each line of HDL source code is run. Breakpoints can be set at any predetermined points in your HDL source code, and the simulation is run (either from the beginning of the test bench or from where you currently are in the design) and stops are made at each breakpoint. You can use the Step, Run All or Run For command to advance the simulation after a stop. For more information, see Using Breakpoints to Debug Your Design.

Stepping

Stepping Through a Simulation

You can use the Step command at any point in the simulation to debug your HDL source code. The Step command executes your HDL source code one line of source code at a time to verify that the design is working as expected. A yellow arrow points to the line of code currently being executed.

You can also create breakpoints for additional stops while stepping through your simulation. For more information on debugging strategies in ISim, see Source Level Debugging Overview

To Step Through a Simulation

1. Do one of the following:
   - Select **Simulation > Step**.
   - Click the Step toolbar button .
   - Type the step command at the Console prompt.

   The HDL associated with the top design unit will be opened as a new tab in the Wave window.

   Note Stepping will start from the current running simulation time. If you'd like to step through the simulation from the start (0 ns), restart the simulation. Use the Restart command to reset time to the beginning of the test bench. See Running a Simulation in ISim.
2. Select **Window > Tile Horizontally** (or **Window > Tile Vertically**) to simultaneously see the waveform and the HDL code.
3. Repeat the Step action until debugging is complete.

As each line is executed, you can see the yellow arrow moving down the code. If the simulator is executing lines in another file, the new file opens, and the yellow arrow will step through the code. It is common in most simulations for multiple files to be opened when running the step command. The Console panel also indicates how far along the HDL code the step command has progressed.

Breakpoints

Setting Breakpoints

When working with ISim, you can set breakpoints in executable lines in your HDL file as follows. This allows you to run your code continuously until the source code line with the breakpoint is reached, as described in Using Breakpoints to Debug Your Design.

Note You can only set breakpoints on lines with executable code.

To Set a Breakpoint

1. Select **View > Breakpoint > Toggle Breakpoint**, or click the Toggle Breakpoint toolbar button .
2. In the HDL file, click a line of code just to the right of the line number.

Note Alternatively, you can right-click a line of code, and select **Toggle Breakpoint**. You can also click a breakpoint to remove it.

Once the procedure is completed, a simulation breakpoint icon appears next to the line of code.

Note If you place a breakpoint on a line of code that is not executable, the breakpoint is not added.

A list of all breakpoints is available in the Breakpoints panel.

Debugging Your Design Using Breakpoints

A breakpoint is a user-determined stopping point in the source code used for debugging the design with ISim. Breakpoints are particularly helpful when debugging larger designs for which debugging with the Step command (stopping the simulation for every line of code) may be too cumbersome and time consuming. For more information on debugging strategies in ISim, see Source Level Debugging Overview.

To Use Breakpoints for Design Debugging

1. Open the HDL source file. See Opening HDL Source Files.
2. Set breakpoints on executable lines in the HDL source file. See Setting Breakpoints
3. Repeat steps 1 and 2 until all breakpoints are set.
4. Click the Wave window to return to the waveform.
5. Run the **Simulation > Restart** command if you wish to start from the beginning.
6. Run a simulation using the **Simulation > Run All** or **Simulation > Run for Specified Time** command .

   The simulation will run until one of the breakpoints is reached, at which point, the simulation stops. The HDL source file displays, and the breakpoint stopping point is indicated with a yellow arrow.
7. Again, click the Wave window to return to the waveform to see if the design behavior (i.e., signal value change) is as expected at the breakpoint.
8. Repeat steps 6 and 7 to advance the simulation, breakpoint by breakpoint, until you are satisfied with the results or you wish to stop this debugging process.

A controlled simulation is run, stopping at each breakpoint set in your HDL source files.

During design debugging, you can also run the **Simulation > Step** command to advance the simulation line by line to debug the design at a more detailed level. For more information, see Stepping Through a Simulation

Deleting Breakpoints

When working with ISim, you can delete breakpoints from your HDL source code as follows.

To Remove a Breakpoint

Use one of the following methods:

- Click the simulation breakpoint icon .
- At the Tcl prompt:
  1. Enter **`bp list`** to list all breakpoints in your design and shows each breakpoint index number and line number.
  2. Enter **`bp del`** or **`bp remove`** followed by the breakpoint index number of the breakpoint you wish to remove. See syntax and examples in bp Command.

Note You can also remove a breakpoint in the Breakpoints Panel by selecting a breakpoint and using the **Delete** context-menu command, or the Delete icon .

The single breakpoint is deleted.

To Remove All Breakpoints

Use one of the following methods:

- Select **View > Breakpoint > Delete All Breakpoints**.
- Click the Delete All Breakpoints toolbar button .
- Enter **`bp clear`** at the Tcl prompt.

All breakpoints in all HDL files are deleted.

XILINX®

Chapter 9

Writing Activity Data for Power Consumption

Writing Activity Data of the Design

ISim writes out files with switching activity data of the design, which is useful for two tasks:

- Estimating the power consumption with a power analysis tool, such as XPower Analyzer.
- Implementing the design for optimal power consumption with Map, and Place & Route (PAR) tools.

The switching activity data can be written out from the simulation of the design either at the RTL-level or after full placement and routing. Map, PAR and XPower Analyzer all work with switching activity data generated from both RTL and post-place and route simulation. For better accuracy in power analysis and power optimized implementation, it is recommended to use switching activity data generated from a post-place and route simulation. The data will match the design internal nodes that result from the placement and routing.

It is also possible to use switching activity data generated from a RTL simulation (quicker than the post-place and route simulation) for both power analysis and power driven implementation. However, only activities for the inputs and outputs of the design will be taken into account. The tools will use their vector-less analysis algorithms to estimate activities of the design's internal nodes.

For more information about how these tools use the switching activity data, see the *Command Line Tools User Guide* for implementation tools (Map and PAR), and XPower Analyzer Help for power analysis.

Activity File Types

Two stimulus files can be written out in an ISim simulation:

- **SAIF -**The Switching Activity Interchange format (SAIF) file contains toggle counts (number of changes) on the signals of the design. It also contains the timing attributes which specify time durations for signals at level 0, 1, X, or Z. The SAIF file is recommended for power related tasks (i.e., power analysis or power driven implementation) since it is smaller than the VCD file.
- **VCD -** The Value Change Dump (VCD) file is an ASCII file containing header information, variable definitions, and value change details for each step of the simulation. The file can be used to estimate the power consumption of the design. The computation time of this file can be very long, and the resulting file size is larger than the SAIF file.

To Write Activity Files for Power

Writing out a switching activity file can be done as follows:

1. Create the desired activity file in order to gather the signals transition during simulation for power estimation.
   - **SAIF -** Use the saif command at a Tcl prompt.
   - **VCD -** Use the vcd command at a Tcl prompt, or set the **`-vcdfile`** option in your simulation executable at the command line.
2. Run simulation, for example, run 1000 ns.

   If you are running your simulation using the simulation executable at the command line, the first and second steps can be accomplished at once.
3. When simulation ends, close the SAIF or VCD file by issuing the appropriate saif or vcd command. Example: **`saif close`** or **`vcd dumpoff`**.
4. Retrieve the SAIF or VCD file from the `isim` working directory for use in another tool.

Chapter 10

Using Tcl Simulation Commands

Simulation Command Overview

Simulation commands enable you to run an interactive simulation at a command prompt.

Simulation Command Entry Methods

Simulation commands can be entered as follows. For more information, see Entering Simulation Commands.

- **ISim GUI -** Enter simulation commands in the ISim Console panel.
- **Command Line -** Enter simulation commands at the command line Tcl prompt.
- **Tclbatch -** Enter simulation commands in a Tcl file, and reference the Tcl file with the **-tclbatch** option of the simulation executable. For more information, see ISim Simulation Executable.

You can enter individual commands or create simulation scripts. For examples of simulation Tcl scripts, see the Simulation Constructs folder available in the Language Templates. To access these examples, click **Edit > Language Templates**, and in the Language Templates, expand **Tcl**, expand **Tools**, and expand **ISim** folder.

You can set a variable to represent a simulation command to quickly run frequently used simulation commands. For more information, see Aliasing Simulation Commands.

Note For information on using Tcl, see Tcl/Tk Documentation.

Simulation Command Summary

Available simulation commands are as follows. Information on syntax, options and examples is available for each simulation command.

Note These commands are case sensitive.

- bp: Sets and deletes breakpoints in your HDL source code for debugging purposes.
- describe: Displays information about the given HDL data or block object.
- dump: Displays a list of variables, generics, parameters and nets along with their values for the current scope of the design hierarchy.
- help: Displays a description with usage and syntax of the specified ISim command.
- isim condition: Enables the execution of a set of commands based on a specified condition on one or more nets or Verilog regs.
- isim force: Forces or removes a value on a net.
- isim get arraydisplaylength: Displays the limit of numbers of elements for an array type HDL object.
- isim get radix: Gets the global radix being used.
- isim get userunit: Displays the current unit of measurement for all time values where unit is unspecified.
- isim ltrace: Turns on and off line tracing.
- isim ptrace: Turns on and off tracing of execution of processes.
- isim set arraydisplaylength: Sets the limit on the number of elements for an array type HDL object.
- isim set userunit: Sets the default unit of measurement for all time values where unit is unspecified.
- isim set radix: Sets the global radix to use.
- ntrace: Controls how ISim traces variables and nets in your simulation.
- onerror: For batch mode, controls the behavior immediately following a failed Tcl simulation command.
- put: Assigns a value to a specified bit, slice or all of a variable or signal.
- quit: Exits either the simulation or the software, depending on the command option.
- restart: Restarts simulation. It sets simulation time back to 0.
- resume: With the **`onerror`** command, continues executing commands after an error is encountered.
- run: Starts simulation.
- saif: Creates a Switching Activity Interchange format (SAIF) file and records estimate power usage.
- scope: Navigates the design hierarchy.
- sdfanno: Back-annotates delays from an SDF file to the HDL design.
- show: Displays selected aspects of the design in the Simulation Console tab.
- step: Executes simulation through your HDL design, line by line, to assist with debug.
- test: Compares the actual value of a net or bus with a supplied value.
- vcd: Generates simulation results in VCD format.
- wave add: Adds simulation objects or blocks to the specified wave configuration in the ISim graphical user interface.

Entering Simulation Commands

How you enter simulation commands depends on how you are running ISim. See the applicable user type below.

For information on simulation commands and command syntax, see Simulation Command Overview or type **`help`** or **`help`** <*command_name*> at the prompt.

To Enter Commands in GUI Mode

Once ISim is launched, you can run Tcl commands in the Console.

1. Click the Console panel.
2. Place the cursor in a new line.
3. Type in the simulation command using the correct syntax.
4. Press **Enter** to run the command.

The command is run. The command log appears in the Console panel.

To Enter Commands in Interactive Command Line Mode

After parsing, and elaborating your VHDL, Verilog or Mixed Language design, the following procedure can be followed to run simulation and analyze the design.

1. Run the ISim simulation executable that you generated from fuse at the command line, for example, **`my_sim.exe`**.

   A Tcl prompt appears:

   ```
   This is a Full version of ISim.
   Time resolution is 1 ps
   ISim>
   ```

2. Type in the simulation command using the correct syntax.
3. Press **Enter** to run the command.

The command is run. Simulation output will automatically be directed to both the `stdout` as well as the `isim.log` file.

Note You can use the **Ctrl+C** keyboard command to stop a simulation once invoked.

Example of a typical sequence.

1. **`scope`** - Shows current scope in design hierarchy.
2. **`dump`** - Shows you current value of signals in current scope.
3. **`show value clk`** - Shows the value of a signal named clk.
4. **`run 1000 ns`** - Runs simulation for 1000 ns.
5. **`restart`** - Resets simulation to time 0.
6. **`ntrace select -o on -m / -l this`** - Selects tracing of all signals at top level /
7. **`ntrace start`** - Turns on tracing of selected signals.
8. **`run 1000 ns`** - Runs simulation for 1000 ns with trace turned on.
9. **`ntrace stop`** - Stops tracing.
10. **`run 1000 ns`** - Runs simulation for 1000 ns with trace turned off.
11. **`quit`** - Quits simulation.

    After quitting simulation, since the **`ntrace`** command was used, you will see that the waveform trace file `isimwavedata.xwv` has been created in your current working directory. Note that this file is not created if you do not use the **`ntrace`** command.

To Enter Commands in Non-Interactive Batch Mode

In batch mode, you must collect all Tcl commands in a Tcl file, and run the simulation executable with the Tcl file being referenced.

1. Create a file with a `.tcl` extension, for example, **`isim.tcl`**, and enter any simulation commands you want to execute into the file.

   You can enter individual commands or create simulation scripts. For examples of simulation Tcl scripts, see the Simulation Constructs folder available in the Language Templates with the Project Navigator. To access these examples in the Language Templates:

   a. Open Project Navigator.

   b. Select **Edit > Language Templates**.

   c. Expand **Tcl**, expand **Tools**, and expand **ISim** folder.

   Note When creating a Tcl file, ensure that you have a quit command on the last line of the file to ensure the simulation completely exits once the run is complete.

   For example, the following could be the contents of `isim.tcl`:

```
ntrace select -o on -m / -l this
run 1000 ns
```

2. To execute the commands in `isim.tcl`, enter the following at the command line prompt:

   `stopwatchsim.exe -tclbatch isim.tcl`

   Simulation output will automatically be directed to both the `stdout` as well as the `isim.log` file.

Aliasing Simulation Commands

You can set a variable to represent a simulation command using a Tcl-based command prompt. You can then use the variable to execute the command many times without having to type it in each time. Setting a variable to represent one or more commands is called *aliasing*.

To Set a Variable at the Simulation Console

In the Console panel (GUI mode) or Tcl prompt (command-line mode), type in a variable as follows:

`set svc "show value count"`

where,

- **`set`** - indicates that you are creating a variable.
- **`svc`** - is the variable name.
- **`"show value count"`** - is the simulation command represented by the variable name.

The variable is set. The Tcl variable **`svc`** is set to show value count.

To then run the variable (and thereby execute the simulation command), type **`eval $svc`**.

Command Line Conventions

You can enter console commands at the command line of the Simulation Console window as an alternative to using menu commands. Console commands do not display the dialog boxes that menu commands invoke.

The following is a summary of the syntax used for the simulation commands.

- When an option occurs with parentheses "()" next to the command name in the syntax, it is required.
- When an option appears in square brackets "[ ]," it is optional.
- When two or more options occur between braces "{ }," the options must be entered with the command.
- If the options are separated by a vertical bar "|," you must choose one of the possible parameters. If one term is divided into a subset of parameters that can be entered separately or together, each subparameter occurs between square brackets.
- Less than and greater than symbols "< >" enclose variables for which you must supply values.

