

ISim User Guide

UG660 (v14.3) October 16, 2012

This document applies to the following software versions: ISE Design Suite 14.3 through 14.7





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 2012 Xilinx, Inc. XILINX, the Xilinx logo, Virtex, Spartan, ISE, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners. PCI, PCIe and PCI Express are trademarks of PCI-SIG and used under license

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
01/19/2011	13.4	<ul style="list-style-type: none">• Date and Revision change only.
04/24/2012	14.1	<ul style="list-style-type: none">• Updated Feature Support, page 3.• Consolidated the Simulation chapters into Chapter 3, Compilation and Simulation.<ul style="list-style-type: none">• Consolidated fuse, vhpcomp, and vlogcomp Command Options, page 51 into a single table and created a top-level command cross-reference• Added `uselib Verilog Directive in Chapter 3.• Added <code>-hil_zynq_ps</code> Hardware Co-Simulation command to fuse command option in both Chapter 3, Compilation and Simulation, and Chapter 8, Using Hardware Co-Simulation.• Consolidated VHDL Language Support Exceptions and Verilog Language Support Exceptions in Appendix B into two exceptions only tables.
07/25/2012	14.2	<ul style="list-style-type: none">• Revised one sentence in Chapter 3 to clarify ‘uselib directive.• Minor grammatical edits.• Updated links in Appendix D, Additional Resources.
10/16/2012	14.3	<ul style="list-style-type: none">• Added a note to <code>function_call</code> that states "In named parameter association in a <code>function_call</code> slicing, indexing or selection of formals is not supported."

Table of Contents

Revision History	2
Chapter 1: Introduction to ISim	
Simulation Libraries	3
Language Support	3
Feature Support	3
Operating System Support	4
ISim Modes of Operation	4
Simulation Steps Overview	4
ISim Tutorials	8
Chapter 2: Using the ISim GUI	
ISim GUI Overview	9
Setting ISim Preferences	32
Chapter 3: Compilation and Simulation	
Parsing Design Files	36
Project File Syntax	43
Predefined XILINX_ISIM Macro for Verilog Simulation	44
Simulating the Design	44
Mixed Language Simulation	45
Timing Simulation (Gate-Level Simulation)	49
ISim Executable Command	49
Pausing a Simulation	56
Saving Simulation Results	56
Closing Simulation	57
Chapter 4: Waveform Analysis	
Working with the Wave Configuration	59
Customizing the Wave Configuration	61
Navigating the Wave Configuration	67
Printing Wave Configurations	71
Using Custom Colors	72
Chapter 5: Viewing Simulation Results	
Waveform Databases and Configuration Files	73
Opening a Static Simulation	74

Chapter 6: Debugging at the Source Level

Stepping Through a Simulation	77
Using Breakpoints	78

Chapter 7: Writing Activity Data for Power Consumption

Chapter 8: Using Hardware Co-Simulation

Prerequisites	83
Use Models	83
Limitations	84
Usage for Compilation	84
fuse Command Line Flow	85
Tools Flow	85
Hybrid Co-Simulation Flow	88
Hardware Board Usage	90
Hardware Co-Simulation	90
ISim Hardware Co-Simulation Tcl Commands	91
Board Support	92
Frequently Asked Questions	95

Chapter 9: ISim Tcl Commands

Aliasing Simulation Commands	102
ISim Wave Viewer Tcl Commands Overview	103
Command Line Conventions	103
Tcl Commands	104

Appendix A: Library Mapping File (xilinxsim.ini)

Appendix B: Exceptions to VHDL and Verilog Language Support

VHDL Language Support Exceptions	141
Verilog Language Support Exceptions	143

Appendix C: Migrating from ModelSim XE to ISim

About ModelSim XE	147
About ISim	148
Feature Comparison	148
Simulation Process	149

Appendix D: Additional Resources

Xilinx Resources	157
ISim Tutorials	157

Chapter 1

Introduction to ISim

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

This document describes the ISim tool features, lists the HDL languages that ISim supports, and explains the methods of interfacing with the tool. For easier navigation through this document, in your PDF reader, turn on **Previous View** and **Next View** buttons to navigate back and forth to linked information.

Simulation Libraries

ISim uses precompiled simulation device libraries and updates those libraries automatically when updates are installed.

Note: Do *not* run the Simulation Library Compilation Wizard (Compplib) to compile libraries for use with ISim.

Language Support

ISim supports the following languages:

- VHDL IEEE-STD-1076-1993
- Verilog IEEE-STD-1364-2001
- Standard Delay Format (SDF) version 2.1
- VITAL-2000

Feature Support

The following features are supported:

- Incremental compilation
- Source code debugging
- SDF Annotation
- VCD Generation
- Power analysis and optimization using SAIF
- Native support for HardIP blocks (such as MGT, PPC, and PCIe®)
- Multi-threaded compilation
- Hardware Co-Simulation (HWCoSim)
- Mixed VHDL/Verilog
- Memory Editor for viewing and debugging memory elements
- Single-click simulation re-compile and re-launch
- Easy to use, one-click compilation and simulation
- Built-in Xilinx simulation libraries

Operating System Support

See the *Xilinx Design Tools: Installation and Licensing Guide (UG798)* for operating systems support.

The *Xilinx Design Tools: Release Notes Guide (UG631)* provides information about the most recent release changes. Links to these document are also available in [Appendix D, Additional Resources](#).

ISim Modes of Operation

ISim has two modes of operation:

- Graphical User Interface (GUI)
Provides a graphical view of simulation data. Menu commands, context commands, and toolbar buttons run simulation, and examine and debug data. For information about working with the GUI, see [Chapter 2, Using the ISim GUI](#).
- Command line mode
Has no interaction with the GUI and you run commands at the command prompt. After the simulation executable runs, a Tool Command Language (Tcl) prompt opens in which you can enter simulation Tcl commands to examine and debug data.

You can specify `-tclbatch <file_name>` option to the simulation executable to run a set of Tcl commands after simulation has been loaded. You must have quit as the last Tcl command if you want the simulation to quit upon completion. For more information, see [Chapter 3, Compilation and Simulation](#).

Simulation Steps Overview

The steps for simulating a design in ISim are:

- [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 required files to run a simulation in ISim are as follows:

- Design files, including stimulus file
- User libraries
- Miscellaneous data files

Stimulus File

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

- **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 tool. 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 upon how you launch ISim, there are different methods available to add user libraries:

- When launching Project Navigator, define the user libraries in the ISE tool. See “Working with VHDL Libraries” in [ISE Help](#) for details.
- When using ISim standalone, interactive command mode, or non-interactive mode, set the library mapping file (see [Appendix A, Library Mapping File \(xilinxim.ini\)](#)) to point to your logical or physical libraries.
- When launching ISim from the PlanAhead tool, define the user libraries in that tool. See the *PlanAhead User Guide (UG632)* for more information. [Appendix D, Additional Resources](#), contains a link to the document.

Step 2: Parsing and Elaborating the Design

Before 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.

GUI Mode

When you invoke ISim from either the ISE or the PlanAhead tool, the ISim GUI is launched, the design is parsed, and design components are elaborated. For details, see “Simulation from ISE” in [Step 3: Simulating the Design](#), or the *PlanAhead User Guide (UG632)*. The design is parsed and elaborated manually at the command line, as described in the next section. Then you can invoke the generated simulation executable with the `-gui` mode to launch the GUI.

Interactive Command Line Mode

The steps in the interactive command-line mode:

1. Creating a project file. See [Project File Syntax, page 43](#)
2. Using the `fuse` command. See [Running fuse, page 38](#)

Step 3: Simulating the Design

After design compilation and elaboration, the next step is to run the simulation executable, and simulate the design. For information about running simulation in read-only mode, see [Opening a Static Simulation in Chapter 5](#).

GUI Mode Simulation at the Command Line

After you generate a simulation executable (`x.exe` (default) or a user-specified name), you can run the simulation executable with the `-gui` switch on the command line; for example, `my_sim.exe -gui`. This command launches the GUI. The simulation executable command does not start the simulation. To start the simulation, use one of the run simulation commands described in [Simulating the Design, page 44](#).

You can then add signals to the Wave configuration. See [Working with the Wave Configuration, page 59](#) 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 to add all signals at top-level of your `my_sim.tcl` file to automatically trace the signals and display the signals in the GUI upon launch.

Simulation from ISE

Parsing, elaboration, and running the simulation executable command is run in the background when you run one of the following processes in the ISE or the PlanAhead tool.

- **Simulate Behavioral Model**
- **Simulate Post-Place & Route Model**

These processes launch the GUI with the top-level signals being traced by default.

Optionally, you can specify custom Tcl files to control the signals that are traced when you launch the GUI.

The simulator runs 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 [Simulating the Design, page 44](#).

Interactive Command Line Mode

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

Ensure 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

After the design is simulated, you debug the design to ensure that it meets the design specification.

You can examine the simulation results by:

- Viewing the signal interactions in the Wave window.
- Viewing or querying the results in the Console panel or the Tcl prompt.

In the debug phase, you can do the following:

- Save the results; see [Saving Simulation Results, page 56](#).
- View and examine simulation results in a read-only static simulator; see [Opening a Static Simulation in Chapter 5](#).

Step 5: Debugging the Design

If you encounter issues, you must debug the design to identify the root cause and the resolution of the issues. ISim provides a variety of ways to debug the design. To debug your design, examine the error messages and log files.

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 tool 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. Type `fuse -v 1` to dump information that might help identify the problem. A `fuse.log` file that contains a list of error messages and errors appears in the ISE tool Console (in ISE Integration Mode).
- Simulator
Indicates an error during executable code generation or simulation. See [Step 3: Simulating the Design, page 6](#). Use the file name and line number in the message to locate the issue.

Examining Log Files

Examining the available log files can provide helpful clues about design errors. The following log files are available:

- `fuse.log`
Log file containing output produced by the `fuse` command during the parsing 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 the `./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. The following commands are linked to the full command description, and can be run at the command line Tcl prompt, or in the Console panel.

- [isim ptrace on](#)
- [isim ltrace on](#)
- [dump](#)
- [show](#)
- [isim force](#)
- [bp](#)
- [onerror](#)

For debug strategies, see [Chapter 6, Debugging at the Source Level](#).

For more commands, see [Chapter 9, ISim Tcl Commands](#).

ISim Tutorials

See the following tutorials for more information:

- [ISE Simulator \(ISim\) In-Depth Tutorial \(UG682\)](#)
Demonstrates how to use ISim for design simulation and debugging.
- [ISE Hardware Co-Simulation Tutorial: Accelerating Floating Point FFT Simulation \(UG817\)](#)
Shows how to use the ISim Hardware Co-Simulation (HWCoSim) feature to accelerate Floating Point FFT simulation.

[Appendix D, Additional Resources](#), provides links to these documents.

Chapter 2

Using the ISim GUI

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 parts of the design that can be simulated
- Add and view signals in the wave configuration
- Use commands to run simulation
- Examine the design, and debug as necessary

ISim GUI Overview

The ISim GUI launches when you run the simulation executable from the ISE® tool, the command line, or the PlanAhead™ tool.

Figure 2-1 shows the ISim GUI.

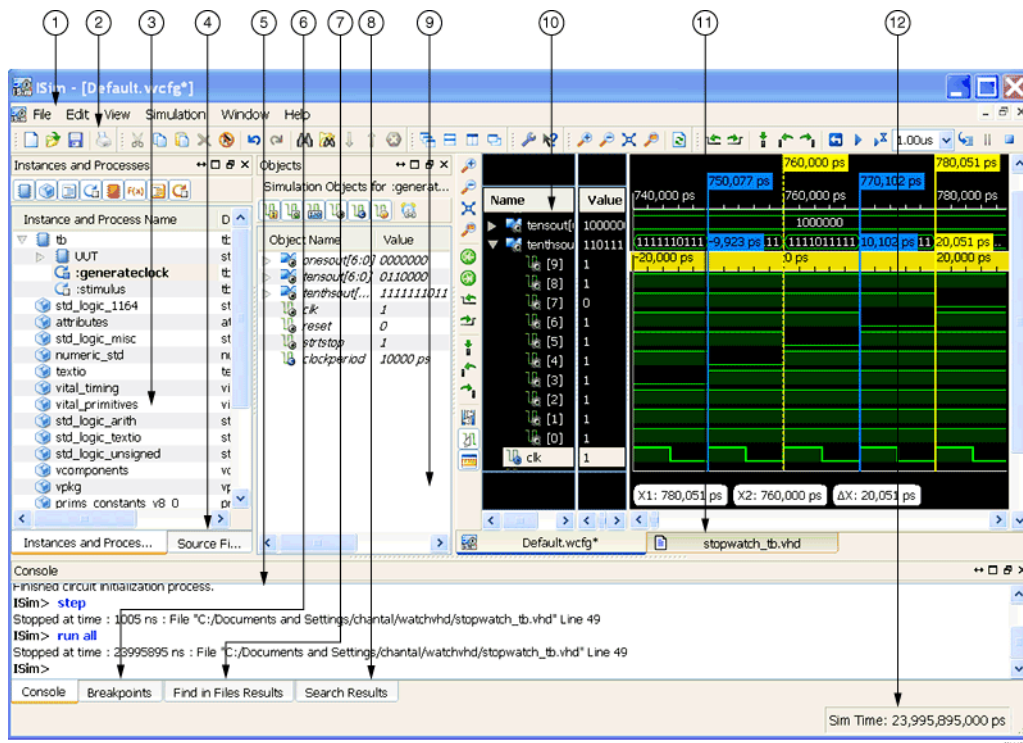


Figure 2-1: ISim GUI

To close ISim, select **File > Exit**. ISim prompts you to save your waveform configuration before closing.

Table 2-1 lists the ISim GUI components as identified in Figure 2-1, page 9, and links to the GUI part description.

Note: In your PDF reader, turn on **Previous View** and **Next View** Buttons to navigate back and forth to linked information.

Table 2-1: ISim GUI Components

GUI Part by #	Description
1. Menus and Toolbar: Commands and Shortcuts	Provides access to most operations available in the tool. Some operations are available in context menu only.
2. ISim Menu and Toolbar	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.
8. Search Results Panel	Displays the results that match the criteria from a search
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 Hardware Description Language (HDL) files.
12. Status Bar	Displays a brief description for a menu command or toolbar button that your cursor is placed over, and the simulation time.