Note *Italics* are used to indicate variables.

Tcl Commands

bp Command

The **bp** command controls the setting and removal of breakpoints in the HDL source code that you are simulating. A breakpoint is used to interrupt the simulation during debugging.

Note This command is case sensitive.

bp (*option*)

Options

add <*file_name*><*line_number*>	Adds a breakpoint in your HDL code. `File_name` is the name of the HDL source file that you are simulating where you want to put a breakpoint. *Line_number* is the number of the line of HDL code where you want the simulation to stop.
clear	Deletes all breakpoints for all HDL files loaded into ISim. If you have breakpoints in multiple files, all breakpoints are deleted.
del <*index*> [<*index*>...]	Deletes individual breakpoints from your HDL code. Before using this command, run the **bp list** command to obtain the index numbers for your breakpoints. See the **list** option for details. *Index* is the index number assigned to the breakpoint you want to delete. Each breakpoint in your design is assigned a unique index number. **Note** This index is *not* the line number of the breakpoint in the source file.
list	Lists all of the breakpoints in your design and shows their index number to be used with the **bp del** command. If you have set breakpoints in multiple files, all breakpoints are listed. The **bp list** commands returns the following: *index* `directory_path/file_name`::*line_number* Where: • *Index* is the index number to use with the bp del command. • `Directory_path` is the fully qualified path to your source files. • `File_name` is the name of the source file that has a breakpoint in it.

	• *Line_number* is the line number in the source file of the breakpoint.
remove <*file_name*> <*line_number*>	Deletes a breakpoint at line *line_number* in file `file_name`. `file_name` is the name of the HDL source file that you are simulating and where you want to remove a breakpoint. *line_number* is the number of the line of HDL code where the breakpoint has been set.

Examples

The **bp** command can be used as follows.

Set a Breakpoint

This example sets a breakpoint at line 2 for the file `statmach.vhd`.

```
bp add statmach.vhd 2
```

List all Breakpoints

This example lists all breakpoints in all files involved in your ISim simulation, and gives them a unique index number.

```
bp list
```

Delete Breakpoints

This example deletes all breakpoints in your simulation.

```
bp clear
```

This example deletes a breakpoint by index number.

1. First use the **bp list** command to get the breakpoint indexes:

   ```
   bp list
   ```

   The simulator returns the following:

```
1 C:/examples/watchvhd/stopwatch_tb.vhd::46
2 C:/examples/watchvhd/stopwatch_tb.vhd::55
```

2. Then delete the breakpoint at line number 46 in `stopwatch_tb.vhd`:

   ```
   bp del 1
   ```

This example deletes a breakpoint at line 2 in `statmach.vhd`.

```
bp remove statmach.vhd 2
```

describe Command

The **describe** command displays information about the given HDL data or block object.

The syntax for this command is as follows.

Note This command is case sensitive.

```
describe <object_name>
```

Options

<*object_name*>	Displays a description about either an HDL object or an HDL block in the current simulation scope.

Examples

The **describe** command can be used as follows.

```
ISim> describe param
```

```
Verilog Instance:  {param}
Path:{/parameter8_hn/param}
Location:  {/home/test5.v:42}
Instantiation:  {/home/test5.v:37}
```

dump Command

The **dump** command displays values for all VHDL signals and generics, and Verilog wires, non-subprogram regs and parameters in the current scope. To navigate the design hierarchy, use the **scope** command. The **dump** command uses the default radix set using **isim set radix** command.

Note This command is case sensitive.

The syntax for this command is as follows.

```
dump
```

Example

This example displays a list of all the signal names and their values at the current scope of the design hierarchy to your computer screen.

```
dump
```

help Command

The **help** command displays a description with usage and syntax of the specified ISim Tcl command. With no command specified, the **help** command displays descriptions, usage and syntax for all of the ISim Tcl commands.

The syntax for this command is as follows.

Note This command is case sensitive.

help [*command_name*]

where *command_name* is optional.

Options

command_name	Displays a description for the specified command. A list of all simulation commands is found in Simulation Command Overview.

Examples

The **help** command can be used as follows.

Help for All Commands

For a description of all of ISim commands:

```
help
```

Help for One Command

For a description of the bp command:

```
help bp
```

isim condition Command

The **isim condition** command adds, removes or generates a list of conditional actions. A conditional action is equivalent to a VHDL process or a Verilog always process. When added, it will start monitoring signals continually (that appear in the **isim condition** expression) during simulation. The condition expression is evaluated anytime a signal change is detected. When a specified condition expression is met, the specified command is run. **isim condition remove** stops monitoring signals. **isim condition list** lists all active conditional actions added and their labels and IDs.

Note This command is case sensitive.

isim condition {**add|remove|list**} [<*condition expression*> <*command*>] [*option*]

Options

{**add|remove|list**}	Specifies whether you wish to add a condition, remove one or more conditions, or list all active conditions.
<*condition expression*> <*command*>	The <*condition expression*> is associated with the **add** function, and it determines when to run the specified <*command*>. Operators used in the condition expression include != (not equal), **==** (equal), **&&** (and), and **||** (or). A space is required between words in the expression and the operator, such as **clk == St1**. The <*command*> is a Tcl command or script that will be executed when the condition expression is true. This command is surrounded by **{}** (braces). The command can include standard tcl commands and simulation tcl command, except run, restart, init, and step. Tcl variables used in the condition expression are surrounded by quotes **""** instead of **{}**. Refer to the syntax examples below.
-radix <*radix_type*>	An optional argument that is used to read the value in the condition expression. The supported radix types are **default**, **dec**, **bin**, **oct**, **hex**, **unsigned** and **ascii**. When no radix type is specified, the global radix type set with the **isim set radix** command is used, or if none is set there, **default** is used as radix.
-label <*label_name*>	An optional argument that specifies the name of a condition. For **isim condition add**, when no label is specified, the simulator generates an index used to identify the condition.
-index <*index_name*>	An optional argument that identifies a condition. Can be used with **isim condition remove** only.
-all	An optional argument that is used to remove all conditions in the current simulation.

Examples

isim condition add Examples

To add a condition that states that when the signal asig is equal to 8, a stop occurs, and the condition is called label0:

```
isim condition add { /top/asig  == 8 } {stop} -label label0 -radix hex
```

To add a VHDL-specific signal condition that states that when the signal asig is equal to 1, a stop occurs, and the condition is called label1:

```
isim condition add { /top/asig  == '1' } {stop} -label label1
```

To add a condition that states that for any change on signal asig, a stop occurs, and the condition is called label2:

```
isim condition add /top/asig  {stop} -label label2
```

To add a Verilog-specific signal condition that states that when clk is equal to St1, a stop occurs, and the condition is called label3:

```
isim condition add { clk == St1 } {stop} -label label3
```

To add a condition that states that when asig (3:0) is equal to 0001 and reset is equal to 1, a stop occurs:

```
isim condition add { asig(3:0) == 0001  && reset == 1 } {stop}
```

isim condition remove Examples

To remove all conditions in the current simulation:

```
isim condition remove
```

or

```
isim condition remove -all
```

To remove the conditions label0, label1 and label2:

```
isim condition remove -label { label0 label1 label2 }
```

To remove a single statement:

```
isim condition remove -index 2
```

or

```
isim condition remove -label label3
```

isim condition list Examples

To list all isim conditions added to the design at the current time.

```
isim condition list
```

isim force Command

The **`isim force`** command adds or removes a value on a signal. This command forces a value on the signal, until **`isim force remove`** is issued. This command works only for VHDL signals and Verilog wire types. It does not work for Verilog regs. The force command cannot be overridden by the stimulus from the design. The force is valid until the simulation is restarted or until the **`isim force remove`** command is issued

Note This command is case sensitive.

`isim force` {**`add|remove`**} < *`object_name`* > < *`value`* > [*`option`*]

Options

{add|remove}	Specifies whether you wish to assign or remove a value from a bus or signal.
-value *<value>*	Used to specify one or more values to add.
-radix *<radix_type>*	This option is used to read the value. The supported radix types are **default**, **dec**, **bin**, **oct**, **hex**, **unsigned** and **ascii**. When no radix type is specified, the global radix type set with the **isim set radix** command is used, or if none is set there, **default** is used as radix.
-time *<time>*	Time can be a string such as 10, 10 ns, "10 ns". When a number entered without a unit, the simulator resolution unit is used, which is ps. Time is relative to the time of execution of the command.
-cancel *<time>*	This option cancels the force command after the specified time.
-repeat *<time>*	This option repeats the cycle after the specified time.

Examples

The **isim force** command can be used as follows.

To Assign a Value

To signal s, assign value 1:

```
isim force add s 1
```

To signal s1, assign value 1 at time 0, and assign value 0 at time 100. Repeat this cycle after 200 units, and cancel force after 1000 time units:

```
isim force add s1 1 -time 0 -value 0 -time 100 -repeat 200 -cancel 1000
```

To signal s2, assign value 1 at time 0, and assign value 0 at time 100. Repeat this cycle after 200 units, and cancel force after 1000 time units:

```
isim force add s2 1 -time "0 ns" -value 0 -time "100 ns" -repeat "200 ns" -cancel
"1000 ns"
```

To Remove a Value

To remove the values on signal s, s1 and s2:

```
isim force remove s s1 s2
```

isim get arraydisplaylength Command

The **isim get arraydisplaylength** command enables you to display the limit of numbers of elements for array type HDL object. The limit can be set with the isim set arraydisplaylength command.

Note This command is case sensitive.

```
isim get arraydisplaylength
```

where *no* options are available.

Example

Enter

```
isim get arraydisplaylength
```

Returns

64

isim get radix Command

The **`isim get radix`** command enables you to display the default radix as a string. The radix is set with the isim set radix command.

Note This command is case sensitive.

The syntax for this command is as follows.

```
isim get radix
```

where *no* options are available.

Example

To return the radix:

```
isim get radix
```

Returns the current global radix.

isim get userunit Command

The **`isim get userunit`** command enables you to display the current unit of measurement for all time values where unit is unspecified. The unit of measurement can be set with the isim set userunit command.

Note This command is case sensitive.

```
isim get userunit
```

where *no* options are available.

Example

Enter

```
isim get userunit
```

Returns

```
1 ps
```

isim ltrace Command

The **`isim ltrace`** command enables you to turn line tracing on or off. When line tracing is turned on, you can do a line-by-line analysis for debugging. The HDL lines that are executed are printed to screen with the following information: simulation time, file path and file name, and line number.

Note **`isim ltrace on`** may slow down the simulation.

Note This command is case sensitive.

The syntax for this command is as follows.

```
isim ltrace (option)
```

Options

**`[on	off]`**	Turns line tracing on or off. Displays the name of the currently executing line in the Console. The default is off.

Example

To see which line is currently executing.

```
isim ltrace on
run
```

The output lists the simulation_time, filename, line number, as follow:

```
1005 ns "C:/Data/ISE_Projects/freqm/watchver/stopwatch_tb.v":26
1005 ns "C:/Data/ISE_Projects/freqm/watchver/stopwatch_tb.v":27
1005 ns(3) "C:/Data/ISE_Projects/freqm/watchver/statmach.v":63
1005 ns(3) "C:/Data/ISE_Projects/freqm/watchver/statmach.v":64
```

isim ptrace Command

The **isim ptrace** command enables you to turn process tracing on or off. When turned on, the command also displays the name of the currently executing VHDL or Verilog process in the Simulation Console tab. This is extremely useful if the simulator is stuck in an infinite loop, in which case the command points out the exact process where the simulator is stuck.

Note This command is case sensitive.

The syntax for this command is as follows.

isim ptrace (*option*)

Options

[**on** | **off**]	Turns process tracing on or off. Displays the name of the currently executing VHDL or Verilog process in the Simulation Console tab. The default is off.

Example

To see which process is currently executing.

```
isim ptrace on
```

isim set arraydisplaylength Command

The **isim set arraydisplaylength** command sets the limit on the number of elements for an array type HDL object. The following are affected by the limit set:

- The display of values in the Objects Panel in the graphical user interface
- The response to the **show value** command

The default is 64.

Note This command is case sensitive.

The syntax for this command is as follows:

isim set arraydisplaylength *<size>*

Options

<size>	Enter a number of elements of an array type HDL object to displays. Use 0 for unlimited length. Default is 64.

Examples

Enter

```
isim set arraydisplaylength 2
```

```
show value xcountout
```

Returns
```
00
00
```

Also, examine Value column for arrays in the Objects Panel in the ISim graphical user interface (GUI).

Enter

```
isim set arraydisplaylength 64
show value xcountout
```

Returns
```
0001000000
0001000000
```

Also, examine Value column for arrays in the Objects Panel in the ISim GUI.

isim set radix Command

The **`isim set radix`** command enables you to set the global radix for the current simulation. This radix type is used for other commands: show value, put, test, dump, isim force, and isim condition.

Note This command is case sensitive.

The syntax for this command is as follows:

```
isim set radix (option)
```

Options

<radix_type>	Sets the global radix for the current simulation. This radix type is used for other commands: show value, put, test, dump, isim force, and isim condition. The supported radix types are **`default`**, **`dec`**, **`bin`**, **`oct`**, **`hex`**, **`unsigned`** and **`ascii`**.

Examples

To set a radix of hex, and to see a count value.

```
isim set radix hex
show value count
```

Returns `a`.

To set a radix of dec, and to see a count value.

```
isim set radix dec
show value count //count is defined as reg[3:0] count
```

Returns `-4`.

To set a radix of unsigned, and to see a count value.

```
isim set radix unsigned
show value count
```

Returns `10`.

isim set userunit Command

The **`isim set userunit`** command enables you to set the default unit of measurement for all time values where unit is unspecified. The default time unit is the same as the engine's time-unit as set in the **`fuse`** command **`-timescale`** or **`-override_timeunit`** options. In the absence of these **`fuse`** options, the timescale is determined by:

- The default time unit, which is 1 ps.
- As defined by the 'timescale simulator directive in Verilog.

Note This command is case sensitive.

The syntax for this command is as follows:

`isim set userunit` (*`option`*)

Options

`<[1|10|100]fs|ps|ns|us|ms|s>`	The *userunit* is entered as number (1|10|100) followed by unit (fs|ps|ns|us|ms|s).

Example

Set the simulator timescale to 1ps:

```
Isim set userunit 1 us
```

ntrace Command

The **`ntrace`** command controls how ISim traces nets in your simulation. Nets are the highest level of declared signals. This command turns tracing on and off, and it selects nets to trace. The **`select`** option enables you to turn tracing on and off for individual nets. When the ntrace command is executed, the simulation then needs to be run. When a simulation is run from the ISE® software, **`ntrace`** is run automatically in order to draw out the waveforms.

Note This command is case sensitive.

Syntax

The syntax for setting this command is as follows:

`ntrace` (*`option`*)

Options

`select`	
`-o on | off`	Turns tracing on or off for a specified module or signal. The default is **`on`**.
`-m` *<module_name>*	Specifies a *module_name* to trace. You can use a full or relative path name to identify a *module_name*. To use this option, you must also use the **`-n`** option. There is no default.
`-l all | this`	Tracing is for "all" levels or "this" level. To use this option, you must also use the **`-m`** option. Entering **`all`** traces through all submodules beneath the specified module. Entering **`this`** only traces through the specified module, not its submodules. The default is **`this`**.
`-n` *<net_name>*	Specifies a *net_name* to trace. You can use a full or relative path name to identify a *net_name*. To use this option, you must also use the **`-m`** option.

start	Starts net trace for all nets that you selected via the **select** option. You must execute **ntrace start** before you execute the run command. After running **ntrace start**, you cannot select any more signals until you execute **ntrace stop**.
stop	Stops net trace for all selected signals.

Examples

The **ntrace** command can be used as follows. First, select the signal, then start and stop the signal tracing. Once the ntrace command is executed, simulation needs to be run.

To Trace All Signals in a Module

To select all signals in module UUT for tracing:

```
ntrace select -m UUT
```

To start tracing the signals:

```
ntrace start
```

To run simulation in order to begin the signal trace:

```
run
```

To stop tracing the signals:

```
ntrace stop
```

To Trace All Signals in a Module and its Submodules

To select all signals in module UUT and its submodules for tracing:

```
ntrace select -m UUT -l all
```

To start tracing the signals:

```
ntrace start
```

To run simulation in order to begin the signal trace:

```
run
```

To stop tracing the signals:

```
ntrace stop
```

To Trace a Single Signal

To select the signal Clk in module UUT for tracing:

```
ntrace select -m UUT -n Clk
```

To start tracing the selected signal:

```
ntrace start
```

To run simulation in order to begin the signal trace:

```
run
```

To stop tracing the selected signal:

```
ntrace stop
```

onerror Command

The **onerror** command enables you to control the behavior immediately following a failed Tcl simulation command. See examples for various applications, such as printing the error and resuming the next command.

This command can be used to debug simulation command errors, and is particularly useful when running a Tcl script in which an error is encountered. This command is not intended for users who enter one Tcl command at a time at the Tcl prompt.

Note This command is case sensitive.

The syntax for this command is as follows:

```
onerror{ options }
```

Options

{*list_of_Tcl_commands* }	You can enter a list of simulation Tcl commands to control behavior upon encountering a simulation command error.
{ source *Tcl_script*}	You can source a Tcl script file that contains a list of simulation Tcl commands. These commands control behavior upon encountering a simulation command error.

Examples

In this example, ISim exits after printing the time that the error occurred and all values in the current scope.

```
onerror { showtime;dump;quit -f }
```

In this example, ISim continues reading the next command in the Tcl script after showing the time the error occurred and all values in the current scope.

```
onerror { show time;dump;resume }
```

In this example, ISim reads the sourced Tcl file and executes its command upon encountering an error.

```
onerror { source myerror.tcl }
```

put Command

The **put** command enables you to modify values of signals or buses during simulation. The **put** command can be used to assign:

- a specified signal or bus
- an array of signals/buses
- a record or array of records containing signals/buses
- a value with the specified radix is put to an object

To use the **put** command, the signal or bus must be declared as a signal in your HDL source code.

You cannot use the **put** command to assign a value to a VHDL variable, VHDL generic or Verilog parameter. You can assign values to the whole signal, a bit of a signal or a slice of a signal. You can also access the signal hierarchically. This command can be overridden. The stimulus from your design can override the **put** command. This is why it is temporary.

Note This command is case sensitive.

```
put <signal_name|vhdl_process_name/process_variable_name > [element reference,
element reference, ...]  <value> | <object> <value> -radix <radix_type>
```

Options

<signal_name|vhdl_process_name/process_variable_name> [*element reference, element reference, ...*] *<value>*	*<signal name>* refers to the name of the signal or bus to which you want to assign a value. This can also be the name of an array of signals/buses, an array of records containing signals and buses, or a record of arrays of buses/signals. *<vhdl_process_name/process_variable_name>* is the name of the process and process variable to assign a value. To assign a value to a process variable, you must also supply the name of the process that contains the variable. The process name and process variable names must be separated by "/". [*element reference*] refers to the different ways in which the sub-elements of the signal name can be referenced when used with the **put** command. This provides the ability to further refine the signals by referencing the individual sub-elements of a signal. Please see Examples below for more information. The accepted *value* depends on the signal type as follows. • Integer type can be a positive or negative integer. • A bit_vector can be 0 or 1 or a series of 0s and 1s. • For VHDL, std_logic values can be U, X, 0, 1, etc. • For Verilog, bit_values can be U, X, 0, 1. • Strength values are not supported.
<object> <value> [**-radix** *<radix_type>*]	Takes a value with the specified radix, converts the value to the datatype of the object, and writes the value to the object. *<object>* specifies the signals, buses or objects to modify. *<value>*refers to the value you wish to add to the object. The supported radix types are **default**, **dec**, **bin**, **oct**, **hex**, **unsigned** and **ascii**. When no radix type is specified, the global radix type set with the **isim set radix** command is used, or if none is set there, **default** is used as radix.

Examples

Assign a Value To a Bus or Signal

To assign a value to a signal called `clk`:

```
put clk 1
```

or

```
put clk 1 -radix bin
```

or

```
put clk "1" -radix bin
```

To assign a value to a 4-bit bus called `busx`:

```
put busx 0101
```

To assign a value FF to signal A:

```
put A FF -radix hex
```

To assign a bit value of "1" to a signal called count(6) in the module u1 that is instantiated under your current scope:

```
put u1/count(6) 1
```

Assign a Value To a Standard Logic Vector

For a standard logic vector called `sigx` that is declared as follows:

```
signal sigx :  std_logic_vector(0 to 5);
```

To set bit 0 of `sigx` to "1":

```
put sigx(0) 1
```

To set slice 1 to 2 of `sigx` to "11":

```
put sigx(1:2) 11
```

To set all of `sigx` to "101010":

```
put sigx 101010
```

Assign a Value To an Array of Objects

For an array of standard logic vectors that is declared as follows:

```
signal sigarray :  (0 to 3) vectorarray(0 to 5, 1 to 4, 2 to 6);
```

To set every bit of the array element of `sigarray` to "1":

```
put sigarray(0,1,2)1111
```

To set the first two bits of the array element of `sigarray` to "10":

```
put sigarray(0,1,2)(1:2)10
```

To set bits three of the array element of `sigarray` to "1":

```
put sigarray(0,1,2)(3)1
```

For an array of records which in turn contain an array of standard logic vectors that is declared as follows:

```
type ram_3d_vector is array(0 to 10, 7 downto 0, 0 to 2) of std_logic_vector(1 to 4);

type rectype is record
a: integer;
b: string(1 to 7);
c: std_logic_vector(0 to 3);
d: ram_3d_vector;
end record;

type recarray is array(0 to 3, 4 downto 1) of rectype;

signal recarrsig : recarray;
signal recsig : rectype;
```

To set the second element (*b*) of the record *recsig* to the string "abc":

```
put recsig.b(2:4)abc
```

To set the four bit wide vector represented by the coordinates 2,3,1 in the three dimensional array *d* in the record `recsig` to "0110":

```
put recsig.d(2,3,1) 0110
```

To set the first two bits of the four bit wide vector represented by the coordinates 2,3,1 in the three dimensional array *d* in the record `recsig` to "01":

```
put recsig.d(2,3,1)(1:2) 01
```

To set the four bit wide vector represented by the coordinates 2,3,1 in the three dimensional array *d* in the record `recsig` represented by the coordinates 2,2 in the two dimensional array `recarrsig` to "0011":

```
put recarrsig(2,2).d(2,3,1)0011
```

quit Command

The **quit** command exits either the simulation or the software, depending on the command option. With no option, the **quit** command closes the ISim software after being prompted to do so (for graphic user interface (GUI) mode) or simply closes ISim (in command-line mode).

Note This command is case sensitive.

The syntax for this command is as follows:

quit [*option*]

Options

-f	Quits the current simulation and quits the ISim software. You are not prompted to save the wave configuration even when changes have been made to wave configurations. The switch has no effect at the command line.
-s	Quits the current simulation while keeping the graphical user interface open. At this point, ISim can only be used for loading static wave databases via **File > Open** to open a WDB file. The switch has no effect at the command line.

Examples

To use this command to exit ISim and leave the Tcl prompt, type:

```
quit
```

To use this command to exit ISim and save your waveform:

```
quit -f
```

To use this command to quit the current simulation:

```
quit -s
Quit the current simulation?
Yes
```

restart Command

The **restart** command stops simulation and sets simulation time back to 0. This enables you to start simulation over again within the same simulation run without reloading the design. The equivalent GUI command is Simulation > Restart.

Specifically, **restart** resets the following command settings:

- onerror — Removes onerror script.
- scope — Resets scope to /top.
- saif — Closes file.
- vcd — Closes file.
- isim force — Removes force.
- put — Returns to initial value.

Note This command is case sensitive.

The syntax for this command is as follows:

```
restart
```

Example

To use this command to set simulation time back to 0, and start simulation:

```
restart
```

resume Command

The **resume** command is used with the **onerror** command to continue executing commands after an error is encountered.

Note This command has no effect when entered alone.

Note This command is case sensitive.

```
resume
```

where *no* options are available.

Example

In this example, ISim continues reading the next command in the Tcl script after showing the time the error occurred and all values in the current scope.

```
onerror { show time;dump;resume }
```

run Command

The **run** command starts simulation. With no options, the **run** command runs simulation for 100 ns. The equivalent GUI commands are Simulation > Run All and Simulation > Run.

Note This command is case sensitive.

Syntax

```
run [option]
```

Options

all	Runs simulation until there are no more events or until ISim encounters a breakpoint. See bp Command for information about setting and removing breakpoints.
continue	Resumes simulation after ISim has stopped at a breakpoint. This option is the same as running **run all**.
<time> <unit>	*Time* is the length of time you want simulation to run. It can be any positive integer or decimal. *Unit* is the unit of time. Possible values are **fs**, **ps**, **ns**, **us**, **ms** and **sec**. Default is **ps**.

Examples

The **run** command can be used as follows.

To run simulation until there are no more events or until ISim reaches a breakpoint:

```
run all
```

To run simulation for 2000 nanoseconds:

```
run 2000 ns
```

To run simulation for 1.2 nanoseconds:

```
run 1.2 ns
```

To run simulation for 100 ns.

```
run
```

saif Command

The **saif** command enables you to create a Switching Activity Interchange format (SAIF) file and record port and signal switching rates. See also Writing Activity Data of the Design.

Note This command is case sensitive.

The syntax for this command is as follows:

```
saif <option>
```

Options

open [**-scope** *<path_name>*] [**-file** *<file_name>*] [**-allnets**]	**open** will create a file for SAIF power estimation. **-scope** *<path_name>* creates power estimation data starting at specified scope and recursively. You can use a relative path, or an absolute path. If no path is specified, the current scope is used. **-file** *<file_name>*creates a new SAIF file. The default name of the SAIF file is `xpower.saif`. Only one SAIF file is allowed to be opened during one run of simulation. **-allnets** includes internal nets and port signals in the power estimation. Without this switch, only port signal changes are monitored.
close	Stops monitoring and flushes the SAIF file.
-level *<number_of_levels>*	When **-level 0** is used, signal transitions in all levels below the specified hierarchy are tracked. When **-level 1** is used, only the signal transitions in the specified hierarchy. When **-level 2** is used, signals in two levels of hierarchy are tracked and dumped.

Examples

The **saif** command can be used as follows.

In this example, all ports of a design starting at the current scope recursively are written to a file called `xpower.saif`.

```
saif open
```

In this example, the ports and internal nets of a design starting at the current scope recursively are written to a file called `xpower.saif`.

```
saif open -allnets
```

In this example, ports and internal nets of a design starting at `uut` recursively are written to a file called `uut_backward.saif`.

```
saif open -scope uut -file uut_backward.saif -allnets
```

scope Command

The **scope** command enables you to navigate the design hierarchy. With no option, the **scope** command displays the current module information.

Note This command is case sensitive.

```
scope [option]
```

Options

`..`	Moves the current scope to the hierarchy directly above the current one.
<path_name>	*Path_name* is the path to the module for which you want to display the module information. You can use a relative path, or an absolute path.

Examples

The **scope** command can be used as follows.

To move up one level in the design hierarchy:

```
scope ..
```

To move to the module UUT that is instantiated in the current module:

```
scope UUT
```

To use the scope Tcl command on child instances in a post route netlist.

For example:

```
X_IPAD \CLK/PAD (
.PAD(CLK)
);
```

where, `\CLK/PAD` is an extended identifier.

Enter:

```
scope /testbench/UUT/\\CLK/PAD\
```

Notice that you must use a backslash before `\CLK` and a backslash after `PAD` at the very end.

sdfanno Command

The **sdfanno** command back-annotates VITAL delays from an SDF file to a VHDL design that is made of VITAL-compliant VHDL models. The **sdfanno** command also back-annotates to the timing specified in specify blocks of Verilog modules.

Note This command is case sensitive.

The syntax for this command is as follows.

```
sdfanno (-min | -typ | -max) <file_name> [option]
```

where **-min | -typ | -max** and *file_name* are required, and *option* is optional.

Options

`(-min | -typ | -max)`	A delay switch type (**-min**, **-typ** or **-max**) must be specified in the **sdfanno** command. • **-min** - Annotates VHDL/Verilog file with fastest possible delay values. Gives you a Hold Time timing simulation. • **-typ** - Annotates VHDL/Verilog file with typical delay values. • **-max** - Annotates VHDL/Verilog file with longest possible delay values. Gives you a Setup Time timing simulation.

<file_name>	Name of the SDF file that contains the delay information. A filename must be specified in the **sdfanno** command.
-nowarn	This option turns off warning messages.
-noerror	This option turns error messages into warning messages. This allows you to continue simulation even though there are errors in SDF back-annotation.
-root *<root_path>*	This option specifies the place in the design to apply annotation. The paths specified in the SDF file are relative to the place in the design hierarchy specified by *root_path*. By default, the root is at the top level of the design.