The following subsections describe each ISim GUI component.

Menus and Toolbar: Commands and Shortcuts

The ISim main window consists of functionally different toolbars that reflect the most commonly used Main menu options.

The Main menu provides extended options within the option categories. The main window toolbar buttons are below the Main menu at the top of the user interface.

To show or hide toolbars, select **View > Toolbars > <toolbar_name>**.

File Menu and Standard Toolbar

The Standard toolbar provides access to frequently used File menu commands.



The File menu and the Standard toolbar provide access to the following options:

- **New**
Use the New dialog box and to select the type of file you want to create. You can open a new text file, schematic, or symbol.
- **Open**
Use this option to browse through your directories and select a file to open. The file displays in the appropriate application or editor.
- **Save**
Use this option to save 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.
- **Save All**
Use this option to save all files that require saving.
- **Print**
Use the Print dialog box to print an active file.

Edit Menu and Toolbar

The Edit toolbar provides access to frequently used Edit menu commands.



- **Cut, Copy, Paste, Delete** are available as well as **Undo, Redo, Find, and Find in File**.

View Menu and Toolbar

The View toolbar provides access to frequently used View menu commands.



View toolbar options are as follows:

- **Zoom In** and **Zoom Out, Set View for all content to be visible, and Zoom to Cursors.**
- The **Refresh** button cleans up the display of the file in focus.



In the View menu, additional options are:

- **Panel**
Opens a dialog box with the following check box options: Search Results, Find in Files Results, Breakpoints, Compilation Log, Source Files, Memory, Objects, Instances and Processes, Console.
- **Toolbars**
Turn the toolbars on and off.
- A check box controls the use of the Status Bar.







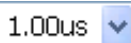



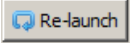
ISim Menu and Toolbar

The ISim toolbar provides access to frequently used ISim commands.



Table 2-2 describes the Simulation toolbar options:

Table 2-2: Simulation Toolbar Options

Button	Description
	Moves the main cursor to the nearest marker to the left of the current position of the marker.
	Moves the main cursor to the nearest marker to the right of the main current position of the marker.
	Adds a marker at the position of the main cursor to the Waveform area.
	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.
	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.
	Relaunch Simulation.

Window Menu and Toolbar

The Window toolbar provides access to frequently used Window menu commands.



The Window toolbar options are the standard options to cascade, tile, show side-by-side, and bring to front.

Help Toolbar

The Help toolbar provides access to frequently used Help menu commands. Support and Services displays the Xilinx® Support page in the default web browser.

What's This? activates tooltips. After clicking this button, you can hover over a menu item or button and get a brief description of its functionality.

Keyboard Shortcuts

Table 2-3 lists the ISim keyboard shortcuts.

Table 2-3: Keyboard Shortcuts

Shortcut	Menu Command
F1	Help Topics (Help menu)
F3	Find Next (Edit menu)
F5	Run All (View menu)
F6	Zoom Full View (View menu)
F7	Zoom Out (View menu)
F8	Zoom In (View menu)
F11	Step
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+A	Select All (Edit menu)
Ctrl+W	Add To Wave Configuration
Ctrl+F4	Close (Window menu)
Ctrl+Tab	Next (Window menu)
Ctrl+Shift+Tab	Previous (Window menu)
Ctrl+Home	Go To Time 0
Ctrl+End	Go To Latest Time
Ctrl+Shift+F5	Restart
Ctrl+ Mouse Wheel	Zooms in and out
Shift+ Mouse Wheel	Zooms left and right

Table 2-3: Keyboard Shortcuts (Cont'd)

Shortcut	Menu Command
Mouse Wheel	Scrolls up and down
Left	Previous Transition
Right	Next Transition
Pause	Break

Instances and Processes Panel

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

The columns in this panel are:

- **Instance and Process Name**
Shows the instance, process, and static tasks or functions buttons in a tree structure showing the block hierarchy of the design.
- **Design Unit**
Displays the names of the design units (Verilog module or VHDL entity architecture) corresponding to the instance, static task or function, or process from the first column.
- **Block Type**
Displays the type of the instance, static task or function, or process (for example, Verilog Module).

The Instances and Processes tabs are:

- **Instance**
Displays the instance, process, and static tasks or functions buttons in a tree structure showing the block hierarchy of the design.
- **Memory**
Displays the memory of the design object. See [Using the Memory Editor, page 26](#).
- **Source Files**
Lists the source files of the design.

Figure 2-2, page 14 shows the Instances and Processes panel.

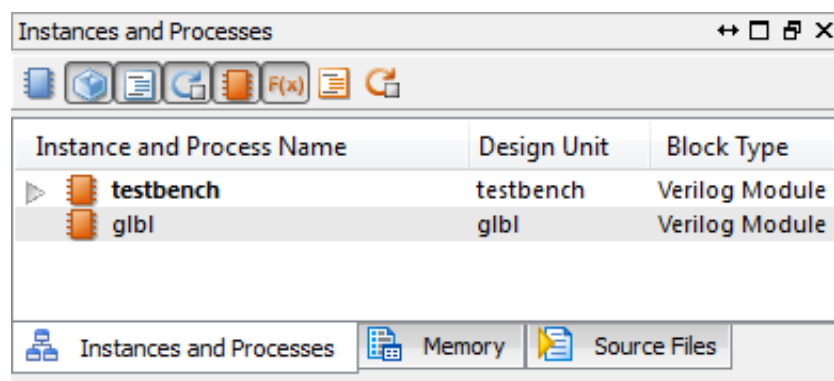










Figure 2-2: Instances and Processes Panel

Design Hierarchy Buttons

Table 2-4 describes the design hierarchy buttons in the Instances and Processes panel.

Table 2-4: Instance and Processes Panel Buttons



Icon	Represents
	VHDL Entity
	VHDL Package
	VHDL Block
	VHDL Process
	Verilog Module
	A toggle for filtering Verilog tasks.
	A toggle for filtering Verilog blocks.
	A toggle for filtering Verilog Processes.

With Hierarchy buttons you can take the following actions:

- To expand a hierarchy to display its components, click the arrows or use the **Expand** context menu commands (see [Expanding and Collapsing a Hierarchy](#), page 16).
- 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**.

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

Arranging the Main Window

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.
- Using Drag and Drop
For other parts of the interface, like panels and the main window toolbar, drag and drop lets you move the object to a new location. To do so:
 1. Click and hold the header for the panel to move.
 2. Move the panel to a new location.
A gray box indicates where the panel is 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**.

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

With these commands, you can hide, restore, float, and dock the panel. [Table 2-5](#) lists the buttons and description.

Table 2-5: Panel Control Buttons

Icon	Description
	Toggle Slide Out Minimizes the panel. Also used to restore the pane by hovering over the name of the panel at the edge of the window, and clicking the minimize button.
	Toggle Maximized Maximizes the panel. Click again to restore the panel size.
	Toggle Floating Floats the panel. Re-click to restore 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.

Wave Window

The Wave window displays signals, buses, and their waveforms. Each tab in the Wave window shows a wave configuration that contains a list of signals and buses, their properties, and any added wave objects such as dividers, cursors, and markers.

In the GUI, the signals and buses in the wave configuration are traced during simulation, and you use the wave configuration to examine the simulation results. The design hierarchy and the signal transitions are not part of the wave configuration, and 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, groups, markers and cursors. You can completely customize a wave configuration: you can add or remove signals and other wave objects at any time when the simulation is not actively running.

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

You can create and simulate multiple wave configurations, and the wave configurations can be saved separately.

For information about saving the Wave Configuration, see [Saving Wave Configurations, page 23](#).

When you invoke the simulator from any mode, it creates the `Default.wcfg` file. You must supply a filename to save a wave configuration file to the disk as a `.wcfg` file.

- In GUI mode, when ISim exits, it prompts you to type a filename in the **Save As** dialog box.
- In batch mode, type **wcfg save** to save the contents of the `Default.wcfg` before exiting the ISim tool.

Active Window

When you invoke the simulator, the first active window is `Default.wcfg`. You can change the active window by clicking the window tab or using the `wave add` command.

- In the GUI, select **File > New** or **File > Open** to change the active window to the newly created waveform configuration window.
- In Tcl, the `wcfg new` and `wcfg open` commands change the active window to the newly created window just like **File > New** and **File > Open**.

Wave Configuration Signal and Bus Buttons

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

Table 2-6 lists the ISim signal buttons. Table 2-7, page 19 lists the Bus signal buttons.

Table 2-6: ISim Signal Buttons

















Icon	Description
	Input Port
	Output Port
	InOut, Bidirectional Port
	Internal Signal
	Constants, parameters, and generics
	Variable
	Linkage Signal (VHDL only)
	Buffer Signal

Table 2-7 lists the ISim Bus buttons.

Table 2-7: ISim Bus Buttons

Button	Description
	Input Bus
	Output Bus
	InOut, Bidirectional Bus
	Internal Bus
	Constants, Parameters, 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.

- 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. See [Cursors in Chapter 4](#).
- 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 lets you display the signal value where the marker intersects the waveform. The time of the marker displays 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 [Markers in Chapter 4](#).

Adding Markers and Displaying Waveform Values With Markers

Hollow/Filled-in Circle


When placing or moving cursors and markers, you can use the **Snap to Transition** button to assist 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 over the signal transition, the mouse displays a filled in circle when hovering over a transition of a signal. 

Dividers


A divider is a visual separator of signals in the wave configuration.

Groups

A group is a virtual collection to which you can add signals and buses in the wave configuration as a means of organizing a set of related signals. A group displays the group 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 a Group in Chapter 4](#).

Virtual Buses

A virtual bus is a grouping 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 in Chapter 4](#). 

Wave Window Toolbar Buttons

Table 2-8 shows and describes the wave window toolbar buttons.

Table 2-8: Wave Window Toolbar Buttons















Button	Description
	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 0 time.

Table 2-8: Wave Window Toolbar Buttons (Cont'd)

Button	Description
	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 current position.
	Moves the main cursor to the nearest marker to the right of the 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

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 opens. The new wave configuration is empty until you add signals (see [Adding Signals to the Wave Configuration](#).)

If more than one wave configuration is open, either:

- Use the wave configuration tab to locate a particular wave configuration.
- 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 GUI, or using Tcl commands in the Console panel.

Note: Changes to the wave configuration, including creating the wave configuration or adding signals, do not become permanent until you save the WCFG file. For more information, see [Wave Configurations and WCFG Files](#).

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 panel, use `wave add` command.

Using Tcl:

- Optionally, you can first identify the objects you want to add by exploring the design hierarchy in the Instances and Processes panel and the Objects panel, or by entering the `scope` command in the Console panel.
- In the Console panel, enter the `wave add` command to enter an individual object or a group of objects.

Wave Configurations and WCFG Files

Although both a wave configuration and a WCFG file refer to the customization of lists of waveforms, there is a conceptual difference between them:

- The wave configuration is an object that is loaded into memory with which you can work.
 - You can name a wave configuration or leave it untitled. The name appears on the tab of the wave configuration window.
 - When saving a wave configuration to a WCFG file using a GUI Tcl command, the WCFG file takes the name supplied as a command argument.
 - When loading a wave configuration from a WCFG file, the wave configuration displays the name of the file.
- The WCFG file is the saved form of a wave configuration on disk.

Saving Wave Configurations

You can save the current wave configuration, and if you have multiple wave configurations open, each can be saved to a unique name for later viewing.

To save a wave configuration, do one of the following:

- Select **File > Save**
- Press **Ctrl+S**
- Click the **Save** button

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



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 in the Objects panel. Search criteria includes a text string, and/or an object-type filter.

To search for objects, do the following:

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 for which you are searching. Click **Match case** if applicable.
5. Click **OK**.

Objects that match the search criteria display in the Search Results Panel.

Opening HDL Source Files

You can open Hardware Description Language (HDL) source file in the ISim Text Editor.

To view an HDL source file, do the following:

1. In the Instances and Processes panel, the Objects panel, or the Source Files panel, select a file.
2. Double-click the file, 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.

When you open a file using the **File > Open** menu command, the file is in write mode.

In the **Open** dialog box, change **Files of type** file to Verilog or VHDL, select the file, and click **Open**. See [Modifying Source Files, page 26](#).

Source Files Panel

The Source Files panel displays as a tab in the Instances and Processes panel. When you select the tab, it displays list of 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.

To open a source code file, do the following:

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

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



Objects Panel

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 or process is selected in the Instances and Processes panel; those 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.
- **Value**
The value of the simulation object at the current simulation time or at the main cursor, as determined by the **Sync Time** button.
- **Data Type**
Displays the data type of the corresponding simulation object, logic or an array.

Toggle buttons are available in the Objects panel, as described in [Table 2-9](#).

Table 2-9: Object Panel Toolbar Buttons








Button	Description
	Toggles the input ports on and off.
	Toggles the output ports on and off.
	Toggles the inout, bidirectional ports on and off.
	Toggles the internal signals on and off.
	Toggles the constants, parameters, and generics on and off.

Table 2-9: Object Panel Toolbar Buttons (Cont'd)

Button	Description
	Toggles the variables on and off.
	Toggles the Sync Time feature on and off. <ul style="list-style-type: none"> When on, Objects panel values are based on the main cursor in the Wave window. When off, values are the values at the Sim Time in the Status Bar (at simulation end time).

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 helps 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 (using the [show driver](#) command)

To show drivers:

1. Select an object, or signal.
2. Select **Edit > Wave Objects > Show Drivers**.

The Console panel lists the drivers for the object or signal. When there is no driver, a message indicates that there is no driver.

Note: Running this command is the same as running `show driver` at the Console panel 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 in the object hierarchy display.

To limit the display of child elements, right-click anywhere in the object list in the Objects panel, and 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**, and enter a number.
4. Click **Apply**, and **OK**.

Selecting an Object in the Wave Window

To highlight signals for an object in the Objects panel:

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

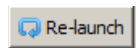
Text Editor Window

The Text Editor window is available for access to the underlying HDL source files.

Modifying Source Files

To modify source files:

1. Open the source file in the ISim Text Editor.
2. Make the appropriate edits, and run **Re-Launch** to re-simulate the design.