Examples

sdfanno -typ Examples

To annotate the submodule "subdesign" with the typical delay values from `mysubdesign.sdf`:

```
sdfanno -typ mysubdesign.sdf -root /subdesign
```

To annotate the current top-level design with the typical delay values from `design.sdf` and ignore all errors or warnings:

```
sdfanno -typ design.sdf -noerror -nowarn
```

sdfanno -min Example

To annotate the submodule "subdesign" with the minimum delay values from `mysubdesign.sdf`:

```
sdfanno -min mysydesign.sdf -root /subdesign
```

sdfanno -max Example

To annotate the current top-level design with the maximum delay values found in `design.sdf` and ignore warnings:

```
sdfanno -max design.sdf -nowarn
```

show Command

The **show** command displays selected aspects of the design.

Note This command is case sensitive.

show [*option*]

Options

child | **child -r**	Without the **-r** option, **child** displays all of the children blocks (one level only) from the current block. **child -r** option recursively lists all children blocks, including the child processes from the current block
constant	Lists all constants, generics and parameters within the current block.
driver	Displays the processes that are driving the signal specified by *signal_name*. If possible, it prints the line numbers of the HDL code that is contributing to the driver.
load	Displays all of the load for the signal specified by *signal_name*.
port	Displays all of the port signals within the current block. This command identifies signals as inputs or outputs.

scope	Displays the current scope in the design hierarchy. You can only view the hierarchy location; you cannot alter it. This command is identical to the scope command without the *path_name* option.
signal	Displays all signals within the current module including port signals.
time	Displays the simulator current time.
value *<generic_name> | <parameter_name> | <process_name>/<process_variable_name> | <signal_name> [element reference, element reference, ...] | <object>* [**-radix** <radix_type>]	*<generic_name>* is the name of the VHDL generic to be queried. *<parameter_name>* is the name of the VHDL parameter to be queried. *<process_name/process_variable_name>* is the name of the process and process variable to be queried. To query the value of a process variable, you must also supply the name of the process that contains the variable. The process name and process variable names must be separated by "/". *<signal_name>* - Name of the signal or bus to be queried. This can also be the name of an array of signals/buses, an array of records containing signals and buses or a record of arrays of buses/signals. • Use a "." to separate the record hierarchy, as in: **show value recsig.c** • Use a "/" to delimit the hierarchical name of the signal to query a value to a signal on a lower level of hierarchy, as in: **show value mymod/mysig** [*element reference*] refers to the different ways in which the sub-elements of the signal name can be referenced when used with the **show** command. This provides the ability to further refine the signals by referencing the individual sub-elements of a signal. Please see the Examples below for more information. • two integers separated by a colon, enclosed in parenthesis, displays a range of values in the vector specified by *signal_name*. For example: **show value(3:0)** • one or more integers separated by commas, enclosed by parenthesis, displays the values of elements in a multidimensional array specified by *signal_name*. For example: **show value(2,3)** *<object>* [**-radix** *<radix_type>*] displays values with the specified radix. For *<object>*, set the HDL object data type. The supported radix types are **default**, **dec**, **bin**, **oct**, **hex**, **unsigned** and **ascii**. When no radix type is specified, the global radix type set with the **isim set radix command** is used, or if none is set there, **default** is used as radix.
variable	Displays all of the variables within the current block. To see variables within a VHDL process, use **scope** to navigate to the VHDL process and then run **show variable**.

Examples

Show Child

If you are located at the top level of the hierarchy in a design called `fifo_controller` and enter **`show child`**. The following hierarchy information displays:

```
Block Name: <fifo_controller>
```

Enter **`show child -r`** for the same level of the design, the current and recursive hierarchy information displays.

Show Driver

If you are located at the top level of the hierarchy in a design called `fifo_count` and enter **`show driver fifocount`**. The following information displays for the signal `fifocount`:

```
<Driver for fifocount>
   fifoctlr_cc_v2.v:221
```

The number 221 at the end of the lines refers to the code line in the source file.

Show Load

If you are located at the top level of the hierarchy in a design called `fifo_count` and enter **`show load fifocount`**. The following information displays for the signal `fifocount`:

```
<Load for fifocount>
Signal <Hex(0)> (Block: fifo_count/Lsbled/)
Signal <Hex(1)> (Block: fifo_count/Lsbled/)
Signal <Hex(2)> (Block: fifo_count/Lsbled/)
Signal <Hex(3)> (Block: fifo_count/Lsbled/)
```

Show Scope

If you are located at the top level of the hierarchy in a design called `fifo_count` and enter **`show scope`**. The following information displays:

```
<Block> /tb_cc_func/
```

Show Signal Value

To query a value to a signal called *clk*:

`show value clk`

To query a value to a 4-bit bus called *busx*:

`show value busx`

To query the value of *addr*:

`show value addr`

Displays `0111010101011101`.

`show value addr -radix hex`

Displays `755D`.

`show value addr -radix dec`

Displays `30045`.

Show Object Value

The following examples refer to a standard logic vector called `sigx` that is declared as follows:

```
signal sigx :  std_logic_vector(0 to 5);
```

To query bit 0 of `sigx` :

```
show value sigx(0)
```

To query slice 1 to 2 of `sigx`:

```
show value sigx(1:2)
```

To query all of `sigx`:

```
show value sigx
```

Show Value of Array of Objects

The following examples refer to an array of standard logic vectors that is declared as follows:

```
signal sigarray :  vectorarray(0 to 5, 1 to 4, 2 to 6);
```

To query every bit of a vector array element of `sigarray`:

```
show value sigarray(0,1,2)
```

To query the first two bits of each array element of `sigarray`:

```
show value sigarray(0,1,2)(1:2)
```

To query bits three of each array element of `sigarray`:

```
show value sigarray(0,1,2)(3)
```

For an array of records which in turn contains an array of standard logic vectors that is declared as follows:

- type `ram_3d_vector is array(0 to 10, 7 downto 0, 0 to 2) of std_logic_vector(1 to 4);`
- type `rectype is record`
- `a:  integer;`
- `b:  string(1 to 7);`
- `c:  std_logic_vector(0 to 3);`
- `d:  ram_3d_vector;`
- `end record;`
- type `recarray is array(0 to 3, 4 downto 1) of rectype;`
- `signal recarrsig :  recarray;`
- `signal recsig :  rectype;`

To query the second element (`b`) of the record `recsig`:

```
show value recsig.b(2:4)
```

To query the four bit wide vector represented by the coordinates 2,3,1 in the three dimensional array `d` in the record `recsig`:

```
show value recsig.d(2,3,1)
```

To query the first two bits of the four bit wide vector represented by the coordinates 2,3,1 in the three dimensional array `d` in the record `recsig`:

```
show value recsig.d(2,3,1)(1:2)
```

To query the four bit wide vector represented by the coordinates 2,3,1 in the three dimensional array `d` in the record `recsig` represented by the coordinates 2,2 in the two dimensional array `recarrsig`:

```
show value recarrsig(2,2).d(2,3,1)
```

step Command

After you run an initial simulation, you can step through your HDL design one line of source code at a time to verify that the design is working as expected. The **step** command advances to the next line of executable code in the Verilog or VHDL file. The equivalent GUI command is Simulation > Step.

Note This command is case sensitive.

```
step
```

where *no* options are available.

Example

To step through one line of HDL source code:

```
step
```

test Command

The **test** command compares the actual value of a VHDL signal, Verilog wire, Verilog reg, VHDL generic, Verilog parameter or VHDL process variable in the current scope with a supplied value. If the two values match, nothing is displayed. Otherwise the current correct value is displayed, and ISim reports an error. You can test one bit, a slice of a vector element or a whole value.

Note This command is case sensitive.

```
test <signal_name|vhdl_process_name/process_variable_name > {element reference,
element reference, ...} <value> | <object> <value> -radix <radix_type>
```

Options

<signal_name | vhdl_process_name/process_variable_name> {element reference, element reference, ...} <value>	*<signal_name>* refers to the name of the signal or bus to be compared. This can also be the name of an array of signals/buses, an array of records containing signals and buses or a record of arrays of buses/signals. *<vhdl_process_name/process_variable_name>* is the name of the process and process variable to be compared. To compare the value of a process variable, you must also supply the name of the process that contains the variable. The process name and process variable names must be separated by "/". *{element reference}* refers to the different ways in which the sub-elements of the signal name can be referenced when used with the **test** command. This provides the ability to further refine the signals by referencing the individual sub-elements of a signal. Please see Examples below for more information. *<value>* refers to the supplied value to compare with actual value on net or bus.
<object> <value> **-radix** *<radix_type>*	Compares the specified value with the specified radix to object value. *<object>* specifies the signals, buses or objects to test. *<value>* refers to the value you wish to add to the object. The supported radix types are **default**, **dec**, **bin**, **oct**, **hex**, **unsigned** and **ascii**. When no radix type is specified, the global radix type set with the isim set radix command is used, or if none is set there, **default** is used as radix.

Examples

The **test** command can be used as follows.

To compare a bit values of "1" to a signal called `count(6)` in the module `u1` that is instantiated under your current scope:

```
test u1/count(6) 1
```

To compare the value of signal A to "FF":

```
test A FF -radix hex
```

To compare value with "clk" value:

```
test clk 'U'
```

Returns `1`.

To compare value with "clk" value:

```
test clk '0'
```

Returns `0`.

To stop simulation if /top/rst is 0.

```
if {[test /top/rst 0 ] } {stop } else ...
```

For the signal Reset in Block Uut, the command **test Reset 1** displays the following message:

```
test failed
Command failed: test Reset 1
1
Net Reset has value 0 not 1 as expected.
```

For the bus Lsbcnt in Uut, the command **test Lsbcnt 1001** displays the following message:

```
test passed
0
```

For a standard logic vector called `sigx` and declared as follows:

```
signal sigx :  std_logic_vector(0 to 5);
```

To compare bit 0 of `sigx` to "1":

```
test sigx(0) 1
```

To compare slice 1 to 2 of `sigx` to "11":

```
test sigx(1:2) 11
```

To compare all of `sigx` to "101010":

```
test sigx 101010
```

For an array of standard logic vectors that is declared as follows:

```
signal sigarray :  vectorarray(0 to 5, 1 to 4, 2 to 6);
```

To compare every bit of a vector array element of `sigarray` to "1":

```
test sigarray(0,1,2)1111
```

To compare the first two bits of each array element of `sigarray` to "10":

```
test sigarray(0,1,2)(1:2)10
```

To compare bits three of each array element of `sigarray` to "1":

```
test sigarray(0,1,2)(3)1
```

For an array of records that contains an array of standard logic vectors and is declared as follows:

```
type ram_3d_vector is array(0 to 10, 7 downto 0, 0 to 2) of std_logic_vector(1 to 4);

type rectype is record
a: integer;
b: string(1 to 7);
c: std_logic_vector(0 to 3);
d: ram_3d_vector;
end record;

type recarray is array(0 to 3, 4 downto 1) of rectype;

signal recarrsig : recarray;

signal recsig : rectype;
```

To compare the second element (b) of the record `recsig` to the string "abc":

```
test  recsig.b(2:4)abc
```

To compare the four bit wide vector represented by the coordinates 2,3,1 in the three dimensional array `d` in the record `recsig` to "0110":

```
test recsig.d(2,3,1) 0110
```

To compare the first two bits of the four bit wide vector represented by the coordinates 2,3,1 in the three dimensional array `d` in the record `recsig` to "01":

```
test recsig.d(2,3,1)(1:2) 01
```

To compare the four bit wide vector represented by the coordinates 2,3,1 in the three dimensional array `d` in the record `recsig` represented by the coordinates 2,2 in the two dimensional array `recarrsig` to "0011":

```
test recarrsig(2,2).d(2,3,1)0011
```

vcd Command

The **vcd** command generates simulation results in VCD format. This command enables you to dump specified instances to a VCD file, to name the VCD file, to start and stop the dump process, and other functions. See also Writing Activity Data of the Design.

Note This command is case sensitive.

vcd (*option*)

Options

dumpfile <*file_name*>	Gives a name to VCD file. Default file name is `dump.vcd`. Invokes Verilog $dumpfile directive.
dumpvars -m <*module_name*> [**-l** <*level*>]	Dumps the specified variables and their values to the VCD file. **-m** <*module_name*> dumps the module of that name. **-l** **0** - Level 0 causes a dump of all variables in the specified module and in all module instances below the specified module. The argument 0 applies only to subsequent arguments which specify module instances, and no to individual variables. **1** -Level 1 dumps all variable within the module specified by -m; it does not dump variables in any of the modules instantiated by the module specified by -m. Invokes Verilog $dumpfile directive.

dumpoff	Temporarily suspends the dumping process, and dumps all selected variables as an 'X' value. Invokes Verilog $dumpoff directive.
dumpon	Resumes the dumping process after the **dumpoff** option has been invoked. Dumps all selected values at the time **dumpon** is invoked. Invokes Verilog $dumpon directive.
dumpall	Creates a checkpoint in the VCD file that dumps the current value of all selected variables. Invokes Verilog $dumpall directive.
dumplimit *<file_size>*	Limits the size of the VCD file. *file_size* specifies the maximum size of the VCD file in bytes. When the size of VCD file reaches the limit, the dump process stops and a comment is inserted in the VCD file indicating that the dump limit was reached. Invokes Verilog $dumplimit directive.
dumpflush	Empties the operating system's VCD file buffer to ensure that all the data in that buffer is stored in the VCD file. After executing, dump process resumes as before so no value changes are lost. Invokes Verilog $dumpflush directive.

Examples

The **vcd** command can be used as follows.

Following are the commands you would use to write the VCD simulation values of the module UUT to a VCD file after running simulation for 1000 ns.

Specify which file to write:

```
vcd dumpfile adder.vcd
```

Specify which module net activities to write:

```
vcd dumpvars -m /UUT
```

Run simulation for given time:

```
run 1000 ns
```

Dump the activity data to the vcd file.

```
vcd dumpflush
```

wave add Command

The **wave add** command adds simulation objects or blocks to the specified wave configuration in the ISim graphical user interface. The wave configuration displays in the Wave window.

The equivalent GUI command is Add to Wave Window.

Note This command is case sensitive.

Syntax

```
wave add [-wcfg <wcfg_name>] <object_name> [<object_name>] [-r]
```

Options

-wcfg *<wcfg_name>*	Determines to which wave configuration file the signals/objects will be added. If a wave configuration filename is entered that does not already exist, the file will be created. **Note** **wave add -wcfg** is not supported when in batch mode, that is, when launching the simulation executable without the **-gui** switch.
<object_name> [*<object_name>*]	Adds one or more objects to the wave configuration. An object is either a simulation object (VHDL signal, Verilog reg or wire) or block (instance name of a VHDL entity or Verilog Module).
-r	With **-r**, the objects associated with each block are added to the waveform, down to the lowest level of hierarchy. Without **-r**, the first level objects of the block(s) entered as *object_name* are added.

Examples

The **wave add** command can be used as follows.

To add the signals associated with object UUT to the current wave configuration:

```
wave add /tb/UUT
```

To add all top-level signals:

```
wave add /
```

To add all signals in the design:

```
wave add –r /
```

Appendix

Reference

Simulation Executable Commands

Simulation Executable Command Overview

Note The following information is intended for advanced users.

You can run ISim at the command line instead of using the ISE® software or the ISim interface.

Running Simulation From the Command Line

To simulate a Verilog design from the command line, see:

- Running a Functional Simulation From the Command Line With a Verilog Design
- Running a Timing Simulation From the Command Line With a Verilog Design

To simulate a VHDL design from the command line, see:

- Running a Functional Simulation From the Command Line With a VHDL Design
- Running a Timing Simulation From the Command Line With a VHDL Design

Executable Command Summary

There are a number of commands used to execute a simulation at the command line.

Compilation Commands

- **VHDL compiler (vhpcomp) -** The VHDL compiler, vhpcomp, parses VHDL source files for all design units in those files.
- **Verilog compiler (vlogcomp) -** The Verilog compiler, vlogcomp, parses Verilog source files for all design units in those files.

Elaboration Commands

HDL linker (fuse) - The HDL linker, fuse, performs static elaboration of a design in terms of parsed nodes, generates object code for each unique module instance, and links the generated object codes with ISim simulation engine library to create a simulation executable. A user runs this generated simulation executable to effect simulation of the design under consideration.

Simulation Commands

Simulation Executable - The ISE Simulation Executable is generated by the fuse command. To run the simulation of a design in ISim, the generated simulation executable needs to be invoked. When ISim is run inside the ISE interface, ISE takes care of invoking the generated simulation executable. A command-line user needs to explicitly invoke the generated simulation executable to effect simulation. The simulation executable effects event-driven simulation and has rich support for driving and probing simulation using Tcl.

Project Files

A project file can be used to invoke vhpcomp, vlogcomp and fuse. You can collect all of the design files for a particular project into a project file, and use the **-prj** option to specify the project file. The project file can contain VHDL or Verilog files.

Each line in a project file must be formatted as follows:

`<language> [<library>] <filename> {[-d <macro>] [-i <include_path>]}`

where,

<language> is either **vhdl** or **verilog**.

<library> is an optional working library name.

Tcl Commands

The simulation executable provides a powerful Tcl interface that can be run in batch mode. Tcl commands can be used to control the simulator and view the results of simulation. More Info

Invoke the simulation executable using the **-tclbatch** option to list a number of Tcl commands in a batch file.

ISim Simulation Executable

ISim Simulation Executable Overview and Syntax

Note The following information is intended for advanced users.

The ISim simulation executable file is a user-defined executable file. Running the file at the command line invokes an ISim simulation. You can set the executable name using the **-o** option of the fuse Examples. If not user-defined, the default executable name is `x.exe`.

The syntax for this command is as follows:

`<executable_name> .exe (option)`

where *<executable_name>.exe* is user defined or is `x.exe` by default.

Note This command is case sensitive.

ISim Simulation Executable Options

The ISim simulation executable command options are as follows.

Note This command is case sensitive.

-f *<cmd_file>*	You can save ISim engine command options in a text file for future use. This option reads and executes the saved options that are specified in *cmd_file*.
-gui	Launches the ISim graphical user interface.
-h	Displays all command line options and their usage.
-intstyle **ise** | **xflow** | **silent**	Use one of the specified styles for printing messages. Specify **ise** to format messages for the ISE Console Window or **xflow** to format messages for XFLOW. Specify **silent** to suppress all messages.
-log *<file_name>*	Generates a log file with name specified by *file_name*.

-maxdeltaid *<number>*	Specifies the maximum delta number. *Number* is any integer.
-nolog	Blocks log file generation.
-sdfnowarn	Do not display SDF warnings.
-sdfnoerror	Treat errors found in SDF file as warnings.
-sdfmin|-sdftyp|-sdfmax *<root=file>*	Specifies the type of delays ISim should use. • **-sdfmin** - SDF annotates <file> at <root> with minimum delay. • **-sdftyp** - SDF annotates <file> at <root> with typical delay. • **-sdfmax** - SDF annotates <file> at <root> with maximum delay.
-sdfroot *<root_path>*	Sets default place in the design hierarchy where SDF annotation is applied.
-tclbatch *<file_name>*	Turn batch mode on. By default batch mode is off. When batch mode off, Tcl commands can be issued from the Console panel even if the engine is busy simulating. When batch mode is on, all of the commands in the specified batch file are executed sequentially until completion, ignoring any commands entered from the command prompt. *file_name* specifies the name of file containing Tcl commands to be executed. For information about ISim Tcl commands, see Simulation Command Overview.
-testpusarg *<string|stringvalue>*	When the simulator matches this command line argument string with the $test$plusarg or $value$plusarg system function of a Verilog design file, the test or design behavior change associated with the system function is run. *string* is any string. For example, **-testplusarg HELLO**. If a Verilog file uses ($test$plusargs("HE")), the function will return true. *stringvalue* is an appropriate string for the Verilog format specifiers. It provides a value to the variable in the $value$plusargs system function call. For example, **-testplusarg FINISH=10000**. If a Verilog file uses ($value$plusargs("FINISH=%d", stop_clock)) and when Verilog format specifier %d matches 10000, and stop_clock will get value 10000 and the function will return true. The same string or string and value need to be set both in this command line switch and in the system function for the action specified in the Verilog file (such as a value display) to occur.
-vcdfile *<vcd_file>*	Specifies the VCD output file for Verilog projects. Default file name is `dump.vcd`.
-vcdunit *<unit>*	Specifies the VCD output time unit. Possible values are **fs**, **ps**, **ns**, **us**, **ms** and **sec**. Default is **ps**.
-view *<waveform_file.wcfg >*	Used in combination with the **-gui** switch to open the specified waveform file in the ISim graphic user interface.
-wdb *<waveform_database_file.wdb >*	Simulation data is saved to the specified file. For example: **x.exe –wdb my.wdb** saves the simulation data to `my.wdb` instead of the default `isimgui.wdb`.

ISim Simulation Executable Examples

The ISim simulation executable command can be used as follows.

To run the simulation executable named `watchvhd` in batch mode:

```
watchvhd.exe -tclbatch batchfile
```

To run the simulation executable named `watchvhd` with back-annotation using the delay information in the file `adder.sdf`:

```
watchvhd.exe -sdftyp adder.sdf
```

To run the simulation executable interactively:

```
watchvhd.exe
```

To run the simulation in the ISim GUI:

```
watchvhd.exe –gui
```

To get clock to terminate simulation if specified.

A Verilog file contains the following:

```
real frequency;
reg [8*32:1] testname;
integer stop_clock;
if ($value$plusargs("FINISH=%d", stop_clock))
begin
repeat (stop_clock) @(posedge clk);
$finish;
end
```

Enter:

```
x.exe -testplusarg FINISH=10000
```

fuse

fuse Overview and Syntax

Note The following information is intended for advanced users.

The fuse command is the Hardware Description Language (HDL) compiler, elaborator and linker used by the ISim. The command links design units already compiled with vhpcomp or vlogcomp, and creates a simulation executable. It also takes a mixed language project file and compiles the design units on the fly. Name(s) of the top-level design unit(s) must be specified on fuse command line argument. The fuse command effects *static elaboration* on the design given the top design units and then compiles the design units to object code. The design unit object codes are then linked together to create a simulation executable.

- Use one or more top level design unit names as *{[<library_name>.]<top_name>}*. For example, design unit name of the test bench file. If the design uses a Verilog UNISIM or Verilog SIMPRIM library, one top name must be `glbl`. Including a library name is optional. The default library name `work` is assumed when a library name is not specified.
- Use the **`-prj`** option to call vhpcomp or vlogcomp, as appropriate, and to compile your HDL code.
- Use the **`-o`** `<sim_exe>` option to changes the simulation executable file name and location from the default `x.exe`.

The fuse command generates object code and data files for each design unit comprising the design and places them inside `isim/<simulation_executable>.sim` directory.

Note Do no remove the `isim/<simulation_executable>.sim` directory otherwise the design cannot be simulated.

The syntax for this command is as follows:

`fuse` (*`option`*)

where *option* is any option found in fuse Options.

Note This command is case sensitive.

fuse Options

The **fuse** command options are as follows.

-d <*macro_definition*> [= <*value*>]	This option is for Verilog only. Define the macros used in Verilog files, and any value they require. More than one **-d** can be specified. **Note** There should be no space between the "**=**" and the value as this space would be interpreted as part of the value.
-f <*cmd_file*>	You can save fuse command options in a text file for future use. This option reads and executes the saved options, specified in *cmd_file*.
-generic_top "<*parameter*>**=**<*value*>**"**	Overrides generic or parameter of a top level design unit with the specified value. For example, **-generic_top "P=10"** would apply the value of 10 on top level parameter P before elaboration.
-h	Displays all command line options and their usage.
-i <*include_path*>	This option is for Verilog only. Specifies that if fuse calls vlogcomp, it should use the specified path for Verilog 'include directives. Each **-i** can be used for only one include path. More than one **-i** can be specified. Place quotes around paths with spaces.
-incremental	Compiles only the files that have changed since last compile.
-initfile <*sim_init_file*>	Specifies a user defined simulator init file to add to or to override the logical-to-physical mappings of libraries provided by the default `xilinxsim.ini` file.
-intstyle **ise** | **xflow** | **silent**	Use one of the specified styles for printing messages. Specify **ise** to print messages for the ISE Console or **xflow** to print messages for XFLOW. Specify **silent** to suppress all messages. By default all messages are printed.
-ise <*file*>	Enables you to specify a Xilinx ISE file.
-L|**-lib** <*search_lib*> [= <*lib_path*>]	Specifies other libraries and optionally the physical path name for those libraries. Multiple **-L** can be used, and are treated as resource libraries. The physical path provided through **-L** overrides mappings provided by the `xilinxsim.ini` file. *Search_lib* is the logical name of the specified library optionally followed by the *lib_path*, the path to the physical library. For example: **-L mylib=C:home/mylib** For Verilog designs, fuse searches for libraries in the order that the **-L** options are coded. For example: **fuse -L unisim -L abcsim -L xyzsim mytop** First, fuse searches for design units in UNISIM, and then abcsim, and xyzsim. If a design unit was defined in abcsim as well as in xyzsim, the one in abcsim would be used as that appears before xyzsim. If the order was changed to: **fuse -L unisim -L xyzsim -L abcsim mytop**

	And both xyzsim and abcsim defined the same design unit, fuse would pick the design unit from xyzsim.
-maxdelay	This option is for Verilog only. Specifies that if fuse calls vlogcomp, it should use worst case delays.
-maxdesigndepth *<depth>*	Overrides maximum design depth allowed by the elaborator. If a design exceeds the depth, elaborator would error out. May be used to increase the depth in case the elaborator falsely thinks that a design has infinite recursive instantiation.
-mindelay	This option is for Verilog only. Specifies that if fuse calls vlogcomp, it should use fastest possible delays.
-mt *<value>*	Specifies the number of sub-compilation jobs which can be run in parallel. Possible values are **on**, **off**, or an integer greater than 1. Default is on, where the compiler automatically chooses a number based on number of cores in the system.
-nodebug	Generates output that has no information for debugging your HDL code during simulation. Output with no debug information runs simulation faster. The default is to generate HDL units for debugging.
-nospecify	This option is for Verilog only. Disables specify block functionality.
-notimingchecks	This option is for Verilog only. Disables the timing checks.
-o *<sim_exe>*	Specifies the name of the simulation executable output file. The name of the file is *sim_exe*. If you do not use this option, the default executable name is: *work_lib*/*mod_name*/*platform*/**x.exe** where: • *work_lib* is the work library. • *mod_name* is the first top module specified. • *platform* is Windows.
-override_timeprecision	Overrides the time precision (unit of accuracy) of all Verilog modules in the design with the time precision specified in the **-timescale** option.
-override_timeunit	Overrides the time unit (unit of measurement of delays) of all Verilog modules in the design with the time unit specified in the **-timescale** option.
-prj *<prj_file>*.**prj**	Specifies a project file to use for input. A project file contains a list of all the files associated with a design. It is the main source file used by the ISE® software. *Prj_file* is the project file and must have a **prj** extension.
-rangecheck	This option is for VHDL only. Specifies value range check to be performed on VHDL assignments. **Note** This option does not affect index range checking for arrays. ISim *always* checks an index into an array for being within the allowed range. By default **-rangecheck** it is turned off.
-sourcelibdir	Specifies the source directory for library modules. For more information and examples, see Supporting Source Libraries.

-sourcelibext	Specifies the file extension for source files for modules. The **-sourcelibdir** option provides the location for these files. For more information and examples, see Supporting Source Libraries.
-sourcelibfile	Specifies the filename for library modules. For more information and examples, see Supporting Source Libraries.
-timeprecision_vhdl<*time_precision*>	Specifies the time precision (unit of accuracy) for all VHDL design units. The *time_precision* is entered as number (1|10|100|...) followed by unit (fs|ps|ns|us|ms|s). The default is 1ps.
-timescale <*time_unit*/*time_precision*>	Specifies the default timescale for Verilog modules that do not have an effective timescale. The *time_unit* is the unit of measurement of delays. The *time_precision* is the unit of accuracy. Both *time_unit* and *time_precision* are entered as number (1|10|100|...) followed by unit (fs|ps|ns|us|ms|s). The default timescale is 1ns/1ps.
-typdelay	This option is for Verilog only. Specifies that if fuse calls vlogcomp, it should use typical delays.
-v <*value*>	Specifies the verbosity level for printing messages. Allowed values are 0, 1, or 2. The default is 0. fuse -v 1 prints useful debugging information, which can help to identify problems in ISim compilers. • Dumps the library mapping as seen by ISim compiler after reading all available library mapping files (`xilinxsim.ini`) • Gets verbose messages from design elaborator • Gets the dumps of current values of environment variables which affect the behavior of the ISim compiler • Gets the list of loaded shared objects by the ISim compiler • Dumps operating system information, including version number and processor • Dumps path to GCC compiler being used to compile the generated code
-version	Prints the compiler version.

fuse Examples

Using Precompiled HDL

The following examples show how to invoke fuse using precompiled HDL.