The ISE project automatically saves the source file changes.

Setting Breakpoints

You can set breakpoints in executable lines in your HDL file so you can run your code continuously until the source code line with the breakpoint is reached, as described in [Using Breakpoints in Chapter 6](#).

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

Using the Memory Editor

The Memory Editor lets you find and change contents of two-dimensional memory arrays in a design during simulation (without recompiling or re-elaborating the design). There are three places that show memory objects: the Memory tab, the Object panel, and the Search Result tab. To open the Memory Editor, follow one of these methods.

- On the Memory tab which contains all the two-dimensional arrays of logic types in a design, double-click a displayed memory object.
- In the Objects panel, right-click a two-dimensional array of logic type, and select **Memory Editor** from the context menu.
- In the Instance and Processes panel, run a search on a memory name. When the searched memory displays in the Search Results panel, you can right-click the memory and select **Memory Editor** from the context menu.

Note: For objects that are not two-dimensional array of a logic type, the Memory Editor choice in the Context Menu is grayed out.

The Memory Editor displays the following fields:

- **Address**
Go to a particular location in the displayed memory.
- **Columns**
Controls the display of the number of elements per row. The auto column displays the maximum of 2 to power N of elements.

- **Address Radix**
Controls the radix of the address displayed in the Memory Editor.
- **Value Radix**
Controls the radix of the display value in the Memory Editor.

You can float the Memory Editor window and the Memory Editor retains the previous state after the float operation.

You can navigate inside Memory Editor with the arrow keys, the current position of a selected item displays on the status bar based on the current address radix.

Console Panel

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

Messages

Generated messages include errors, warnings, and informational messages. The Console panel also echoes simulator commands that were invoked from the graphical controls in the ISim GUI.

Simulation commands

The command prompt lets you enter simulation Tcl commands, and to view the command dump (or print-out) in the Console panel. See [Simulating the Design in Chapter 3](#).

A number of right-click menu commands are available to help manage the contents of the Console panel.

Breakpoints Panel

A breakpoint is a user-determined stopping point in the source code used for debugging a design with ISim. The Breakpoints panel displays a list of breakpoints that are set in the design. See [Using Breakpoints in Chapter 6](#).

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

To set a breakpoint, use one of the following options:

- Select **View > Breakpoint > Toggle Breakpoint**.
- Click the **Toggle Breakpoint** button.
- In the HDL file, click a line of code just to the right of the line number.
- Type `bp <option>` in the Tcl console.



Alternatively, you can right-click a line of code, and select **Toggle Breakpoint**.

After the procedure completes, a simulation breakpoint icon appears next to the line of code.






A list of breakpoints is available in the Breakpoints panel. If you place a breakpoint on a line of code that is not executable, the breakpoint is not added.

To remove the breakpoint click the breakpoint.

Breakpoint Toolbar Buttons

Table 2-10 describes the Breakpoint buttons.

Table 2-10: **Breakpoint Buttons**

Button	Description
	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.
	Opens the HDL source file in the text editor with the breakpoint in focus.






Search Results Panel

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 Command Buttons

Table 2-11 shows and describes the buttons available in the Search Results panel.

Table 2-11: **Search Results Toolbar Buttons**








Button	Description
	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

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

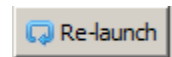
To use the **Find in Files** options:

Table 2-12: Find in File Buttons and Actions

Button	Action
	Select Edit > Find in Files , or click the Find Text in Files button. In the Find in Files dialog box, specify the text to find, set the parameters for your search, and click Find . In the Find in Files Results panel, do any of the following:
	To clear all results from the panel, click the Clear All button.
	To open the file that contains the find result in the Workspace, select a find result, and click the Show Current Result button. Alternatively, you can double-click the find result to open the file.
	To view the next find result, click the Show Next Result button.
	To view the previous find result, click the Show Previous Result button.
	To stop the currently running Find in Files search, click the Stop Job button.
	To save your Find in Files search results to a Comma Separated Value (CSV) file, click the Save Results as a Text File button.

Re-launching Simulation

The Re-launch button lets you re-launch the ISim simulation after making a modification in an Hardware Description Language (HDL) file to fix an identified issue. You also can recompile from the ISim GUI.



Recompile and Re-launch are fully automated features. The dialog box messages specify where an issue is located. Re-launch keeps all the options as set at compile time, and automatically runs simulation to the specified runtime when the flow was launched from either the Project Navigator or the PlanAhead tool.

- When you successfully re-launch a simulation, your simulation completes without errors.
- When you re-launch an unsuccessful simulation, a dialog box opens with the syntax error failure to the compiled source code. The links go to the source code with errors in the source window. It is recommended that you address the linked errors sequentially to correct the issues and then recompile using the **Re-launch** button to verify the fix.

Applying Stimulus

Use the **Force Selected Signal** dialog box to enter parameters to force a VHDL signal, Verilog wire, or a Verilog reg to a constant value. This dialog box opens when you select a signal then right-click the **Force Constant** option. After you assign a new constant force, those values override the assignments made from within HDL code or any previously applied constant or clock force. Click **Apply** to apply all changes. Figure 2-3 shows the Force Selected Signal dialog box.

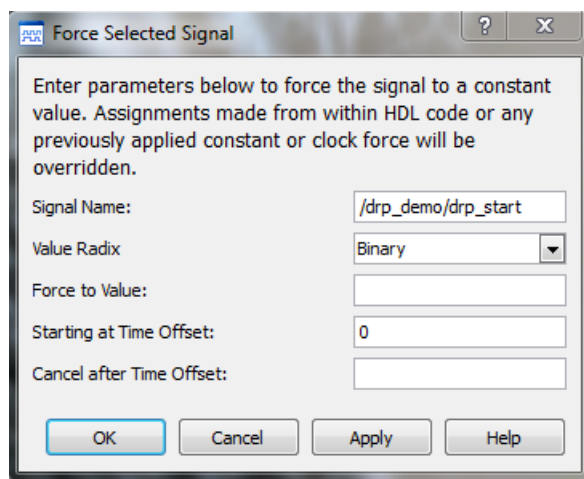


Figure 2-3: Force Selected Signal Dialog Box

The Force Selected Signal options are:

- **Signal Name**
Displays the default signal name. The default signal name is the full path name of the selected item. You can change the signal name in the edit box. When you enter an invalid signal name in the edit box, the edit box turns red.
- **Value Radix**
Displays the current radix setting of the selected signal. You can choose one of the supported radix types - Binary, Hexadecimal, Unsigned Decimal, Signed Decimal, Octal, and ASCII.
- **Force to Value**
Specifies a force constant value using the defined radix value.
- **Starting at Time Offset**
Starts after the specified time. The default starting time is 0. Time can be a string, such as "10" or "10 ns." When you enter a number without a unit, ISim uses the default.
- **Cancel after Time Offset**
Cancels after the specified time. Time can be a string such as 10 or 10 ns. When a number entered without a unit, the default simulation time unit is used.

Applying Clock Stimulus

When you right-click **Force Clock**, you can use the **Define Clock** dialog box to enter parameters to force a VHDL signal, Verilog wire, or a Verilog reg to an alternating pattern (clock). The newly applied clock pattern overrides assignments made from within HDL code or any previously applied constant or clock force. Click **Apply** to apply changes.

Define Clock Dialog Box

Right-click **Force Clock** to open the **Define Clock** dialog box. The options in the dialog box are:

- **Signal Name**
Displays the default signal name, which is the full path name of the item selected in the Objects panel or waveform. You can change the signal name in the edit box. When you enter an invalid signal name in the edit box, the edit box turns red.
Note: Running the `restart` command cancels all the effective `isim force` commands.
- **Value Radix**
Displays the current radix setting of the selected signal. You can choose one of the supported radix types from the dropdown box: Binary, Hexadecimal, Unsigned Decimal, Signed Decimal, Octal, and ASCII.
- **Leading Edge Value**
Specify the first edge of the clock pattern. The Leading Edge Value uses the radix defined in Value Radix.
- **Trailing Edge Value**
Specify the second edge of the clock pattern. The Trailing Edge Value uses the radix defined in the **Value Radix** field.
- **Starting at Time Offset**
Start the force command after the specified time from the current simulation. The default starting time is 0. Time can be a string, such as 10 or 10 ns. If you enter a number without a unit, ISim uses the default user unit as returned by the `isim get userunit` Tcl command.
- **Cancel after Time Offset**
Cancel the force command after the specified time from the current simulation time. Time can be a string, such as 10 or 10 ns. When you enter a number without a unit, ISim uses the default simulation time unit.
- **Duty Cycle (%)**
Specify the percentage of time that the clock pulse is in an active state. The acceptable value ranges from 0 to 100.
- **Period**
Specify the length of the clock pulse, defined as a time value. Time can be a string, such as 10 or 10 ns.

Define Clock Settings Examples

To assign a permanent clock to a signal (100 MHz clock), set the following fields:

- **Leading Edge Value:** 1
- **Trailing Edge Value:** 0
- **Starting at Time Offset:** 0
- **Cancel after Time Offset:** <blank>
- **Duty Cycle (%):** 50
- **Period:** 10 ns

To assign a clock to a signal for a specific period of time (start toggling at 100 ns, stop toggling after 1 ms), set the following fields:

- **Leading Edge Value:** 1
- **Trailing Edge Value:** 0
- **Starting at Time Offset:** 100 ns
- **Cancel after Time Offset:** 1 ms
- **Duty Cycle (%):** 50
- **Period:** <specify clock period>

To assign a toggling value for a signal (toggle between hex F and hex A every 50 ns for 1us), set the following fields:

- **Value Radix:** Hexadecimal
- **Leading Edge Value:** F
- **Trailing Edge Value:** A
- **Starting at Time Offset:** 0
- **Cancel After Time Offset:** 1us
- **Duty Cycle (%):** 50
- **Period:** 50 ns

Setting ISim Preferences

The preference settings let you 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 Preferences](#)
 - [ISim Simulator Preferences](#)
3. Make the necessary setting changes.
4. Click the **Apply** button, and then click **OK**.

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

ISE Text Editor Preferences

The preference setting associated with ISE Text Editor controls the behavior of Hardware Description Language (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 Simulator page in the **Preferences** dialog box. Select **Edit > Preferences > ISim Simulator** in the left pane.

Draw Waveform Shadow

Shows or 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 Default Radix](#).

Console text font

The window to the right of the field shows example text for the specified font. Click the **Change** button to open a dialog box in which you can specify the font used in the Console.

ISim Color Preferences

Use the Colors page to set your color preferences for displaying the waveform. Click **Apply** to apply changes. The color preference options are:

- **Current Color Scheme**
Displays the default color scheme and any custom schemes you have created.
- **New**
Creates a new scheme. Enter the new name in the **Current Color Scheme** field and edit the colors in the scheme table.
- **Delete**
Deletes the custom scheme that you have selected. You can edit the color of this scheme.

See [Setting ISim Preferences, page 32](#).

Time Format Preferences

You can customize the appearance of displayed time values using the **Time Format Preferences**. Select **Edit > Preferences > Time Format** in the left pane. The following subsections describe the two categories of time formats.

Waveform Window

Time format options apply to the GUI elements inside the waveform viewer window. The time format options are:

- **Link All Waveform Time Units To Ruler**
Is on by default and keeps the Units setting of the Cursors/Markers and Measure Bubble categories in sync with changes to the Ruler category.
- **Rulers**
Applies to the main ruler at the top of the waveform window as well as to the floating ruler.
- **Cursors/Markers**
Applies to the time values displayed for the all cursors and markers.
- **Measure Bubbles**
Applies to the cursor value bubbles displayed at the bottom of the waveform window.

Other GUI Elements

The time format options apply to the GUI elements outside the Waveform window as follows:

- **All Time Values**
Applies to the current simulation time shown at the bottom right of the main window and time values shown in the Objects panel. Time formats allow setting of time units and precision of display of values using following fields:
 - **Units**
Lets you set the units for time values. The default settings for the Other GUI elements is **Default**; for the Waveform Window, the default is **Auto**.
 - **Decimal Places**
Lets you set the number of decimal places to be used in displaying time values. The default setting for all categories that have a setting is Maximum.
 - **Reset To Defaults**
Resets the values to the original default settings.

Compilation and Simulation

You can run simulation for VHDL, Verilog, or Mixed Language components.

- Functional Simulation can be run early in the design process.
- Timing Simulation must be run post-Place and Route (PAR)

In addition to the two types of simulation, you have the options of running simulation as follows:

- From a Tools Command Language (Tcl) batch file or from the command line using Tcl commands.
- From the GUI, which can be invoked from either the ISE® Design Suite or the PlanAhead™ tools.

Running a simulation from the command line for either a Functional or a Timing simulation requires the following steps:

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

There are additional requirements for a Timing simulation, which is described in [Timing Simulation \(Gate-Level Simulation\)](#), page 49.

The following subsections describe these steps.

Parsing Design Files

The `vhpcomp` and `vlogcomp` commands parse VHDL and Verilog files, respectively.

vlogcomp Command

The `vlogcomp` command parses Verilog source files and generates a binary representation of the Verilog files. The binary representation generated by `vlogcomp` is used by the `fuse` command 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. See [Project File Syntax, page 43](#) for information about the project file.

vlogcomp Command Syntax

To go to the command description click the option link.

Note: In your PDF reader, turn on **Previous View** and **Next View** buttons to navigate back and forth.

```
vlogcomp
[-d <macro_definition>=<value>=<value>]
[-f <cmd_file>]
[-h]
[-i "<include_path>"]
[-intstyle [ise | xflow | silent | default]]
[-initfile <sim_init_file>]
[[-L]-lib <search_lib>[=<lib_path>]]
[-prj <prj_file>.prj]
[-sourcelibdir <directory_name>]
[-sourcelibext <file_extension>]
[-sourcelibfile <file_name>]
[-v [-verbose] <value>]
[-version]
[<verilog_files>...]
[-work [<work_library>[=<library_path>] ] <filenames>...]
```

The option descriptions are in the [fuse, vhpcomp, and vlogcomp Command Options](#) section.

vlogcomp Command Examples

Use two Verilog files, specify a source library directory, and specify a Verilog file extension:

```
vlogcomp tb.v fft.v -sourcelibdir ./mylib -sourcelibext .v
```

Specify a Verilog file and the source library file:

```
vlogcomp dff.v -sourcelibfile ./mylib/dff_lowest.v
```

Specify the work directory and a Verilog file:

```
vlogcomp -work my_lib tb.v
```

Use two files:

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

Search for unresolved cells inside directory `mydir/cells`:

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

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

Supporting Source Libraries

The following command arguments provide equivalent support to the de-facto Verilog-XL standard.