For VHDL using a top level configuration:

```
fuse work.yourtop
```

For Verilog or mixed language design using all of the top level modules:

```
fuse work.top_level_module_name_1 work.top_level_module_name_2 work.glbl -L
simprims_ver -L logicalLib1 -o mysim.exe
```

Using HDL Source

Example of fuse invoked using source VHDL

This example produces an executable called `tb.exe` from VHDL source listed in the project file called `x.prj`. The contents of the project file are as follows:

```
VHDL work x1.vhd
VHDL work x2.vhd
VHDL work x3.vhd
VHDL work tb.vhd
```

```
fuse -prj x.prj work.tb_top -o tb.exe
```

To start simulation, execute the following:

```
tb.exe
```

Example of fuse invoked using source Verilog

This example produces an executable called `tb.exe` from Verilog source listed in the project file called `myproj.prj`. The top level design unit is *tb* defined in file `tb.v`. The contents of `myproj.prj` are as follows:

```
Verilog work x1.v
Verilog work x2.v
Verilog work x3.v
Verilog work tb.v
```

Use the following command to run fuse:

```
fuse -prj myproj.prj work.tb work.glbl -o tb.exe
```

To start simulation, execute the following:

```
tb.exe
```

Note For Verilog, if the design instantiates any modules that have been compiled into any libraries other than the *work* library, those libraries must be passed to fuse using the **`-L`** command line option so that fuse will find those design units and link them into the simulation executable.

Example of fuse invoked using mixed VHDL/Verilog design

This example produces an executable called `tb.exe` from Verilog and VHDL source code listed in the project file called `myproj.prj`. There are two top level design units: a VHDL top level called *tb* and a Verilog top level unit called *glbl* defined in `tb.vhd` and `glbl.v` respectively. The contents of `myproj.prj` are as follows:

```
Verilog work x1.v
VHDL work x2.vhd
Verilog work x3.v
VHDL work x4.vhd
Verilog work glbl.v
VHDL work tb.vhd
```

Use the following command to run fuse:

```
fuse work.tb work.glbl -prj x.prj -o tb.exe
```

To start simulation, execute the following:

```
tb.exe
```

Note For Mixed language designs, for the modules on language boundary and for the Verilog modules that have been compiled into any libraries other than the *work* library, those libraries must be passed to fuse using the **`-L`** command line option in the desired search order so that fuse will find those design units and link them into the simulation executable. For more information, see Mixed Language Simulation Overview.

Using the Command File Option

The following example shows how to invoke fuse using the **`-f`** option:

```
fuse -f my_design.cmd
```

The following is an example of a command file for Verilog:

```
-nodebug
-intstyle xflow
-incremental
top_level_module_name_1
top_level_module_name_n
-L logicalLib1
-L logicalLib2
```

vlogcomp

vlogcomp Overview and Syntax

Note The following information is intended for advanced users.

The Verilog compiler, vlogcomp, is used by the ISim to parse Verilog source files and generate object code for all design units in those files. The object code generated by vlogcomp is used by fuse to create a simulation executable.
You must specify either a project file or one or more Verilog source files to compile. If neither the project file nor the Verilog file is specified, vlogcomp issues an error.

The syntax for this command is as follows:

`vlogcomp` (*`option`*)

where *option* is any option found in vlogcomp Options.

Note This command is case sensitive.

vlogcomp Options

The **`vlogcomp`** command options are as follows.

<verilog_file>	Specifies the Verilog file to be compiled.
-d *<macro_definition>* [=*<value>*]	Define the macros used in Verilog files, and any value they require. More than one **-d** can be specified
-f *<cmd_file>*	You can save vlogcomp command options in a text file for future use. This option reads and executes the saved options, specified in *cmd_file*.
-h	Displays all command line options and their usage.
-i *<include_path>*	Specifies path for Verilog 'include directives. More than one **-i** can be specified.
-incremental	Compiles only the files that have changed since last compile.
-initfile<*sim_init_file*>	Specifies the physical path to user defined simulator init file instead of default `xilinxsim.ini` file.
-intstyle **ise |xflow |silent**	Use one of the specified styles for printing messages. Specify **ise** to print messages for the ISE Console tab or **xflow** to print messages for XFLOW. Specify **silent** to suppress all messages. By default all messages are printed.
-ise *<file>*	Enables you to specify a Xilinx ISE file.

-L|-lib <*search_lib*> [=<*lib_path*>]	Specifies other libraries and optionally the physical path name for those libraries. Multiple **-lib** options can be used, and are treated as resource libraries. The physical path provided through **-lib** overrides the mappings provided by the `xilinxsim.ini` file. *Search_lib* is the logical name of the specified library optionally followed by the *lib_path*, the path to the physical library. For example: **-lib mylib=C:/home/mylib**
-maxdelay	Specifies that vlogcomp should use worst case delays.
-mindelay	Specifies that vlogcomp should use fastest possible delays.
-nodebug	Generates output that has no information for debugging your HDL code during simulation. Output with no debug information runs simulation faster. Default is to generate HDL debug units.
-nospecify	Ignores Verilog path delays and timing checks.
-notimingchecks	Ignores timing check constructs in Verilog specify blocks.
-prj <*prj_file*> **.prj**	Specifies a project file to use for input. A project file contains a list of all the files associated with a design. It is the main source file used by the ISE® software. *prj_file* is the path to the project file and the project file name with a **.prj** extension. You can include an absolute or relative path to the project file. For a relative path, include ../ as part of the path.
-sourcelibdir	Specifies the source directory for library modules. For more information and examples, see Supporting Source Libraries.
-sourcelibext	Specifies the file extension for source files for modules. The **–sourcelibdir** option provides the location for these files. For more information and examples, see Supporting Source Libraries.
-sourcelibfile	Specifies the filename for library modules. For more information and examples, see Supporting Source Libraries.
-typdelay	Specifies that vlogcomp should use typical delays.
-v <*value*>	Specifies the verbosity level for printing messages. Allowed values are 0, 1, or 2. The default is 0.
-version	Prints the compiler version.
-work <*work_lib*> [= <*lib_path*>]	Specifies the **work** library, and optionally, the physical path for the work library. The physical path provided through this option overrides mappings provided by the `xilinxsim.ini` file. The default work library is the logical library **work**. *Work_lib* is the logical name of the specified work library optionally followed by *lib_path*, the path to the physical library. For example: **mywork=C:/home/worklib**

vlogcomp Examples

The **vlogcomp** command can be used as follows.

To invoke the Verilog compiler with all options saved in a file called `run32.txt`:

```
vlogcomp -f run32.txt
```

To invoke the Verilog compiler using a work library with the logical name mysimwork located in the `/home/smithjj/mylib` directory, and compile all the Verilog files in the project file `dsp64.prj`:

```
vlogcomp -work mysimwork=/home/smithjj/mylib -prj dsp64.prj
```

To invoke the Verilog compiler using the default work library specified in the `xilinxsim.ini` file and compile the Verilog files `suba.v` and `subb.v`:

```
vlogcomp suba.v subb.v
```

vhpcomp

vhpcomp Overview and Syntax

Note The following information is intended for advanced users.

The VHDL compiler, vhpcomp, is used by ISim to parses VHDL source files and generate object code for all design units in those files. The object code generated by vhpcomp is used by fuse to create a simulation executable.

You must specify either a project file or one or more VHDL files to compile. If neither project file nor VHDL file are specified, vhpcomp issues an error.

The syntax for this command is as follows:

`vhpcomp` (*`option`*)

where *option* is any option found in vhpcomp Options.

Note This command is case sensitive.

vhpcomp Options

The **`vhpcomp`** command options are as follows.

<vhdl_file>	Specifies one or more VHDL source files to be compiled.
`-f` *<cmd_file>*	You can save vhpcomp command options in a text file for future use. This option reads and executes the saved options, specified in *cmd_file*.
`-h`	Displays all command line options and their usage.
`-incremental`	Compiles only the files that have changed since last compile.
`-intstyle` **`ise`** | **`xflow`** | **`silent`**	Use one of the specified styles for printing messages. Specify **`ise`** to print messages for the ISE® software Console tab or **`xflow`** to print messages for XFLOW. Specify **`silent`** to suppress all messages. By default all messages are printed.
`-ise` *<file>*	Enables you to specify a Xilinx ISE file.
`-L`|**`-lib`** *<search_lib>* [=*<lib_path>*]	Specifies other libraries and optionally the physical path name for those libraries. Multiple **`-lib`** options can be used, and are treated as resource libraries. The physical path provided through **`-lib`** overrides the mappings provided by the `xilinxsim.ini` file. *Search_lib* is the logical name of the specified library optionally followed by the *lib_path*, the path to the physical library. For example: **`-lib mylib=C:/home/mylib`**

-nodebug	Generates output that has no information for debugging your HDL code during simulation. Output with no debug information results in a faster simulation runtime. Default is to generate HDL debug units.
-prj *<prj_file>* **.prj**	Specifies a project file to use for input. A project file contains a list of all the files associated with a design. It is the main source file used by the ISE software. *Prj_file* is the path to the project file and the project file name with a **.prj** extension. You can include an absolute or relative path to the project file. For a relative path, include ../ as part of the path.
-rangecheck	Enables run time value range check (VHDL only). This option causes vhpcomp to generate output that checks that values assigned to VHDL signals are within their valid range. For example, if a signal is declared as *positive*, fuse will check that the signal is not assigned a negative number. Or if a signal is declared as std_logic, vhpcomp generates output to check that the signal is assigned only valid std_logic values (U,X,0,1,Z,W,L,H,-). **Note** This option does not affect the checking of index ranges. ISim *always* checks the ranges of indexes. By default **-rangecheck** is turned off.
-v *<value>*	Specifies the verbosity level for printing messages. Allowed values are 0, 1, or 2. The default is 0.
-work *<work_lib>* [=*<lib_path>*]	Specifies the work library, and optionally, the physical path for the work library. The physical path provided through this option overrides the mappings provided by the `xilinxsim.ini` file. The default work library is the logical library **work**. *Work_lib* is the logical name of the specified work library optionally followed by *lib_path*, the path to the physical library. For example: **mywork=C:/home/worklib**

vhpcomp Examples

The **vhpcomp** command can be used as follows.

To invoke the VHDL compiler with all options saved in a file called **run32.txt**:

```
vhpcomp -f run32.txt
```

To invoke the VHDL compiler using a work library with the logical name `mysimwork` located in the `/home/smithjj/mylib` directory, and compile all the VHDL files in the project file `dsp64.prj`:

```
vhpcomp -work mysimwork=/home/smithjj/mylib -prj dsp64.prj
```

To invoke the VHDL compiler using the default work library specified in the `xilinxsim.ini` file and compile the VHDL files `suba.vhd` and `subb.vhd`:

```
vhpcomp suba.vhd subb.vhd
```

Third-Party Command Equivalency

Third-Party Simulation Command Support Overview

ISim does not support third-party commands. If simulation commands in a DO (*.do) file are used in ISim, ISim will not understand the commands unless there is an exact ISim equivalent command, such as **run**).

You can map your DO file commands to ISim commands using the information below:

- Third-Party Compiler Commands
- Third-Party Tcl Commands

Third-Party Compiler Commands

Use the following table as guidance when mapping your DO file compilation commands to ISim commands.

Compiler Command Compatibility Table

DO File Command	ISim Command	Remarks
vcom -work <*libname*> **-93** <*file_name*>	**vhpcomp** <*file_name*>	Compiles VHDL file.
vlog -work <*libname*> <*file_name*>	**vlogcomp** <*file_name*>	Compiles verilog file.
vsim <*lib_name*>**.**<*design_name*>	**fuse -lib** <*lib_name*> <*design_name*>	Builds a simulation executable.
vsim <*lib_name*>**.**<*design_name*> <mti.do>	<*executable_name*>*.exe* **-tclbatch** <*design_name*>**.tclbatch**	Runs simulation.
vsim [**-sdfmin** | **-sdftyp** | **-sdfmax**] [<*instance*>**=**]<*sdf_filename*>] [**-sdfnoerror**] [**-sdfnowarn**] [**+sdf_verbose**] sdf commands can only be invoked as command arguments to vsim commands.	**sdfanno** {**-min|-typ|-max**} <*file_name*> [**-nowarn**] [**-noerror**] [**-root**<*path_name*>]	SDF annotation.

Third-Party Tcl Commands

Use the following table as guidance when mapping your DO file simulation commands to ISim commands.

Simulation Tcl Command Compatibility Table

DO File Command	ISim Command	Remarks
add [**wave**] **-recursive <item_name>** [**-ports** | **-in** | **-out** | **-inout**] or delete command selects the nets to be traced.	**wave add** or **ntrace select -o** {**on** | **off**} **-m** <*module_name*> **-l** {**all** | **this**} [**-n** <*net_name*>]	Selects modules and signals for tracing during simulation. **-o** turns tracing on or off on a specified module or signal. This command is for GUI signal trace only. Ignored options: All GUI options associated with **add wave**, such as **-divider** and **-scale**.
bd *id#* 1 or more can be specified.	**bp del** <*index*>[<*index*> ...]	Removes breakpoint based on the index where the *index* is the index number assigned to the breakpoint you want to delete. Each breakpoint in your design is assigned a unique index number.
bd<*file_name*> <*line_number*>	**bp remove** <*file_name*> <*line_number*>	Removes break point at <*line_number*> in <*file_name*>.
bd<*file_name*> <*line_number*> | <*id#*>	**bp remove** <*file_name*> <*line_number*>	Removes breakpoint at <*line_number*> in <*file_name*>.
bp <*file_name*> <*line_number*>	**bp add** <*file_name*> <*line_number*>	Adds breakpoint at <*line_number*> in <*file_name*>. Ignored options: [**-id** <*id#*>], [**-inst** <*region*>], [**-cond** {<*condition_expression*>}]

DO File Command	ISim Command	Remarks
bp -query *<file_name>*	**bp list**	Lists all breakpoints.
delete wave	**ntrace -o off**	Turns tracing off for a specified module or signal.
drivers *<item name>*	**show driver***<net_name>*	Shows all the drivers that are driving the specified *<net_name>*.
env	**scope**	Displays where you are in the design hierarchy.
env ..	**scope ..**	Changes to the parent of current scope.
env *<pathname>*	**scope** *path_name*	Changes to the scope specified by *<path_name>* Ignored options: **-nodataset**, **-dataset**.
examine*<signal_name>*	**show value** *<signal_name>*	Shows the value of a signal.
exit	**exit**	Exits the simulator.
force -deposit *<signal_name>* *<value>* [*<time>*]	**put**	Changes the value to new value, but the new value gets overwritten by assignments made in HDL.
force -freeze	**isim force add**	Overrides all assignments done from HDL, and makes the signal/wire truly stuck or frozen at a particular value.
help	**help**	Displays all Tcl commands and their usage.
help [*command* | *topic*]	**help** *<command>*	Displays help info on a command.
if { [**exa sig_a**] **== "0011ZZ"**} {**echo"Signal value matches"**}	**test** *<signal_name>* *<value>*	Tests the signal value display if it is different.
noforce *<signal>*	**isim force remove** *<signal>*	Removes a value on a signal. This command forces a value on the signal, until isim force remove is issued. This command works only for VHDL signals and Verilog wire types. It does not work for Verilog regs.
quit	**quit**	Exits Tcl prompt. Ignored options: [**-f** | **-force**] [**-sim**].
radix	**isim get radix**	Returns the default radix as a string in Tcl result variable, and displays the default radix to stdout.
radix -*<radix_type>*	**isim get radix** *<radix_type>*	Sets the global radix for the current simulation.
restart	**restart**	Stops simulation and sets simulation time back to 0.
run *<length>* *<unit>*	**run** *<length>* *<unit>*	Runs simulation for *<length>* *<unit>* time.
run -all|-continue	**run** {**all** | **continue**}	Runs simulation until there are no more events.

DO File Command	ISim Command	Remarks
run for time steps and time units **Note** Also set with the *RunLength* and *UserTimeUnit* variables in the `modelsim.ini` file.	**run**	Runs simulation for 100 ns.
show	**show scope**	Shows current scope in the design hierarchy.
show	**show signal**	Shows all signals and port signals within the current block.
show -all	**show child -r**	Recursively shows all children blocks of current block.
vcd add -r	**vcd dumpvars -m** <*module instance*> **-l** <*int*>	Dumps specific instance with level. Ignored options: [**-in**] [**-out**] [**-inout**] [**-internal**] [**-ports**].
vcd file <*file_name*>	**vcd dumpfile** <*file_name*>	Writes output to <*file_name*>. Ignored options: **-dumpports**, **-map**.
vcd flush	**vcd dumpflush**	Flushes vcd data to file. No support for <*file_name*>. The file specified by vcd dumpfile" will be flushed.
vcd limit <*size*>	**vcd dumplimit** <*no_of_bytes*>	Limits var dump size.
vcd on | **off** [<*file_name*>]	**vcd** {**dumpon** | **dumpoff**}	Turns vcd tracing on or off.

HDL Language Support

VHDL Language Support (a to m)

ISim supports the following VHDL constructs, with exceptions noted. The constructs are listed in alphabetical order. See VHDL Language Support (n to z) for the second half of the list.

Supported VHDL Construct	Exceptions
abstract_literal	Floating point expressed as based literals are unsupported.
access_type_definition	
actual_designator	
actual_parameter_part	
actual_part	
adding_operator	
aggregate	Mixing choice directions in an aggregate not supported.

Supported VHDL Construct	Exceptions
alias_declaration	Alias to non-objects are in general not supported, in particular following are not supported: • Alias of an alias is not supported. • Alias declaration without subtype_indication is not supported. • Signature on alias declarations is not supported. • Operator symbol as alias_designator is not supported. • Alias of an operator symbol is not supported. • Character literals as alias designators are not supported.
alias_designator	• Operator_symbol as alias_designator is not supported. • Character_literal as alias_designator is not supported.
allocator	
architecture_body	
architecture_declarative_part	
architecture_statement_part	
array_type_definition	
assertion	
assertion_statement	
association_element	Now, globally locally static range is acceptable for taking slice of an actual in an association element. Note that non locally static index/slice in formal is an error and we detect and report so. However, a formal name with an index/slice/selection is not supported if the prefix of the formal name is also indexed/sliced/selected.
association_list	
attribute_declaration	
attribute_designator	
attribute_name	Signature after prefix is not supported. The following predefined attributes are supported: • A'ACTIVE, A'ASCENDING([N]), A'HIGH([N]), A'LENGTH([N]), A'LEFT([N]), A'LOW([N]), A'RANGE([N]), A'REVERCE_RANGE([N]), A'RIGHT([N]) • S'DELAYED[(T)], S'EVENT, S'LAST_ACTIVE, S'LAST_EVENT, S'LAST_VALUE • T'ASCENDING, T'BASE, T'HIGH, T'IMAGE(X), T'LEFT, T'LEFTOF(X), T'LOW, T'POS(X), T'PRED(X), T'RIGHT, T'RIGHTOF(X), T'SUCC(X), T'VAL(X)
attribute_specification	
base	
base_specifier	
base_unit_declaration	
based_integer	
based_literal	

Supported VHDL Construct	Exceptions
basic_character	
basic_graphic_character	
basic_identifier	
binding_indication	Binding_indication without use of entity_aspect is not supported.
bit_string_literal	Empty bit_string_literal ("") is not supported.
bit_value	
block_configuration	
block_declarative_item	
block_declarative_part	
block_header	
block_specification	
block_statement	Guard_expression is not supported; for example, guarded blocks, guarded signals, guarded targets, and guarded assignments are not supported.
case_statement	
case_statement_alternative	
character_literal	
choice	Aggregate used as choice in case statement is unsupported.
choices	
component_configuration	
component_declaration	
component_instantiation_statement	
component_specification	
composite_type_definition	
concurrent_assertion_statement	Postponed is not supported.
concurrent_procedure_call_statement	Postponed is not supported.
concurrent_signal_assignment_statement	Postponed is not supported.
concurrent_statement	Concurrent procedure call containing wait statement is not supported.
condition	
condition_clause	
onditional_signal_assignment	Keyword **guarded** as part of options is not supported as there is no supported for guarded signal assignment.
conditional_waveform	
configuration_declaration	Non locally static for generate index used in configuration is unsupported.
configuration_declarative_item	
configuration_declarative_part	
configuration_item	

Supported VHDL Construct	Exceptions
configuration_specification	
constant_declaration	
constrained_array_definition	
constraint	
context_clause	
context_item	
decimal_literal	
declaration	
delay_mechanism	
design_file	
design_unit	
designator	
direction	
discrete_range	
element_association	
element_declaration	
element_subtype_definition	
entity_aspect	
entity_class	Literals, unit, file and group as entity class are not supported
entity_class_entry	Optional <> intended for use with group templates is not supported.
entity_class_entry_list	
entity_declaration	
entity_declarative_item	
entity_declarative_part	
entity_designator	
entity_header	
entity_name_list	
entity_specification	
entity_tag	
enumeration_literal	
enumeration_type_definition	
exit_statement	
exponent	
expression	
extended_digit	
extended_identifier	
factor	
file_declaration	

Supported VHDL Construct	Exceptions
file_logical_name	Although file_logical_name is allowed to be any wild expression evaluating to a string value, only string literal and identifier is acceptable as file name.
file_open_information	
file_type_definition	
floating_type_definition	
formal_designator	
formal_parameter_list	
formal_part	
full_type_declaration	
function_call	In named parameter association in a function_call slicing, indexing or selection of formals is unsupported.
generate_statement	
generate_scheme	
generic_clause	
generic_list	
generic_map_aspect	
graphic_character	
identifier	
identifier_list	
if_statement	
incomplete_type_declaration	
index_constraint	
index_specification	
index_subtype_definition	
indexed_name	
instantiated_unit	Direct configuration instantiation is unsupported.
instantiation_list	
integer	
integer_type_definition	
interface_constant_declaration	
interface_declaration	
interface_element	
interace_file_declaration	
interface_list	
interface_signal_declaration	
interafce_variable_declaration	
iteration_scheme	
label	

Supported VHDL Construct	Exceptions
letter	
letter_or_digit	
library_clause	
library_unit	
literal	
logical_name	
logical_name_list	
logical_operator	
loop_statement	
miscellaneous_operator	
mode	Linkage and Buffer ports are not supported completely.
multiplying_operator	

VHDL Language Support (n to z)

ISim supports the following VHDL constructs, with exceptions noted. The constructs are listed in alphabetical order. See VHDL Language Support (a to m) for the first half of the list.

Supported VHDL Construct	Exceptions
name	
next_statment	
numeirc_literal	
object_declaration	
operator_symbol	
options	Guarded is not supported.
package_body	
package_body_declarative_item	
package_body_declarative_part	
package_declaration	
package_declaration_item	
package_declarative_part	
parameter_specification	
physical_literal	
physical_type_definition	
port_clause	
port_list	
port_map_aspect	
prefix	
primary	At places where primary is used, allocator is expanded there.
primary_unit	

Supported VHDL Construct	Exceptions
procedure_call	In named parameter association in a procedure_call slicing, indexing or selection of formals is unsupported.
procedure_call_statement	
process_declarative_item	
process_declarative_part	
process_statement	Postponed processes are not supported.
process_statement_part	
qualified_expression	
range	
range_constraint	
record_type_definition	
relation	
relational_operator	
report_statement	
return_statement	
scalar_type_definition	
secondary_unit	
secondary_unit_declaration	
selected_name	
selected_signal_assignment	The "guarded" keyword as part of options is not supported as there is no support for guarded signal assignment.
selected_waveform	
sensitivity_clause	
sensitivity_list	
sequence_of_statements	
sequential_statement	
shift_expression	
shift_operator	
sign	
signal_assignment_statement	
signal_declaration	Signal_kind is not supported. Signal_kind is used for declaring guarded signals, which are not supported.
signal_list	
signature	
simple_expression	
simple_name	
slice_name	
string_literal	
subprogram_body	

Supported VHDL Construct	Exceptions
subprogram_declaration	
subprogram_declarative_item	
subprogram_declarative_part	
subprogram_kind	
subprogram_specification	
subprogram_statement_part	
subtype_declaration	
subtype_indication	Resolved subtype of composites (arrays and records) is not supported.
suffix	
target	
term	
timeout_clause	
type_conversion	
type_declaration	
type_definition	
type_mark	
unconstrained_array_defintion	
use_clause	
variable_assignment_statement	
variable_declaration	
wait_statement	
waveform	**Unaffected** is not supported.
waveform_element	Null waveform element is unsupported as it only has relevance in the context of guarded signals.

Verilog Language Coverage

Verilog Language Support

The ISim supports Verilog-2001. To determine if a particular Verilog construct is supported, go to the appropriate construct category.

- Behavioral Statements
- Compiler Directives
- Declarations
- Expressions
- General
- Primitive and Module Instances
- Source Text
- Specify Functions
- System Tasks and Functions
- UDP Declarations and Instances

Behavioral Statement Constructs

The following Verilog Behavioral Statements constructs are supported by the ISim as defined below.

Continuous Assignment Statements

Verilog Construct	ISim Support
continuous_assign	supported
list_of_net_assignments	supported
net_assignment	supported

Procedural Blocks and Assignments

Verilog Construct	ISim Support
initial_construct	supported
always_construct	supported
blocking_assignment	supported
nonblocking_assignment	supported
procedural_continuous_assignment	supported
function_blocking_assignment	supported
function_statement_or_null	supported

Parallel and Sequential Bloc

Verilog Construct	ISim Support	Comment
function_seq_block	supported	
variable_assignment	supported	
par_block	partially supported	*Fork* and *join* constructs are not supported within tasks or functions
seq_block	supported	

Statements

Verilog Construct	ISim Support
statement	supported
statement_or_null	supported
function_statement	supported

Timing Control Statements

Verilog Construct	ISim Support
delay_control	supported
delay_or_event_control	supported
disable_statement	supported
event_control	supported
event_trigger	supported
event_expression	supported
procedural_timing_control_statement	supported
wait_statement	supported

Conditional Statements

Verilog Construct	ISim Support
conditional_statement	supported
if_else_if_statement	supported
function_conditional_statement	supported
function_if_else_if_statement	supported

Case Statements

Verilog Construct	ISim Support
case_statement	supported
case_item	supported
function_case_statement	supported
function_case_item	supported

Looping Statements

Verilog Construct	ISim Support
function_loop_statement	supported
loop_statement	supported

Task Enable Statements

Verilog Construct	ISim Support
system_task_enable	supported
task_enable	supported

Compiler Directive Constructs

The following Verilog Compiler Directives constructs are supported by the ISim as defined below.

Compiler Directive Constructs

Verilog Construct	ISim Support	Comment
‘celldefine	unsupported	
‘endcelldefine	unsupported	
‘default_nettype	supported	
‘define	supported	
‘undef	supported	ISim now supports parameterized ‘define macros.
‘ifdef	supported	
‘ifndef	supported	
‘elsif	supported	
‘else	supported	
‘endif	supported	
‘include	supported	
‘resetall	supported	
‘line	supported	
‘timescale	supported	
‘unconnected_drive	unsupported	
‘nounconnected_driv	unsupported	

Declaration Constructs

The following Verilog Declaration constructs are supported by the ISim as defined below.

Module Parameter Declarations

Verilog Construct	ISim Support
local_parameter_declaration	supported
parameter_declaration	supported
specparam_declaration	supported

Type Declarations

Verilog Construct	ISim Support
event_declaration	supported
genvar_declaration	supported
integer_declaration	supported
net_declaration	supported
real_declaration	supported
reg_declaration	supported
time_declaration	supported

Net Variable Types

Verilog Construct	ISim Support
net_type	supported
output_variable_type	supported
real_type	supported
variable_type	supported

Strengths

Verilog Construct	ISim Support
drive_strength	supported
strength0	supported
strength1	supported
charge_strength	unsupported

Delays

Verilog Construct	ISim Support
delay2	supported
delay3	supported
delay_value	supported

Declaration Lists

Verilog Construct	ISim Support
list_of_event_identifiers	supported
list_of_genvar_identifiers	supported
list_of_net_decl_assignments	supported
list_of_net_identifiers	supported
list_of_param_assignments	supported
list_of_port_identifiers	supported
list_of_real_identifiers	supported
list_of_specparam_assignments	supported
list_of_variable_identifiers	supported
list_of_variable_port_identifiers	supported

Declaration Assignments

Verilog Construct	ISim Support
net_decl_assignment	supported
param_assignment	supported
specparam_assignment	supported
pulse_control_specparam	supported
error_limit_value	supported
reject_limit_value	supported
limit_value	supported

Declaration Ranges

Verilog Construct	ISim Support
dimension	supported
range	supported

Function Declarations

Verilog Construct	ISim Support
function_declaration	supported
function_item_declaration	supported
function_port_list	supported
range_or_type	supported

Task Declarations

Verilog Construct	ISim Support
task_declaration	supported
task_item_declaration	supported
task_port_list	supported
task_port_item	supported
tf_input_declaration	supported
tf_output_declaration	supported
tf_inout_declaration	supported
task_port_type	supported

Block Item Declarations

Verilog Construct	ISim Support
block_item_declaration	supported
block_reg_declaration	supported
list_of_block_variable_identifiers	supported
block_variable_type	supported

Expression Constructs

The following Verilog Expression constructs are supported by the ISim as defined below.