Pass the following command options to the `vlogcomp` command:

```
vlogcomp -sourcelibdir <library_location>
```

Note: `-sourcelibdir` provides the same functionality as the `-y` switch in Verilog-XL.

After the source files on the command line are parsed, if there are any unresolved references to modules, the command searches the source libraries for resolution.

During this search, `vlogcomp` 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, `vlogcomp` analyzes that file.

While processing the `-sourcelibdir` switch, `vlogcomp` ignores any files with extensions such as `.v` or `.h`, unless `-sourcelibext` is also used.

Note: `-sourcelibext` provides functionality that is similar to the `+libext+` switch in Verilog-XL. You can use this command line argument in conjunction with `-sourcelibdir` when the source library files have extensions. Examples:

```
vlogcomp -sourcelibdir /project/mysources tb.v fft.v
vlogcomp -sourcelibdir /project/mysources tb.v fft.v -sourcelibext .v
```

You can also provide a source Verilog library file that contains definitions of all the unresolved modules.

Example:

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

vhpcomp Command

The `vhpcomp` command parses VHDL source files and stores a binary representation of the HDL files. The binary representation generated by `vhpcomp` is used by the `fuse` command to create a simulation executable.

Specify either a project file or one or more VHDL files. If neither project file nor VHDL file are specified, `vhpcomp` issues an error. See [Project File Syntax, page 43](#) for information about the project file.

vhpcomp Syntax

To go to the command option, click the option link.

Note: In your PDF reader, turn on **Previous View** and **Next View** Buttons to navigate back and forth.

The `vhpcomp` command syntax is:

```
vhpcomp
[-f <cmd_file>]
[-h]
[-i "<include_path>"]
[-intstyle [ise | xflow | silent | default]]
[-initfile <sim_init_file>]
[[-L|-lib <search_lib>[=<lib_path>]]]
[-prj <prj_file>.prj]
[-rangecheck]
[-v [-verbose] <value>]
[-version]
[<vhdl_files>...]
[-work [<work_library>[=<library_path>] ] <filenames>...
```

The options descriptions are in the [fuse, vhpcomp, and vlogcomp Command Options](#) section.

vhpcomp Command Example

`vhpcomp` using two files:

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

Running fuse

The `fuse` command:

- Performs static elaboration of a design in terms of parsed nodes
- Generates object code for each unique module instance
- Links the generated object codes with the simulation engine library to create a simulation executable

The `fuse` command generates object code and data files for each design unit comprising the design, and puts them in the `isim/<simulation_executable>.sim` directory.

Note: Do not remove the `isim/<simulation_executable>.sim` directory; otherwise the design cannot be simulated.

fuse Command Syntax

Click the link to go to the command option descriptions.

Note: In your PDF reader, turn on **Previous View** and **Next View** Buttons to navigate back and forth. The fuse command syntax is:

```

fuse
[-d <macro_definition>=<value>=<value>]
[-f <cmd_file>]
[-generic_top "<parameter>=<value>"]
[-h]
[-hil_zynq_ps]
[-hwcosim_board <arg>]
[-hwcosim_clock <arg>]
[-hwcosim_instance <arg>]
[-hwcosim_no_combinatorial_path]
[-hwcosim_incremental <arg>]
[-i "<include_path>"]
[-incremental]
[-initfile <sim_init_file>]
[-intstyle [ise | xflow | silent | default]]
[[-L|-lib <search_lib>[=<lib_path>]]]
[-log <file_name>]
[-maxdeltaid <number>]
[-maxdelay]
[-maxdesigndepth <depth>]
[-mindelay]
[-mt <value>]
[-nodebug]
[-nolog]
[-nospecify]
[-notimingchecks]
[-o <sim_exe> ]
[-override_timeprecision ]
[-override_timeunit]
[-prj <prj_file>.prj]
[-rangecheck]
[-sdfnoerror]
[-sdfnowarn]
[-sdfmin][[-sdftyp][[-sdfmax] <root=file>]]
[-sdfroot <root_path>]
[-sourcelibdir <directory_name>]
[-sourcelibext <file_extension>]
[-sourcelibfile <file_name>]
[-timeprecision_vhdl <time_precision>]
[-timescale <time_unit/time_precision>]
[-typdelay]
[-v [-verbose] <value>]
[-version]

```

The option descriptions are in the [fuse, vhpcomp, and vlogcomp Command Options](#) section.

fuse Command Examples

Use the fuse command with a project file and source library:

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

Where test.prj contains “verilog work test.v”

The fuse command uses files from the -sourcelibfile options for modules used in test.v as it analyzes the modules and elaborates the test design.

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

Use the fuse command with the work.globl option:

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

Note: Using globl as <top_name> is necessary in fuse during timing simulation. The -o switch is optional. Without -o, the default name for the simulation executable is x.exe. For example:

```
fuse topunit work.globl -prj <mydesign>.prj -o <my_sim>.exe
```

Note: To exclude certain lines in a PRJ file use the -- option.

Verilog Search Order

The fuse command uses the following search order to search and bind instantiated Verilog design units:

1. A library specified by the `uselib directive in the Verilog code. For example:

```
module
full_adder(c_in, c_out, a, b, sum)
input c_in,a,b;
output c_out,sum;
wire carry1,carry2,sum1;
`uselib lib = adder_lib
half_adder adder1(.a(a),.b(b),.c(carry1),.s(sum1));
half_adder adder1(.a(sum1),.b(c_in),.c(carry2),.s(sum));
c_out = carry1 | carry2;
endmodule
```

2. Libraries specified on the command line with -lib|-L switch
3. A library of the parent design unit
4. The work library

Verilog Instantiation Unit

When a Verilog design instantiates a component, the `fuse` command 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 the unit 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 a case-insensitive search.

The `fuse` command searches for an entity with the same name as the instantiated module name in the user-specified list and order of unified logical libraries, searches for and selects the first one matching name, then stops the search.

- If the case sensitive search is not successful, `fuse` performs a case sensitive search for a VHDL design unit name constructed as an extended identifier in the user-specified list and order of unified logical libraries.
- If `fuse` finds a unique binding for any one library, it selects that name and stops the search.

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.

VHDL Instantiation Unit

When a VHDL design instantiates a component, the `fuse` command treats the component name as a VHDL unit and searches for it in the logical `work` library.

- If a VHDL unit is found, the `fuse` command binds it and the search stops.
- If `fuse` does not find a VHDL unit, it treats the case-preserved component name as a Verilog module name and continues a case-sensitive search in the user-specified list and order of unified logical libraries. The command selects the first matching the name, then stops the search.
- If case sensitive search is not successful, `fuse` performs a case-insensitive search for a Verilog module in the user-specified list and order of unified logical libraries. If a unique binding is found for any one library, the search stops.

``uselib` Verilog Directive

The Verilog ``uselib` directive is supported in ISim and by the library search order.

``uselib` Syntax

```
<uselib compiler directive> ::= `uselib [<Verilog-XL uselib
directives>|<lib directive>]
<Verilog-XL uselib directives> ::= dir = <library_directory> | file
= <library_file> | libext = <file_extension>
<lib directive> ::= <library reference> { <library reference>}
<library reference> ::= lib = <logical library name>
```

``uselib` Lib Semantics

The ``uselib lib` directive can not be used with any of the Verilog-XL ``uselib` directives. For example:

```
`uselib dir=./ file=f.v lib=newlib
```

Is illegal.

Multiple libraries can be specified in one ``uselib` directive.

The order in which libraries are specified determine the search order. For example:

```
`uselib lib=mylib lib=yourlib
```

Specifies that the search for an instantiated module is made in `mylib` first, followed by `yourlib`.

Like ``uselib dir`, ``uselib file`, and ``uselib libext` directives, the ``uselib lib` persists across HDL files in a given invocation of `parse/analyze`, such as in a given invocation of `parse` unless another ``uselib` is encountered; any previously encountered ``uselib` (including any Verilog XL ``uselib` directive) in the HDL source remains in effect.

A ``uselib` without any argument removes the effect of any currently active ``uselib lib|file|dir|libext`.

The following module search mechanism is used for resolving an instantiated module or UDP by the Verific Verilog elaboration algorithm:

- First, search for the instantiated module in the ordered list of logical libraries of the currently active ``uselib lib` (if any).
- If not found, search for the instantiated module in the ordered list of libraries provided as search libraries in `fuse` command line.
- If not found, search for the instantiated module in the library of the parent module. For example, if module A in library `work` instantiated module B of library `mylib` and B instantiated module C, then search for module C in library `mylib`, which is the library of C's parent B.
- If not found, search for the instantiated module in the `work` library, which is one of the following:
 - The library into which HDL source is being compiled
 - The library explicitly set as `work` library
 - The default work lib is named as `work`

`uselib Examples

File half_adder.v compiled into logical library named adder_lib	File full_adder.v compiled into logical library named work
<pre> module half_adder(a,b,c,s); input a,b; output c,s; s = a ^ b; c = a & b; endmodule </pre>	<pre> module full_adder(c_in, c_out, a, b, sum) input c_in,a,b; output c_out,sum; wire carry1,carry2,sum1; `uselib lib = adder_lib half_adder adder1(.a(a),.b(b),. c(carry1),.s(sum1)); half_adder adder1(.a(sum1),.b(c_in),.c (carry2),.s(sum)); c_out = carry1 carry2; endmodule </pre>

Project File Syntax

To parse design files using a project file, create a file called <proj_name>.prj, and use the following syntax inside the project file:

```

verilog <work_library> <file_names>... [-d <macro>]...
[-i <include_path>]...
vhdl <work_library> <file_name>

```

Where:

- <work_library> is the library into which the HDL files on the given line should be compiled.
- <file_names> are Verilog source files. You can specify multiple Verilog files per line.
- <file_name> is a VHDL source file; specify only one VHDL file per line.
- For Verilog, [-d <macro>] optionally lets you define one or more macros.
- For Verilog, [-i <include_path>] optionally lets you define one or more <include_path> directories.

Predefined XILINX_ISIM Macro for Verilog Simulation

XILINX_ISIM is a Verilog predefined-macro. The value of this macro is 1. Predefined macros perform tool-specific functions, or 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
```

Simulating the Design

Simulation is the process of verifying the logic and timing of a design, and can be run from ISim using functions in the GUI, in a Tcl batch file, or at the command line. After you parse the design files and create a simulation executable using `fuse`, you can run a functional or a timing simulation.

Running Simulation

You can run simulation using any of the following methods:

- [Using the Command Line](#)
- [Using the GUI](#)

Using the Command Line

You can run the executable generated by the `fuse` command to launch simulation in a command shell. If you do not specify the `-gui` switch to the simulation executable, the simulation launches in command line mode.

For example, type:

```
x.exe
```

The **ISim** > Tcl prompt opens, and you can interactively type simulation Tcl commands that let you run simulation and debug the design.

You can use the `-tclbatch` option to contain commands within a file and execute those command as simulation starts.

For example, you can have a file named `run.tcl` that contains the following:

```
run 20ns
show time
quit
```

Then launch simulation as follows:

```
x.exe -tclbatch run.tcl
```

Using the GUI

The following example shows command options for running the simulation executable and opening the GUI:

```
<executable_name>.exe -gui
```

You can use the following GUI menu commands 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. Alternatively, you can type the `restart` Tcl command in the Console panel.
- **Simulation > Run All**
Runs simulation until all events are executed. Alternatively, you can type the `run -all` Tcl command in the Console.
- **Simulation > Run**
Runs simulation for 100ns, or for the specified amount of time. Type the Time and time unit in the **Value** box. Alternatively, you can use the `run` Tcl command with a specified `-length` and `-unit`.
- **Simulation > Step**
Runs simulation for one executable HDL instruction at a time. See [Stepping Through a Simulation in Chapter 6](#), and the `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 -a11` command. See [Chapter 6, Debugging at the Source Level](#).

Note: The current simulation time displays on the status bar in the lower-right corner.

Mixed Language Simulation

ISim supports mixed language project files and mixed language simulation. This lets you include Verilog modules in a VHDL design, and vice versa.

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.

Key Steps in a Mixed Language Simulation

1. Optionally, specify the search order for VHDL entity or Verilog modules in the design libraries of a mixed language project.
2. Use `fuse -L` to specify the binding order of a VHDL entity or a Verilog module in the design libraries of a mixed language project.

Note: The library search order specified by `-L` is used for binding Verilog modules to other Verilog modules as well.

Mixed Language Binding and Searching

When you instantiate a VHDL component or a Verilog module, the `fuse` command:

- 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 by the `-lib` option.

The search order is the same as the order of appearance of libraries on the `fuse` command line. See [Verilog Search Order, page 40](#) for more information.

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

Instantiating Mixed Language Components

In a mixed language design, you can instantiate a Verilog module in a VHDL design unit or a VHDL module in a Verilog design unit as described in the following subsections.

Mixed Language Boundary and Mapping Rules

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.

Instantiating 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, review the [Port Mapping, page 47](#).

Port Mapping

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

Supported Port Types

Table 3-1 lists the supported port types.

Table 3-1: Supported Port Types

VHDL ¹	Verilog ²
IN	INPUT
OUT	OUTPUT
INOUT	INOUT

1. Buffer and linkage ports of VHDL are not supported.
2. Connection to bidirectional pass switches in Verilog are not supported. Unnamed Verilog ports are not allowed on mixed design boundary.

Table 3-2 shows the supported VHDL and Verilog data types for ports on the mixed language design boundary.

Table 3-2: Supported VHDL and Verilog data types

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

ISim supports the following VHDL generic types (and their Verilog equivalents):

- integer
- real
- string
- boolean

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

VHDL and Verilog Values Mapping

Table 3-3 lists the Verilog states mappings to `std_logic` and `bit`.

Table 3-3: Verilog States mapped to `std_logic` and `bit`

Verilog	<code>std_logic</code>	<code>bit</code>
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.

Table 3-4 lists the VHDL type `bit` mapping to Verilog states.

Table 3-4: VHDL `bit` Mapping to Verilog States

<code>bit</code>	Verilog
0	0
1	1

Table 3-5 lists the VHDL type `std_logic` mappings to Verilog states.

Table 3-5: VHDL `std_logic` mapping to Verilog States

<code>std_logic</code>	Verilog
U	X
X	X
0	0
1	1
Z	Z
W	X
L	0
H	1
-	X

Because 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

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);
  );
```

Timing Simulation (Gate-Level Simulation)

Before launching a timing simulation, you must have a timing simulation model and a Standard Delay File (SDF) for back-annotation. Use the NetGen tool to generate these files. See “Generating Gate-Level Netlist (Running NetGen)” in the *Synthesis and Simulation Design Guide (UG626)*. [Appendix D, Additional Resources](#) contains a link to this document.