Concatenations

Verilog Construct	ISim Support
concatenation	supported
constant_concatenation	supported
constant_multiple_concatenation	supported
module_path_concatenation	supported
module_path_multiple_concatenation	supported
multiple_concatenation	supported
net_concatenation	supported
net_concatenation_value	supported
variable_concatenation	supported
variable_concatenation_value	supported

Function Calls

Verilog Construct	ISim Support
constant_function_call	supported
function_call	supported
system_function_call	supported

Expressions

Verilog Construct	ISim Support
base_expression	supported
conditional_expression	supported
constant_base_expression	supported
constant_expression	supported
constant_mintypmax_expression	supported
constant_range_expression	supported
dimension_constant_expression	supported
expression1	supported
expression2	supported
expression3	supported
expression	supported
lsb_constant_expression	supported
mintypmax_expression	supported
module_path_conditional_expression	supported
module_path_expression	supported
module_path_mintypmax_expression	supported
msb_constant_expression	supported
range_expression	supported
width_constant_expression	supported

Primaries

Verilog Construct	ISim Support
constant_primary	supported
module_path_primary	supported
primary	supported

Expresson Left-Side Values

Verilog Construct	ISim Support
net_lvalue	supported
variable_lvalue	supported

Operators

Verilog Construct	ISim Support
unary_operator	supported
binary_operator	supported
unary_module_path_operator	supported
binary_module_path_operator	supported

Numbers

Verilog Construct	ISim Support
number	supported
real_number	supported
exp	supported
decimal_number	supported
binary_number	supported
octal_number	supported
hex_number	supported
sign	supported
size	supported
non_zero_unsigned_number	supported
unsigned_number	supported
binary_value	supported
octal_value	supported
hex_value	supported
decimal_base	supported
binary_base	supported
octal_base	supported
hex_base	supported
non_zero_decimal_digit	supported
decimal_digit	supported
binary_digit	supported
octal_digit	supported
hex_digit	supported
x_digit	supported
z_digit	supported

Strings

Verilog Construct	ISim Support
string	supported

General Constructs

The following Verilog general and miscellaneous constructs are support by the ISim as defined below.

Attributes

Verilog Construct	ISim Support
attribute_instance	unsupported
attr_spec	unsupported
attr_name	unsupported

Comments

Verilog Construct	ISim Support
comment	supported
one_line_comment	supported
block_comment	supported
comment_text	supported

Identifiers

Verilog Construct	ISim Support
arrayed_identifier	supported
block_identifier	supported
cell_identifier	unsupported
config_identifier	unsupported
escaped_arrayed_identifier	supported
escaped_hierarchical_identifier	supported
escaped_identifier	supported
event_identifier	supported
function_identifier	supported
gate_instance_identifier	supported
generate_block_identifier	supported
genvar_identifier	supported
genvar_function_identifier	unsupported
hierarchical_block_identifier	supported
hierarchical_event_identifier	supported
hierarchical_function_identifier	supported
hierarchical_identifier	supported
hierarchical_net_identifier	supported
hierarchical_variable_identifier	supported
hierarchical_task_identifier	supported
identifier	supported
inout_port_identifier	supported
input_port_identifier	supported
instance_identifier	supported
library_identifier	unsupported
memory_identifier	supported
module_identifier	supported
module_instance_identifier	supported
net_identifier	supported
output_port_identifier	supported

Verilog Construct	ISim Support
parameter_identifier	supported
port_identifier	supported
real_identifier	supported
simple_arrayed_identifier	supported
simple_hierarchical_identifier	supported
simple_identifier	supported
specparam_identifier	supported
system_function_identifier	supported
system_task_identifier	supported
task_identifier	supported
terminal_identifier	supported
text_macro_identifier	supported
topmodule_identifier	supported
udp_identifier	supported
udp_instance_identifier	supported
variable_identifier	supported

Identifier Branches

Verilog Construct	ISim Support
simple_hierarchical_branch	supported
escaped_hierarchical_branch	supported

White Space

Verilog Construct	ISim Support
white_space	supported

Primitive and Module Instance Constructs

The following Verilog primitive instance and module instance constructs are supported by the ISim as defined below.

Primitive Instantiations

Verilog Construct	ISim Support
gate_instantiation	supported
cmos_switch_instance	unsupported
enable_gate_instance	supported
mos_switch_instance	unsupported
n_input_gate_instance	supported
n_output_gate_instance	supported

Verilog Construct	ISim Support
pass_switch_instance	unsupported
pass_enable_switch_instance	unsupported
pull_gate_instance	supported
name_of_gate_instance	supported

Primitive Strengths

Verilog Construct	ISim Support
pulldown_strength	supported
pullup_strength	supported

Primitive Terminals

Verilog Construct	ISim Support
enable_terminal	supported
inout_terminal	supported
input_terminal	supported
ncontrol_terminal	supported
output_terminal	supported
pcontrol_terminal	supported

Primitive Gate and Switch Types

Verilog Construct	ISim Support
cmos_switchtype	unsupported
enable_gatetype	supported
mos_switchtype	unsupported
n_input_gatetype	supported
n_output_gatetype	supported
pass_en_switchtype	unsupported
pass_switchtype	unsupported

Module Instantiations

Verilog Construct	ISim Support	Comment
module_instantiation	supported	
parameter_value_assignment	supported	
list_of_parameter_assignments	supported	
ordered_parameter_assignment	supported	
named_parameter_assignment	supported	
module_instance	supported	
name_of_instance	partially supported	Module instance arrays are not supported

Verilog Construct	ISim Support	Comment
list_of_port_connections	supported	
ordered_port_connection	supported	
named_port_connection	supported	

Generated Instantiation

Verilog Construct	ISim Support	Comment
generated_instantiation	supported	
generate_item_or_null	partially supported	The module_or_generate_item alternative is not supported. Production from 1364-2001 Verilog standard: `generate_item_or_null ::= generate_conditional_statement	generate_case_statement	generate_loop_statement	generate_block	module_or_generate_item` Production supported by ISim: `generate_item_or_null ::= generate_conditional_statement	generate_case_statement	generate_loop_statement	generate_block`
generate_item	supported	
generate_conditional_statement	supported	
generate_case_statement	supported	
generate_case_item	supported	
generate_loop_statement	supported	

genvar_assignment	partially supported	All generate blocks must be named.
		Production from 1364-2001 Verilog standard:
		`generate_block ::= begin [ : generate_block_identifier ] { generate_item } end`
		Production supported by ISim:
		`generate_block ::= begin : generate_block_identifier { generate_item } end`

Source Text Constructs

The following Verilog source text constructs are supported by the ISim as defined below.

Library Source Text

Verilog Construct	ISim Support	Comment
library_text	unsupported	
library_descriptions	unsupported	
library_declaration	unsupported	
file_path_spec	unsupported	
include_statement	unsupported	This refers to include statements within library map files (See IEEE 1364-2001, section 13.2). This does *not* refer to the 'include compiler directive.

Configuration Source Text

Verilog Construct	ISim Support
config_declaration	unsupported
design_statement	unsupported
config_rule_statement	unsupported
default_clause	unsupported
inst_clause	unsupported
inst_name	unsupported
cell_clause	unsupported
liblist_clause	unsupported
use_clause	unsupported

Module and Primitive Source Text

ISim Support	ISim Support
source_text	supported
description	supported
module_declaration	supported
module_keyword	supported

Module Parameters and Ports

Verilog Construct	ISim Support
module_parameter_port_list	supported
list_of_ports	supported
list_of_port_declarations	supported
port	supported
port_expression	supported
port_reference	supported
port_declaration	supported

Module Items

Verilog Construct	ISim Support
module_item	supported
module_or_generate_item	supported
module_or_generate_item_declaration	supported
non_port_module_item	supported
parameter_override	supported

Specify Function Constructs

The following Verilog Specify function constructs are supported by the ISim as defined below.

Specify Block Declarations

Verilog Construct	ISim Support
specify_block	supported
specify_item	supported
pulsestyle_declaration	supported
showcancelled_declaration	supported

Specify Path Declarations

Verilog Construct	ISim Support
path_declaration	supported
simple_path_declaration	supported
parallel_path_declaration	supported
full_path_description	supported
list_of_path_inputs	supported
list_of_path_outputs	supported

Specify Block Terminals

Verilog Construct	ISim Support
specify_input_terminal_descriptor	supported
specify_output_terminal_descriptor	supported
input_identifier	supported
output_identifier	supported

Specify Path Delays

Verilog Construct	ISim Support
path_delay_value	supported
list_of_path_delay_expressions	supported
t_path_delay_expression	supported
trise_path_delay_expression	supported
tfall_path_delay_expression	supported
tz_path_delay_expression	supported
t01_path_delay_expression	supported
t10_path_delay_expression	supported
t0z_path_delay_expression	supported
tz1_path_delay_expression	supported
t1z_path_delay_expression	supported
tz0_path_delay_expression	supported
t0x_path_delay_expression	supported
tx1_path_delay_expression	supported
t1x_path_delay_expression	supported
tx0_path_delay_expression	supported
txz_path_delay_expression	supported
tzx_path_delay_expression	supported
path_delay_expression	supported
edge_sensitive_path_declaration	supported
parallel_edge_sensitive_path_declaration	supported
full_edge_sensitive_path_declaration	supported
data_source_expression	supported
edge_identifier	supported
state_dependent_path_declaration	supported
polarity_operator	supported

System Timing Check Commands

Verilog Construct	ISim Support
system_timing_check	supported
$hold_timing_check	supported
$setuphold_timing_check	supported
$recovery_timing_check	supported
$removal_timing_check	supported
$recrem_timing_check	supported
$skew_timing_check	unsupported
$timeskew_timing_check	unsupported
$fullskew_timing_check	unsupported
$period_timing_check	supported
$width_timing_check	supported
$nochange_timing_check	unsupported

System Timing Check Command Arguments

Verilog Construct	ISim Support
checktime_condition	unsupported
controlled_reference_event	supported
data_event	supported
delayed_data	unsupported
delayed_reference	unsupported
end_edge_offset	unsupported
event_based_flag	unsupported
notify_reg	supported
reference_event	supported
remain_active_flag	unsupported
stamptime_condition	unsupported
start_edge_offset	unsupported
threshold	supported
timing_check_limit	supported

System Timing Check Event Definitions

Verilog Construct	ISim Support
timing_check_event	supported
controlled_timing_check_event	supported
timing_check_event_control	supported
specify_terminal_descriptor	supported
edge_control_specifier	supported
edge_descriptor	supported
zero_or_one	supported
z_or_x	supported
timing_check_condition	supported
scalar_timing_check_condition	supported
scalar_constant	supported

System Task and Function Constructs

The following Verilog System Tasks and Functions constructs are supported by the ISim as defined below.

Display System Tasks

Verilog Construct	ISim Support
$display	supported
$displayb	supported
$displayh	supported
$displayo	supported
$monitor	supported
$monitorb	supported
$monitorh	supported
$monitoro	supported
$monitoroff	supported
$monitoron	supported
$strobe	supported
$strobeb	supported
$strobeh	supported
$strobeo	supported
$write	supported
$writeb	supported
$writeh	supported
$writeo	supported

File I/O Tasks

Verilog Construct	ISim Support

Verilog Construct	ISim Support
$fclose	supported
$fdisplay	supported
$fdisplayb	supported
$fdisplayh	supported
$fdisplayo	supported
$ferror	supported
$fflush	supported
$fgetc	supported
$fgets	supported
$fmonitor	supported
$fmonitorb	supported
$fmonitorh	supported
$fmonitoro	supported
$fopen	supported
$fread	supported
$fscanf	supported
$fseek	supported
$fstrobe	supported
$fstrobeb	supported
$fstrobeh	supported
$fstrobeo	supported
$ftell	supported
$fwrite	supported
$fwriteb	supported
$fwriteh	supported
$fwriteo	supported
$readmemb	supported
$readmemh	supported
$rewind	supported
$sdf_annotate	supported
$sformat	supported
$sscanf	supported
$swrite	supported
$swriteb	supported
$swriteh	supported
$swriteo	supported
$ungetc	supported

Timescale Tasks

Verilog Construct	ISim Support
$printtimescale	supported
$timeformat	supported

Simulation Control Tasks

Verilog Construct	ISim Support
$finish	supported
$stop	supported

PLA Modeling Tasks

Verilog Construct	ISim Support
$async$and$array	unsupported
$async$nand$array	unsupported
$async$nor$array	unsupported
$async$or$array	unsupported
$sync$and$array	unsupported
$sync$nand$array	unsupported
$sync$nor$array	unsupported
$sync$or$array	unsupported
$async$and$plane	unsupported
$async$nand$plane	unsupported
$async$nor$plane	unsupported
$async$or$plane	unsupported
$sync$and$plane	unsupported
$sync$nand$plane	unsupported
$sync$nor$plane	unsupported
$sync$or$plane	unsupported

Stochastic Analysis Tasks

Verilog Construct	ISim Support
$q_add	supported
$q_exam	supported
$q_full	supported
$q_initialize	supported
$q_remove	supported

Simulation Time Functions

Verilog Construct	ISim Support
$realtime	supported
$stime	supported
$time	supported

Conversion Functions

Verilog Construct	ISim Support
$bitstoreal	supported
$realtobits	supported
$itor	supported
$rtoi	supported
$signed	supported
$unsigned	supported

Probabilistic Distribution Functions

Verilog Construct	ISim Support
$dist_chi_square	supported
$dist_erlang	supported
$dist_exponential	supported
$dist_normal	supported
$dist_poisson	supported
$dist_t	supported
$dist_uniform	supported
$random	supported

Command Line Input

Verilog Construct	ISim Support
$test$plusargs	supported
$value$plusargs	supported

Value Change Dump (VCD) Files

Verilog Construct	ISim Support
$dumpall	supported
$dumpfile	supported
$dumpflush	supported
$dumplimit	supported
$dumpoff	supported
$dumpon	supported
$dumpports	unsupported
$dumpportsall	unsupported
$dumpportsflush	unsupported
$dumpportslimit	unsupported
$dumpportsoff	unsupported
$dumpportson	unsupported
$dumpvars	supported

UDP Declaration and Instantiation Constructs

The following Verilog UDP instantiation constructs are supported by the ISim as defined below.

UDP Declaration

Verilog Construct	ISim Support
udp_declaration	supported

UDP Ports

Verilog Construct	ISim Support
udp_port_list	supported
udp_declaration_port_list	supported
udp_port_declaration	supported
udp_output_declaration	supported
udp_input_declaration	supported
udp_reg_declaration	supported

UDP Body

Verilog Construct	ISim Support
udp_body	supported
combinational_body	supported
combinational_entry	supported
sequential_body	supported
udp_initial_statement	supported
init_val	supported
sequential_entry	supported
seq_input_list	supported
level_input_list	supported
edge_input_list	supported
edge_indicator	supported
current_state	supported
next_state	supported
output_symbol	supported
level_symbol	supported
edge_symbol	supported

UDP Instantiation

Verilog Construct	ISim Support	Comment
udp_instantiation	supported	
udp_instance	supported	
name_of_udp_instance	partially supported	UDP instance arrays are not supported