Timing Simulation of a Verilog Design on the Command Line

In a timing simulation of a Verilog design, the following rules apply:

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

ISim Executable Command

Note: Commands are case-sensitive.

The following subsections provide an overview of the ISim executable, compilation, and elaboration commands as well as the command syntax, and command options.

The ISim executable file is user-defined. Running the file at the command line invokes a simulation. You can set the executable name using the fuse command `-o` option. If not user-defined, the default executable name is `x.exe`.

ISim Executable Syntax

The syntax for this command is:

```
<executable_name>.exe <options>
```

Where:

- `<executable_name>.exe` is user-defined or, by default, `x.exe`.
- `<options>` are the options specified in [Table 3-6](#).

ISim Executable Command Options

Table 3-6 lists the command options used by ISim executable.

Table 3-6: ISim Command Options

Option	Description
-f <cmd_file>	Lets you save 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 GUI.
-h	Displays all command line options and usage.
-intstyle [ise xflow silent default]	Use one of the specified styles for printing messages: <ul style="list-style-type: none"> ise formats messages for the ISE Console xflow formats messages for XFLOW. silent suppress all messages. default is the specified default message setting
-log <file_name>	Generates a log file with the specified <file_name>.
-maxdeltaid <number>	Specifies the maximum delta number in an integer.
-nolog	Blocks log file generation.
-sdfnowarn	Blocks the display of SDF warnings.
-sdfnoerror	Treats SDF errors as warnings.
[-sdfmin -sdftyp -sdfmax] <root=file>]	Specifies the type of delays for ISim to use: <ul style="list-style-type: none"> -sdfmin - Annotates <file> at <root> with minimum delay. -sdftyp - Annotates <file> at <root> with standard delay. -sdfmax - 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>	Specify a Tcl script file to be run after simulation is loaded. The <file_name> specifies the name of file containing Tcl commands. To quit simulation after executing the Tcl commands in <file_name>, you must include the <code>quit</code> Tcl command as the last command in the <file_name>.
-testplusarg <string string_value>	When the simulator matches this command line argument string with the <code>\$test\$plusarg</code> or <code>\$value\$plusarg</code> system function of a Verilog design file, the test or design behavior change associated with the system function is run where <string> is any string. For example, <code>-testplusarg HELLO</code> , if a Verilog file uses <code>(\$test\$plusargs ("HE"))</code> , the function returns true. <string_value> is an appropriate string for the Verilog format specifiers. It provides a value to the variable in the <code>\$value\$plusargs</code> system function call. For example, <code>-testplusarg FINISH=10000</code> , if a Verilog file uses <code>(\$value\$plusargs ("FINISH=%d", stop_clock))</code> and, when Verilog format specifier <code>%d</code> matches 10000, and <code>stop_clock</code> gets values 10000, the function returns 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.

Table 3-6: ISim Command Options (Cont'd)

Option	Description
-transport_int_delays	Use transport model for interconnect delays. No pulse rejection on interconnect delays.
-vcdfile <vcd_file>	Verilog-only option. Specifies the VCD output file for Verilog projects. The default filename is dump.vcd.
-vcdunit <unit>	Verilog-only option. Specifies the VCD output time unit. Unit values are fs, ps, ns, us, ms, and sec. The default is ps.
-view <waveform_file>.wcfg	Used in combination with the -gui switch to open the specified waveform file in the ISim GUI.
-wdb <waveform_file>.wdb	Saves simulation data to the specified WBD file. For example: x.exe -wdb my.wdb saves the simulation data to my.wdb instead of the default isimgui.wdb.

ISim Executable Command Examples

```
<executable_name>.exe -tclbatch <tcl_file_name> -sdfmin  
<instance>=<sdf_file_name>
```

Where:

- <executable_name>.exe is the simulation executable called x.exe unless otherwise specified with the fuse -o switch.
- -tclbatch is an optional switch to use when you want to run additional Tcl commands.
- -sdfmin is the type of delay (minimum) to use.
- <instance> is the hierarchical path name of the instance at which the Standard Delay File (SDF) back annotation needs to be done.
- <sdf_file_name> is the filename of the SDF file to annotate.

fuse, vhpcomp, and vlogcomp Command Options

Table 3-7 lists the fuse, vhpcomp, and vlogcomp command options.

Table 3-7: fuse, vhpcomp, and vlogcomp Command Options

Option	Description
-d <macro_definition>=<value>=<value>	Verilog-only option. Define the macros used in Verilog files, and any required values. You can specify more than one -d option. Note: Ensure that there is no space between the = and the value; the space would be interpreted as part of the value. Double-quote any path that contains white spaces.
-f <cmd_file>	Lets you save command options in a text file for future use. This option reads and executes the saved options that are 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" applies the value of 10 on the top-level parameter before elaboration.

Table 3-7: fuse, vhpcomp, and vlogcomp Command Options (Cont'd)

Option	Description
-gui	Launches the ISim GUI.
-h	Displays all command line options and usage.
-hil_zynq_ps	Use this to enable Zynq Processor System (PS) Hardware In Loop (HIL) simulation.
-hwcosim_board <arg>	Hardware Co-Simulation (HWCoSim) option specifies the board name.
-hwcosim_clock <arg>	Specifies the clock port name on the HWCoSim instance.
-hwcosim_instance <arg>	HWCoSim option specifies the hierarchical name of the instance to be run on the FPGA. For example: /testbench/UUT.
-hwcosim_no_combinatorial_path	HWCoSim option to speed up simulation if the design run on FPGA does not have a purely combinatorial path from any input to any output.
-hwcosim_incremental <arg>	HWCoSim option skips the implementation phase and reuses the previously created bit file. Allowed values are: 0 and 1 (Default: 0)
-i "<include_path>"	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 initialization 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 default]	Use one of the specified styles for printing messages: <ul style="list-style-type: none"> • ise formats messages for the ISE Console • xflow formats messages for XFLOW • silent suppress all messages • default is the specified default message style
-ise <file>	Lets you specify an ISE file.
[-L -lib <search_lib>[=<lib_path>]]	Specifies other libraries and optionally the physical path name for those libraries. You can use multiple -L switches that are treated as resource libraries. The physical path provided through -L overrides mappings provided by the xilinxsim.ini file. The <search_lib> is the logical name of the specified library optionally followed by the <lib_path>, the path to the physical library. Note: Ensure that there is no space between the = and the value; the space would be interpreted as part of the value. Double-quote any path that contains white spaces.

Table 3-7: fuse, vhpcomp, and vlogcomp Command Options (Cont'd)

Option	Description
-log <file_name>	Generates a log file with the specified <file_name>.
-maxdeltaid <number>	Specifies the maximum delta number in an integer.
-maxdelay	Verilog-only option. Specifies that if fuse calls vlogcomp, it should use worst case delays.
-maxdesigndepth <depth>	Overrides maximum design depth allowed by the fuse elaborator. If a design exceeds the depth, fuse errors out. This option lets you increase the depth in case fuse has inaccurately determined that a design has infinite recursive instantiation.
-mindelay	Verilog-only option. 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. Values are on, off, or an integer greater than 1. Default is on, where the compiler automatically chooses a number based upon the 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 results in a faster simulation runtime. The default is to generate HDL debug units.
-nolog	Blocks log file generation.
-nospecify	Verilog-only option. Disables specify block functionality.
-notimingchecks	Verilog-only option. 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_library>/<mod_name>/<platform>/x.exe, where: <ul style="list-style-type: none"> • <work_library> is the work library. • <module_name> is the first specified top module. • <platform> is the operating system.
-override_timeprecision	Verilog-only option. Overrides the time precision (unit of accuracy) of 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 tool. A <prj_file> must have a .prj extension.

Table 3-7: fuse, vhpcomp, and vlogcomp Command Options (Cont'd)

Option	Description
-rangecheck	<p>VHDL-only option. Specifies value range check to be performed on VHDL assignments. This option does not affect index range checking for arrays. ISim always checks an index into an array for being within the allowed range. For example:</p> <ul style="list-style-type: none"> If a signal is declared as positive, fuse checks that the signal is not assigned a negative number. If a signal is declared as <code>std_logic</code>, vhpcomp generates output to check that the signal is assigned only valid <code>std_logic</code> values (U,X,0,1,Z,W,L,H,-). <p>Note: This option does not affect the checking of index ranges. The simulator always checks the index ranges.</p> <p>By default -rangecheck is turned off.</p>
-sdfnoerror	Treats SDF errors as warnings.
-sdfnowarn	Blocks the display of SDF warnings.
[-sdfmin] [-sdfmax] [-sdfmax] <root=file>	<p>Specifies the type of delays to use:</p> <ul style="list-style-type: none"> -sdfmin - Annotates <file> at <root> with minimum delay. -sdfmax - Annotates <file> at <root> with standard delay. -sdfmax - Annotates <file> at <root> with maximum delay.
-sdfroot <root_path>	Sets the default location in the design hierarchy in which to apply the SDF annotation.
-sourcelibdir <directory_name>	Specifies the source directory for library modules.
-sourcelibext <file_extension>	Specifies the file extension for source files for modules. The -sourcelibdir option provides the location for these files.
-sourcelibfile <file_name>	Specifies the filename for library modules.
-timeprecision_vhdl <time_precision>	<p>VHDL-only option. 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). Default is 1ps.</p>
-timescale <time_unit/ time_precision>	<p>Verilog-only option. Specifies the default timescale for Verilog modules that do not have an effective timescale:</p> <ul style="list-style-type: none"> <time_unit> is the unit of measurement of delays <time_precision> is the unit of accuracy. <p>Both <time_unit> and <time_precision> are entered as number (1 10 100 ...) followed by unit (fs ps ns us ms s). Default is 1ns/1ps.</p>
-typdelay	Verilog-only option. Specifies that if fuse calls vlogcomp, it should use typical delays.
-timeprecision_vhdl <time_precision>	<p>VHDL-only option. 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). Default is 1ps.</p>

Table 3-7: fuse, vhpcomp, and vlogcomp Command Options (Cont'd)

Option	Description
<p>-v [-verbose] <value></p>	<p>Specifies the verbosity level for printing messages. Values are 0, 1, 2. Default is 0. For example:</p> <p>fuse -v 1 prints useful debugging information, which can help to identify problems in ISim compilers. The verbosity level 1:</p> <ul style="list-style-type: none"> • Dumps the library mapping as seen by the 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 that affect the behavior of the compiler. • Gets the list of loaded shared objects by the compiler. • Dumps operating system information, including version number and processor. • Dumps path to the GCC compiler being used to compile the generated code.
<p><verilog_files>...</p>	<p>Specifies the Verilog file to be compiled.</p>
<p><vhdl_files>...</p>	<p>Specifies one or more VHDL source files to be compiled.</p>
<p>-version</p>	<p>Prints the compiler version.</p>
<p>-wdb <waveform_file>.wdb</p>	<p>Saves simulation data to the specified WBD file.</p> <p>For example: x.exe -wdb my.wdb saves the simulation data to my.wdb instead of the default isimgui.wdb.</p>
<p>-work [<work_library>[=<library_path>]] <filenames>...</p>	<p>Specifies the work library, and optionally, the physical path for the work library. The physical path overrides the mappings provided by the xilinxsim.ini file. The default work library is the logical library /work.</p> <p>The <work_library> is the logical name of the specified work library optionally followed by <library_path>, the path to the physical library. For example: mywork=C:/home/worklib.</p> <p>Note: Ensure that there is no space between the = and the value; the space would be interpreted as part of the value. Double-quote any path that contains white spaces.</p>

Pausing a Simulation

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

To pause a running simulation, do one of the following:

- Select **Simulation > Break**.
- Click the **Break** button.
- Type **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 are not compiled with the `-nodebug` switch.

The simulation can be resumed at any time by using the **Run All**, **Run**, or **Step** commands. See [Stepping Through a Simulation, page 77](#).

Saving Simulation Results

ISim saves the simulation results of the objects (VHDL signals, or Verilog reg or wire) being traced 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. See [Chapter 4, Waveform Analysis](#), for more information.

Saving a Database to a WDB File

When ISim is launched from the:

- ISE tool or the PlanAhead tool, the WDB file is named according what is specified in the ISim Properties dialog box.
- Command line, use the `-wdb` switch 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 simulation results in WDB files `isim1.wdb`, `isim2.wdb`, and so forth.

Note: The name of the database file cannot be changed while running simulation.

Saving a Wave Configuration to a WCFG File

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.

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 [Working with the Wave Configuration in Chapter 4](#).

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

Closing Simulation

You can terminate a simulation with one of the following commands.

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

Chapter 4

Waveform Analysis

Before starting the waveform analysis, you need to open the ISim GUI, using one of the following methods:

- In read-only mode to view or analyze the data from a previous simulation, see [Opening a Static Simulation, page 74](#).
- From the ISE® or the PlanAhead™ tools, a wave configuration with top-level signals displays. You can then proceed to add additional signals, or run the simulation. See [Running Simulation in Chapter 3](#).
- From the command line, run the simulation executable with the `-gui` switch, and an empty wave configuration displays. You must add signals to the wave configuration before you run simulation. See [Running Simulation in Chapter 3](#).

ISim populates design data in other areas of the GUI, such as the Objects , and the Instances and Processes panel.

Working with the Wave Configuration

You can add signals and buses to the wave configuration file, then save that configuration to a WDB file. See [Opening a Wave Configuration and Waveform Database in Chapter 5](#).

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 GUI, or using Tools Command Language (Tcl) commands in the Console panel.

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

Adding Signals 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 panel, use the `wave add` command.

Adding Signals Using Tcl

1. Optionally, you can first identify the objects you want to add by exploring the design hierarchy in the Instances and Processes panel and the Objects panel, as described above, or type `scope` in the Console panel.
2. In the Console panel, type `wave add` to add an individual object or a group of objects.

Wave Configurations and WCFG Files

Though both a wave configuration and a WCFG file refer to customizing lists of waveforms, there is a conceptual difference between them.

- The wave configuration is an object with which you can work that is loaded into memory.
- The WCFG file is the saved form of a wave configuration on disk.

Wave Configuration Names and WCFG File Names

A wave configuration can have a name or be untitled. The name shows on the tab of the wave configuration window.

- When saving a wave configuration to a WCFG file using a GUI Tcl command, the WCFG file takes the supplied name as a command argument.
- When loading a wave configuration from a WCFG file, the wave configuration uses the name of the WCFG file.

Adding Copies of Signals or Buses

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

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

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

Customizing the Wave Configuration

You can customize the Wave configuration using the features that are listed and briefly described in [Table 4-1](#); the feature name links to the subsection that fully describes the feature.

Note: In your PDF reader, turn on **Previous View** and **Next View** Buttons to navigate back and forth.

Table 4-1:

Feature	Description
Cursors	The main cursor and secondary cursor in the Wave window let you display and measure time, and they form the focal point for various navigation activities.
Markers	You can add markers to navigate through the waveform, and to display the waveform value at a particular time.
Dividers	You can add a divider to create a visual separator of signals.
Groups	You can add a group, that is a collection to which signals and buses can be added in the wave configuration as a means of organizing a set of related signals.
Virtual Buses	You can add a virtual bus to your wave configuration, to which you can add logic scalars and arrays.
Renaming Objects	You can rename objects, signals, buses, and groups.
Display Names	You can display the full hierarchical name (long name), the simple signal or bus name (short name), or a custom name for each signal.
Radixes	The default radix controls the bus radix that displays in the wave configuration, Objects panel, and the Console panel.
Bus Bit Order	You can change the Bus bit order from Most Significant Bit (MSB) to Least Significant Bit (LSB) and vice versa.

Cursors

Cursors are used primarily for temporary indicators of time and are expected to be moved frequently, as in the case when you are measuring the time between two waveform edges. For more permanent indicators, used in situations such as establishing a time-base for multiple measurements, add markers to the Wave window instead. See [Markers, page 62](#) for more information.

Placing the Main Cursor

You can place the main cursor with a single click in the Wave window.

Placing the Secondary Cursor

Place the secondary cursor as follows:

1. 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.

2. Press **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 appears.

Moving a Cursor

To move a cursor, hover over the cursor until you see the grab symbol, and click and drag the cursor to the new location.

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

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


A secondary cursor can be hidden by clicking anywhere in the Wave window where there is no cursor, marker, or floating ruler.

Markers

You can add, move, and delete markers.

Adding a Marker




You add markers to the wave configuration at the location of the main cursor.

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

- Click the marker label (at the top of the marker) and drag it to the location.
 - The drag symbol indicates that the marker can be moved. As you drag the marker in the Wave window, you see a hollow or filled-in circle if the **Snap to Transition** button is selected, which is the default behavior. 
 - A filled-in circle indicates that you are hovering over a transition of the waveform for the selected signal or over another marker. 
 - For markers, the filled-in circle is white.
 - A hollow circle indicates that you are between transitions in the waveform of the selected signal. 
- Release the mouse key to drop the marker to the new location.

Deleting a Marker

You can delete one or all markers with one command.

- Right-click over a marker.
- 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: You can also use the **Delete** key to delete a selected marker.

Use **Edit > Undo** to reverse a marker deletion.

Dividers

Dividers create a visual separator between signals.

Adding Dividers

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

- In a **Name** column of the Wave window, click a signal to add a divider below that signal.
- From the context menu, select **Edit > New Divider**, or right-click and select **New Divider**.

The change is visual and nothing is added to the HDL code. The new divider is saved with the wave configuration file when you save the file.

Changing Dividers

The following changes can be made to a divider:

- Dividers can be renamed. See [Renaming Objects, page 65](#).
- Dividers can be moved to another location in the waveform by dragging and dropping the divider name.

Deleting Dividers

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


Groups

A Group is a collection of expandable and collapsible categories, to which signals and buses can be added in the wave configuration to organize related sets of signals. The group itself displays no waveform data but can be expanded to show its contents or collapsed to hide them.

Adding a Group

To add a Group:

1. In a wave configuration, select one or more signals or buses 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 group 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 or bus name.

The new group and its nested signals or buses is saved when you save the wave configuration file.

Changing Groups

You can change groups as follows:

- You can rename Groups. See [Renaming Objects, page 65](#).
- You can move Groups to another location in the **Name** column by dragging and dropping the group name.

Removing a Group

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

Caution! Pressing the **Delete** key removes the group and its nested signals and buses from the wave configuration.


Virtual Buses

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

Adding Virtual Buses

To add a virtual bus:

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

The virtual bus is represented with the **Virtual Bus** button . 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 or bus name. The new virtual bus and its nested signals or buses are saved when you save the wave configuration file.

Changing Virtual Buses

The following changes can be made to a virtual bus:

- You can rename them. See [Renaming Objects, page 65](#).
- You can move it to another location in the waveform by dragging and dropping the virtual bus name.

Removing Virtual Buses

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 removes 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.

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 type a new name.

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

Display Names

You can display the full hierarchical name (long name), the simple signal or bus name (short name), or a custom name for each signal. The signal or 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.

Changing Display Names

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, page 65](#).

The name changes immediately according to your selection.

Radixes

The default radix controls the bus radix displayed in the wave configuration, the Objects panel, and the Console panel.

Changing the Default Radix

The default radix is binary. To change the radix, do the following:

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

Changing an Individual Radix

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

1. Right-click a bus in the Objects panel.
2. Select **Radix** and the format you want from the drop down 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 Tcl command, `isim set radix`.

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 the 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

You can navigate the wave configuration using a variety of options, which include:


- [Expanding and Collapsing a Hierarchy](#)
- [Zooming In and Out](#)
- [Using the Floating Ruler](#)
- [Displaying Waveform Values With Markers](#)
- [Displaying Waveform Values at Signal Transitions](#)
- [Measuring Time with Cursors](#)
- [Using Go To Time](#)
- [Using Show Drivers](#)

Expanding and Collapsing a Hierarchy

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

Using the Arrows

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

Click the collapse arrow to collapse the hierarchy. 

Using the Menu

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 selected object.

Zooming In and Out

Use the zoom functions to view your wave configuration in the Wave window as needed. See the [View Menu and Toolbar, page 11](#).

Using 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.

You can display (or hide) a floating ruler and move it to a location in the Wave window. The time base (time 0) of the floating ruler 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.

1. Do either of the following to display or hide a floating ruler:
 - Place the secondary cursor.
 - 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 displays each time the secondary cursor is placed or marker is selected.


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.

1. In the wave configuration in the Wave window, add one or more markers, as described in [Adding a Marker, page 62](#).
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** 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** 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.

Using 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** button. 
3. 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.
4. Repeat step 2 as necessary.
5. To go back to the previous transition, select **View > Cursors > Previous Transition** or click the **Previous Transition** button. 
6. The marker moves back to the previous transition for the selected signal. The values for *all* signals at that time display in the Value column.
7. 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.

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. This feature assists 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** button



6. 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 the Secondary Cursor](#), 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.

1. Display the floating ruler using a selected marker as the time base.
2. Add additional markers.
3. Move markers to a location in 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 by clicking, and thus selecting, another marker.

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 lets you jump the main cursor to a particular time in the wave configuration.

Going To a User-Specified Time

With a wave configuration open:



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

The **Go To Time** dialog box opens in the Wave window below the wave configuration.

2. Type the time and time unit to which to jump the cursor, or select a time/time unit from the drop down list, if available.
3. Press **Enter**.

Going to Time 0 or Latest Time

With a wave configuration open:

1. Select the **Go To Time 0** button to move the cursor to time 0 in the wave configuration. 
2. Select the **Go To Latest Time** 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 helps 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

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.

The Console panel lists the drivers for the object or signal. When there is no driver, a Console message 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.

Printing 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 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.

Printing

1. Select **File > Print**.
2. In the **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, and click **Print**.

The wave configuration prints according to the print settings.

Using Custom 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. See [ISim Color Preferences in Chapter 2](#).

You can use pre-defined color schemes or create your own color scheme.

Changing an individual signal or bus color overrides the general color setting for signal or bus waveforms.

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

Chapter 5

Viewing Simulation Results

A live simulation consists of the following:

- A waveform database file (WDB), which contains all simulation data
- A wave configuration file (WCFG), which contains the order and settings associated with objects in the wave configuration

Waveform Databases and Configuration Files

A WDB opens automatically when you run a simulation. For more information about running a simulation, see [Running Simulation in Chapter 3](#).

When you run a simulation from the ISE® tool or the PlanAhead™ tool, a default wave configuration opens automatically. 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. See the following subsections and [Running Simulation in Chapter 3](#) for more information.

Open a Wave Configuration in the GUI

1. Select **File > Open**.
2. Select **.wcfg** from the file type list.
3. Select the file to open, and 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.

Opening a Wave Configuration from a Command Prompt

Because 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 GUI
- `-wdb <wdb>.wdb` specifies the filename where the simulation data is stored
- `-view <wcfg>.wcfg` opens the specified waveform file in the GUI

The 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 [Working with the Wave Configuration in Chapter 4](#).

Opening a Static Simulation

A static read-only simulation consists of a:

- Wave configuration (WCFG) file, which contains the order and settings associated with objects the display in the wave configuration, and
- WDB 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 [Simulating the Design in Chapter 3](#).

Opening a Wave Configuration and 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 WDB.

To open a wave configuration and the waveform database, do the following:

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.

Alternatively, you can type **isimgui.exe -open <wcfg_file>.wcfg**:

Where:

- `isimgui.exe` is the application executable
- `-open` instructs the tool open the specified file.
- `-view <wcfg>.wcfg` opens the specified waveform file in the GUI

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 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 issues a warning for any object name that is present in the WCFG but not found in the WDB file and displays only those objects that match.

In a command prompt, 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 graphical 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, type:

```
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:

1. In a command prompt, type:

```
isimgui.exe
```

This opens 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.

Alternatively, you can type:

```
isimgui.exe -open <wdb_file>.wdb
```

Where:

- `isimgui.exe` is the application executable.
- `-open <wdb_name>.wdb` opens the specified waveform database file in the GUI.

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**.


Chapter 6

Debugging at the Source Level


You can 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 issues might be occurring. Available strategies for debugging 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, run the **Step** command 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](#), page 78.

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 [Using Breakpoints](#), page 78.

1. To step through a simulation:
 - From the current running time, do one of the following:
 - Select **Simulation > Step**.
 - Click the **Step** button.
 - Type the `step` command at the Console prompt. 

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

- From the start (0 ns), restart the simulation. Use the **Restart** command to reset time to the beginning of the test bench. See [Running Simulation in Chapter 3](#).
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 steps 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.

Using Breakpoints



A breakpoint is a user-determined stopping point in the source code used for debugging the design. Breakpoints are particularly helpful when debugging larger designs for which debugging with the Step command (stopping the simulation for every line of code) might be too cumbersome and time consuming.

You can set breakpoints in executable lines in your HDL file so you can run your code continuously until the source code line with the breakpoint is reached.

Note: You can set breakpoints on lines with executable code only. If you place a breakpoint on a line of code that is not executable, the breakpoint is not added.

Setting a Breakpoint

To set a breakpoint, do the following:


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

Note: Alternatively, you can right-click a line of code, and select **Toggle Breakpoint**.

After the procedure completes, a simulation breakpoint icon opens next to the line of code, and a list of breakpoints is available in the Breakpoints panel.

Debugging Your Design Using Breakpoints

1. Open the HDL source file. See [Opening HDL Source Files, page 23](#).
2. Set breakpoints on executable lines in the HDL source file, as described in [Setting a Breakpoint, page 78](#).
3. Repeat steps 1 and 2 until all breakpoints are set.
4. Click the Wave window to return to the waveform.
 - To run from the beginning, use the **Simulation > Restart** command.
 - Use the **Simulation > Run All** or **Simulation > Run for Specified Time** command.

The simulation runs until a breakpoint is reached, then stops. The HDL source file displays, and the breakpoint stopping point is indicated with a yellow arrow. 

5. Click the Wave window again to return to the waveform to determine if the design behavior (such as the signal value change) is as expected at the breakpoint.
6. Repeat the steps to advance the simulation, breakpoint by breakpoint, until you are satisfied with the results.

A controlled simulation runs, 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.

Deleting Breakpoints

You can delete a single breakpoint or all breakpoints from your HDL source code.

Deleting a Single Breakpoint


Delete a single breakpoint using one of the following methods:

- Click the **Breakpoint** button. 
- At the Tcl prompt:
 - Type `bp list` to list all breakpoints in your design and shows each breakpoint index number and line number.
 - Type `bp del` or `bp remove` followed by the breakpoint index number of the breakpoint to remove. See [bp](#), page 104.

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

Removing All Breakpoints

Use one of the following methods to remove all breakpoints:

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

Writing Activity Data for Power Consumption

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 PAR. Map, PAR, and XPower Analyzer all work with switching activity data generated from both RTL and post-PAR simulation.

For better accuracy in power analysis and power optimized implementation, it is recommended to use switching activity data generated from a post-PAR simulation. The data then matches 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-PAR simulation) for both power analysis and power driven implementation. However, only activities for the inputs and outputs of the design are taken into account. The tools use their vector-less analysis algorithms to estimate activities of the internal nodes of the design.

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. [Appendix D, Additional Resources](#), contains links to these documents

Creating Activity Files

You can write out two types of stimulus files:

- 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 (such as, power analysis or power driven implementation) because 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 out a switching activity file:

1. Create the activity file 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 the same time.

3. When simulation ends, close the SAIF or VCD file by issuing the `saif` or `vcd` command. Examples are:

```
saif close
```

```
vcd dumpoff
```

4. Retrieve the SAIF or VCD file from the working directory for use in another tool.

Chapter 8

Using Hardware Co-Simulation

Hardware Co-Simulation (HwCoSim) is a complementary flow to the tool-based HDL simulation. This feature lets you simulate a design or a portion of the design and offload that simulation to hardware. It can accelerate the simulation of a complex design and verify that the design works in hardware.

Prerequisites

Hardware co-simulation has the following requirements:

- Xilinx® ISE® Design Suite 14.x (any edition)
- 32-bit or 64-bit Windows or Linux
- An FPGA board with a JTAG header

Supported FPGA devices are:

- Virtex®-4, Virtex-5, Virtex-6, Virtex-7
- Spartan®-3, Spartan-3E, Spartan-3A, Spartan-3AN, Spartan-3A DSP, Spartan-6
- Artix™-7 and Kintex™-7
- A Xilinx Parallel Cable IV or Platform Cable USB

Use Models

HwCoSim supports two use models: one for pure logic-based designs and another for hybrid designs.

Pure Logic-based Designs

Pure logic-based designs are co-simulated in a lockstep fashion with ISim. The modules under co-simulation typically have the following characteristics:

- Composed of LUTs, FFs, block RAMs, and DSP primitives only
- Port controlled by ISim and accessible from the tool test bench (no external I/Os)
- Functionality of the module is irrelevant of the clock frequency at which it operates (there is no need to run on a continuous clock or a clock at a specific frequency)

The pure logic, lockstep-based HwCoSim provides the following advantages:

- Simulation acceleration for computational-intensive designs
- In-hardware functional verification
- Bit-and-cycle accurate with respect to pure tool simulation

Hybrid Designs

The pure logic-based design use model is simple and trivial to set up, but is not suitable for designs that require hard IPs, external I/Os, and specific clock frequencies. ISim HWCoSim provides a hybrid co-simulation flow that supports designs with the following characteristics:

- Composed of hard IP blocks, DCMs/PLLs, and MGTs.
- Some clocks are in lockstep with the tool simulation using emulated clock sources, and other clocks are free-running using external clock sources.
- Some ports can be mapped to external I/Os, which are neither controlled by ISim nor accessible from tool test bench.

The hybrid co-simulation flow offers the following advantages:

- Accelerates simulation
- Verifies functionality in hardware
- Allows customized or complicated software and hardware interactions beyond a typical co-simulation setup.

Limitations

HWCoSim has the following limitations:

- Only one instance in a design can be selected for hardware co-simulation, and it cannot be the top-level test bench itself.
- The selected instance for hardware co-simulation must be able to be synthesized using XST, and must be able to be implemented on the target FPGA device of the selected board.

The lockstep hardware co-simulation has additional restrictions on clocking and I/Os:

- The co-simulation instance in hardware is clocked with an emulated clock source that ISim controls, and is asynchronous to the simulation. Thus, the co-simulation does not exactly model the design scenario running in hardware, or serve as a timing simulation.
- The instance under co-simulation cannot have access to external I/Os or Multi-Gigabit Transceivers (MGTs), nor can it instantiate primitives (such as DCMs/PLLs) that require a continuous clock or a clock at a specific frequency.
- All ports of the instance under co-simulation must be routable to a slice register or LUT. Certain resources on the FPGA require dedicated routes, such as to an IOB or to certain port of a primitive, and thus cannot be wired to any port of the instance under co-simulation.

Usage for Compilation

As with tool-based HDL simulation, you first compile a design into a simulation executable before performing HWCoSim. Invoke the `fuse` command for compilation, using the command line or through the ISE tool or the PlanAhead™ tool to produce the co-simulation executable, a HWCoSim bitstream, and a co-simulation project file.

fuse Command Line Flow

The fuse command provides some additional compiler options to compile a design for hardware co-simulation.

Usage:

```
fuse -prj <project file> <top level modules>
    -hil_zynq_ps
    -hwcosim_board <board>
    -hwcosim_clock <clock>
    -hwcosim_constraints <constraints file>
    -hwcosim_incremental [0|1]
    -hwcosim_instance <instance>
    -hwcosim_no_combinatorial_path
```

Where:

- `-hil_zynq_ps` enables Zynq™ Processor System (PS) Hardware In Loop (HIL) simulation.
- `-hwcosim_board` specifies the identifier of the hardware board to use for co-simulation. See [Board Support, page 92](#) for a list of supported boards.
- `-hwcosim_clock` specifies the port name of the clock input for the instance. For a design with multiple clocks, specify the fastest clock using this option so that ISim can optimize the simulation. Other clock ports are treated as regular data ports.
- `-hwcosim_constraints` (optional) specifies the custom constraints file that provides additional constraints for implementing the instance for hardware co-simulation. A custom constraints file is also used in the hybrid co-simulation flow to specify which ports of the instance are mapped to external I/Os or clocks.
- `-hwcosim_incremental` (optional) specifies whether fuse reuses the last generated hardware co-simulation bitstream and skips the implementation flow.
- `-hwcosim_instance` specifies the full hierarchical path of the instance to co-simulate in hardware
- `-hwcosim_no_combinatorial_path` speeds up simulation if the design run on FPGA does not have a purely combinatorial path from any input to any output.

Tools Flow

1. Ensure you have **ISim** selected as the simulator for the project. Switch to the Simulation view.
2. From the Hierarchy pane, select the instance to co-simulate in hardware and right-click to show the context menu.
3. From the context menu, select **Source Properties** to open the **Source Properties** dialog box.
4. In the **Source Properties** dialog box, select **Hardware Co-Simulation** from the category list.

5. Set the following properties for hardware co-simulation:
 - Check the **Enable Hardware Co-Simulation** check box.

Because only one instance can be enabled for hardware co-simulation, enabling a instance for hardware co-simulation disables any other instance that has been previously enabled for hardware co-simulation.

- In the **Clock Port** field, specify the name of the clock port on the instance. For an instance with multiple clocks, specify the name of the fastest clock port.
- From the **Target Board for Hardware Co-Simulation** drop down list, select a board. The list shows only the boards with an FPGA of same project device family.
- If a previous hardware co-simulation bitstream is available and the instance under co-simulation remains unchanged, check the **Reuse Last Bitstream File** check box to skip the implementation flow for hardware co-simulation.
- Click **OK**.

Notice that the instance enabled for hardware co-simulation is now marked with a special icon.



Select the test bench module in the hierarchy pane to the start the simulation. Hardware Co-Simulation must be started at a level above the instance that is selected for co-simulation.

6. In the Instances and Processes panel of the test bench, double-click **Simulate Behavior Model** to start the compilation and simulation process.
7. Open the **Process Properties** for the Simulate Behavior Model to specify any additional options for ISim before starting the compilation and simulation process. [Figure 8-1](#) through [Figure 8-3](#), page 88 display steps 1 through 8:

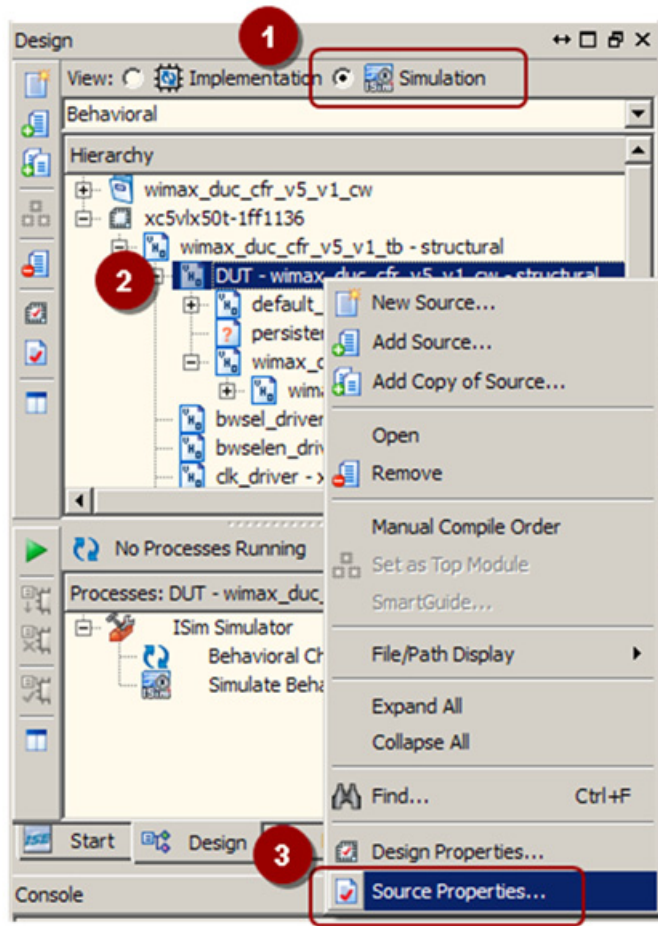


Figure 8-1: Steps 1, 2, and 3

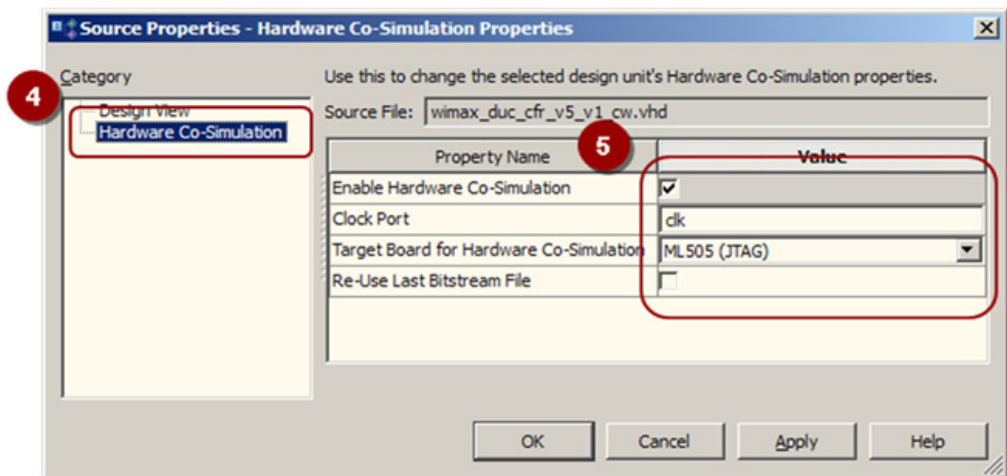


Figure 8-2: Steps 4 and 5

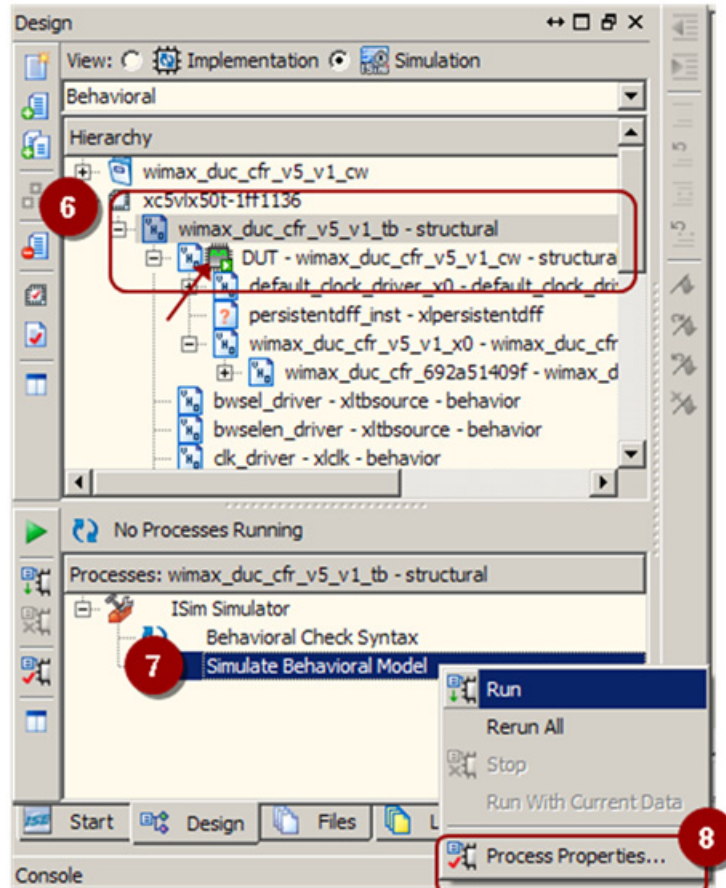


Figure 8-3: Steps 6, 7, and 8

Hybrid Co-Simulation Flow

To use external pins and free-running clocks, you need to provide a custom constraints file in UCF format. The flow currently reads in a constraints file specified through the `-hwcosim_constraints` option to determine which pins are mapped to FPGA IOBs.

1. Decide which portion of your design to run in lockstep with ISim simulation and which to be free-running. [Figure 8-4, page 89](#) outlines the concept:

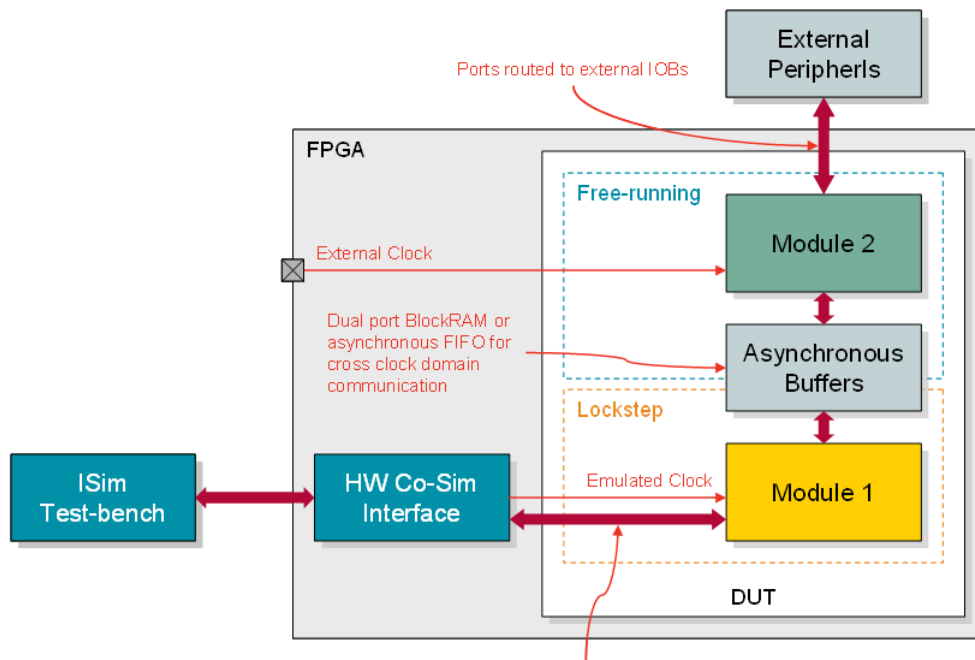


Figure 8-4: Determining Lockstep and Free-running Design Portions

2. Copy the original design constraints file and use it as the basis for the custom constraints file.
3. Modify the custom constraints file to comment out the LOC constraints for those pins that are controlled by ISim. Other pins with LOC constraints are assumed to be external.
4. As an example, for a FIFO design that you want to single-step the write side and free-run the read side, such as:

```
module FIFO (WCLK, WDATA, WE, RCLK, RDATA, RE);
```

The isim_hwcosim.ucf would be:

```
# The following pin LOCs commented out as they are driven by ISim
# NET "WCLK" PERIOD = 5 ns HIGH 50%;
# NET "WCLK" LOC = "A1"; # <--- this becomes single-step clock
# NET "WDATA" LOC = "A2"; # <--- these are accessible from ISim
testbench
# NET "WE" LOC = "A3";
NET "RCLK" PERIOD = 10 ns HIGH 50%;
NET "RCLK" LOC = "B1"; # <-- this becomes free-running clock
NET "RDATA" LOC = "B2"; # <-- these go to external IOBs
NET "RE" LOC = "B3";
```

Hardware Board Usage

The following procedure describes how to install and set-up the hardware to run hardware co-simulation.

1. Ensure the hardware board is powered off.
2. If you are using a Xilinx® Parallel Cable IV, follow steps 2a through 2d.
 - a. Connect the DB25 Plug Connector on the Xilinx Parallel Cable IV to the IEEE-1284 compliant PC Parallel (Printer) Port Connector.
 - b. Using the narrow (14 pin) 6" High Performance Ribbon cable, connect the pod end of the Xilinx Parallel Cable IV to the FPGA JTAG header on the board.
 - c. Connect the attached Power Jack cable to the Keyboard/Mouse connector on the PC.
 - d. If necessary, connect the male end of the Keyboard/Mouse cable to the associated female connector on the Xilinx Power Jack cable (splitter cable).
3. If you are using a Xilinx Platform Cable USB, follow steps 3a and 3b.
 - a. Connect the Xilinx Platform Cable USB to a USB port on the PC.
 - b. Using the narrow (14 pin) 6" High Performance Ribbon cable, connect the pod end of the Xilinx Platform Cable USB to the FPGA JTAG header on the board.
4. If your board supports Point-to-Point (P2P) Ethernet co-simulation, and you want to use Ethernet P2P for (faster) co-simulation, then in addition to the above steps, use an Ethernet cable to connect the Ethernet port of your PC to the Ethernet port of your board. If your computer has more than one Ethernet card/port, note the MAC address of the port that you connected to the board. Specify that address to the ISim engine using the ISim Tcl command: **hwcosim set ethernetInterfaceID**.
5. Power the board on and check to ensure the LED on the cable is green.
6. Install the Xilinx cable drivers when prompted. For more information about Ethernet, see [Determining the Ethernet, page 94](#).

Hardware Co-Simulation

Unlike the executable for tool simulation, the executable for HWCoSim communicates with a hardware board and offloads the simulation of the selected portion of a design into the hardware. It is invoked in the same way as in the tool simulation flow:

- Invoking the executable launches a Tcl shell interface for controlling the simulation.
- Invoking the executable with the `-gui` option launches the GUI front end with waveform display.

Before the simulation starts and each time the simulation is restarted, the executable configures the FPGA with the hardware co-simulation bitstream. The configuration process can take a few seconds or longer, depending on the speed of the JTAG cable. ISim prints a message to the Console panel when the configuration is complete.

ISim Hardware Co-Simulation Tcl Commands

ISim provides the following Tool Command Language (Tcl) commands to access a given property of an instance in the design hierarchy under hardware co-simulation. These commands are scope-sensitive; you must first use the `scope` command to select an instance that is marked **enabled**.

- `hwcosim get <property>`

Get a property of the hardware co-simulation instance in the current scope. For example, the following Tcl commands get the `cableParameters` property of the hardware co-simulation instance under `/mytestbench/top/hwcosim_inst`.

```
scope /mytestbench/top/hwcosim_inst
hwcosim get cableParameters
```

- `hwcosim set <property> <value>`

Set a property of the hardware co-simulation instance in the current scope. The effect of the `hwcosim set` command is only taken after the initialization finishes (after the `init/restart` command) and before the simulation runs (before the `run` command). Also, the effect is not preserved between simulation runs; you might need to call the `-hwcosim set` command each time after calling the `init` or `restart` command. For example, the following Tcl commands set the `skipConfig` property of the hardware co-simulation instance under `/mytestbench/top/hwcosim_inst` in two simulation runs.

```
init
scope /mytestbench/top/hwcosim_inst
hwcosim set skipConfig 1
run 1000 ns
restart
scope /mytestbench/top/hwcosim_inst
hwcosim set skipConfig 1
run 1000 ns
```

The following hardware co-simulation properties can be changed before a simulation runs:

- **skipConfig**
Default is 0. Set to 1 if the FPGA configuration should be skipped. To skip the configuration, the FPGA should have been configured with a valid hardware co-simulation bitstream. Otherwise, unexpected behavior can occur.
- **cableParameters**
Default is an empty string. This property is used to specify a third-party JTAG cable supported by iMPACT or the ChipScope™ debugging analyzer. Refer to the iMPACT help and the *ChipScope Pro Software and Cores User Guide (UG029)* for details on specifying cable plug-in parameters. [Appendix D, Additional Resources](#) has a link to this document.
- **shareCable**
Default is 0. This property is only available for JTAG-based co-simulation interfaces. Set to 1 if the JTAG cable is to be shared with the Xilinx Microprocessor Debugger (XMD) or the ChipScope debugging analyzer for concurrent access. Share the JTAG cable only when necessary as this could decrease the hardware co-simulation performance substantially.

- **ethernetInterfaceID**
Default is an empty string. This property is only available for Ethernet-based co-simulation interfaces.

If you have multiple Ethernet cards available on your host machine, you need to select the Ethernet card that is connected to your FPGA board. You can select an Ethernet card by setting the value of this property to the MAC address (in the format of `xx:xx:xx:xx:xx:xx`) of the Ethernet card. For more information, see [Determining the Ethernet, page 94](#).

Board Support

To support a new FPGA board for hardware co-simulation in ISim, the board must have a JTAG header. Provide a board support file that records the following information of the board:

- FPGA part information
- Period and pin location of the system clock
- JTAG boundary scan chain information

After you enter the board information into a board support file, you can use that board for HwCoSim. There is no GUI option to generate the board support file.

To support additional boards, you can either modify the default board support file or provide your own board support file, named `hwcosim.bsp`, in the directory where `fuse` is invoked. The board support file contains board specifications in a specific format.

In the following example, `m1402-jtag` is the board identifier that is provided to the `fuse` command to compile the design for the given board.

```
'm1402-jtag' => {
  'Description' => 'ML402 (JTAG)',
  'Vendor' => 'Xilinx',
  'Type' => 'jtag',
  'Part' => 'xc4vsx35-10ff668',
  'Clock' => {
    'Period' => 10,
    'Pin' => 'AE14',
  },
  'BoundaryScanPosition' => 3,
},
```

Where the board identifier includes the following list of properties:

- Description is the description of the board.
- Vendor is the the board vendor.
- Type is the type of co-simulation interface to be used. Allowed values are as follows:
 - `jtag`
 - `ppethernet` (for Point-To-Point Ethernet-based HwCoSim).
- Part is the part name of the FPGA on the board.
- Clock is the system clock information, where: `Period` (and `VariablePeriods`) specifies the supported clock period(s) in nanoseconds.

- Pin is the clock pin location. For differential clock sources, provide both Positive and Negative clock pin locations.
- BoundaryScanPosition is the position of the FPGA on the JTAG boundary scan chain, beginning with 1. This information can be determined by running the Xilinx iMPACT tool.

Note: For P2P Ethernet HwCoSim, additional fields must be specified. See the setup for the ml605-ppethernet entry in the `$XILINX/sysgen/hwcosim/data/hwcosim.bsp` file as an example.

Board Support File

Xilinx boards are supported by default. The default board support file is installed under the following directory of an ISE® 14.x installation: `$XILINX/sysgen/hwcosim/data/hwcosim.bsp`.

Boards with default support are as follows:

- **ml401-jtag** - Xilinx® Virtex®-4 ML401 Evaluation Platform
- **ml402-jtag** - Xilinx ML402 Evaluation Platform
- **ml403-jtag** - Xilinx ML403 Evaluation Platform
- **ml405-jtag** - Xilinx ML405 Evaluation Platform
- **ml410-jtag** - Xilinx ML410 Evaluation Platform
- **ml501-jtag** - Xilinx Virtex-5 ML501 Evaluation Platform
- **ml505-jtag** - Xilinx ML505 Evaluation Platform
- **ml506-jtag** - ML506 ppethernet - Xilinx ML506 Evaluation Platform
- **ml507-jtag** - Xilinx ML507 Evaluation Platform
- **xupv5-jtag** - Xilinx University Program XUPV5-LX110T Development System
- **ml510-jtag** - Xilinx ML510 Evaluation Platform
- **ml605-jtag** - ML605 ppethernet Xilinx Virtex-6 ML605 Evaluation Platform
- **kc705-jtag** - Xilinx Kintex™-7 FPGA KC705 Evaluation Kit
- **zc702-jtag** - Zynq™-7000 EPP board
- **vc707-jtag** - Xilinx Virtex-7 FPGA VC707 Evaluation Kit
- **s3e-sk-jtag** - Xilinx Spartan®-3E Starter Kit
- **s3e-mb-jtag** - Xilinx Spartan-3E MicroBlaze Development Kit
- **s3a-sk-jtag** - Xilinx Spartan-3A Starter Kit
- **s3an-sk-jtag** - Xilinx Spartan-3AN Starter Kit
- **s3adsp1800a-jtag** - Xilinx Spartan-3A DSP 1800A Starter Platform
- **s3adsp3400a-jtag** - Xilinx Spartan-3A DSP 3400A Development Platform
- **sp601-jtag** - sp601 ppethernet Xilinx Spartan-6 SP601 Evaluation Platform
- **sp605-jtag** - sp605 ppethernet Xilinx Spartan-6 SP605 Evaluation Platform

Determining the Ethernet

To run an Ethernet-based Hardware Co-Simulation, when multiple Ethernet interfaces are present, you must select the Ethernet interface that you want to co-simulate.

If you ran a previous hardware co-simulation using the Point-to-Point interface option, you see the following error message:

```
"ERROR: In process wrapper AHIL_INITIALIZE Failed to open hardware
co-simulation instance. Error in Point-to-point Ethernet Hardware
Co-simulation. There are multiple Ethernet interfaces available. Please
select an interface."
```

Use the following steps to determine the Ethernet port, set and verify the Ethernet address, and verify that the simulation runs.

1. Determine the Ethernet port to which the co-simulation board is connected.
 - a. On your system command prompt, open a command terminal window (**cmd**).
 - b. In the command window, type **ipconfig -all** to list all Ethernet ports and connections. [Figure 8-5](#) shows the command in the **cmd** window.

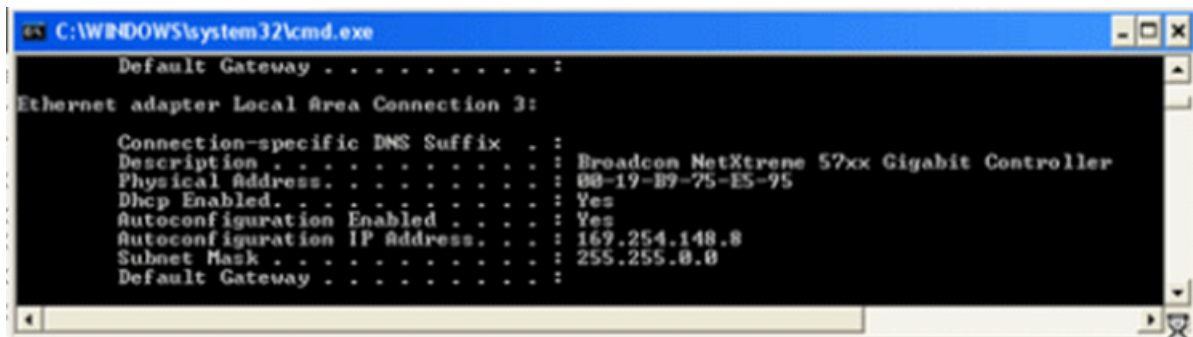


Figure 8-5: **ipconfig -all** in **cmd.exe**

- c. Locate the physical address of the Ethernet port connected to the co-simulation board.
 - d. Convert the physical address delimiter from a dash (-) to a colon (:). For example:
00:19:B9:75:E5:95
2. Set and verify the correct Ethernet port, as follows:
 - a. Open the GUI.
 - b. Select the Design under Test (DUT).
 - c. Go to the Tcl Console panel.
 - d. In the Tcl Console panel, type the following commands:
 - i. Set the Ethernet address:

```
hwcosim set  
ethernetInterfaceID <##:##:##:##:##:##> <physical address>
```
 - ii. Verify the Ethernet address:

```
hwcosim get ethernetInterfaceID
```
 - iii. Verify that the simulation runs:

```
run 10us
```


Figure 8-6 outlines the process within the GUI.

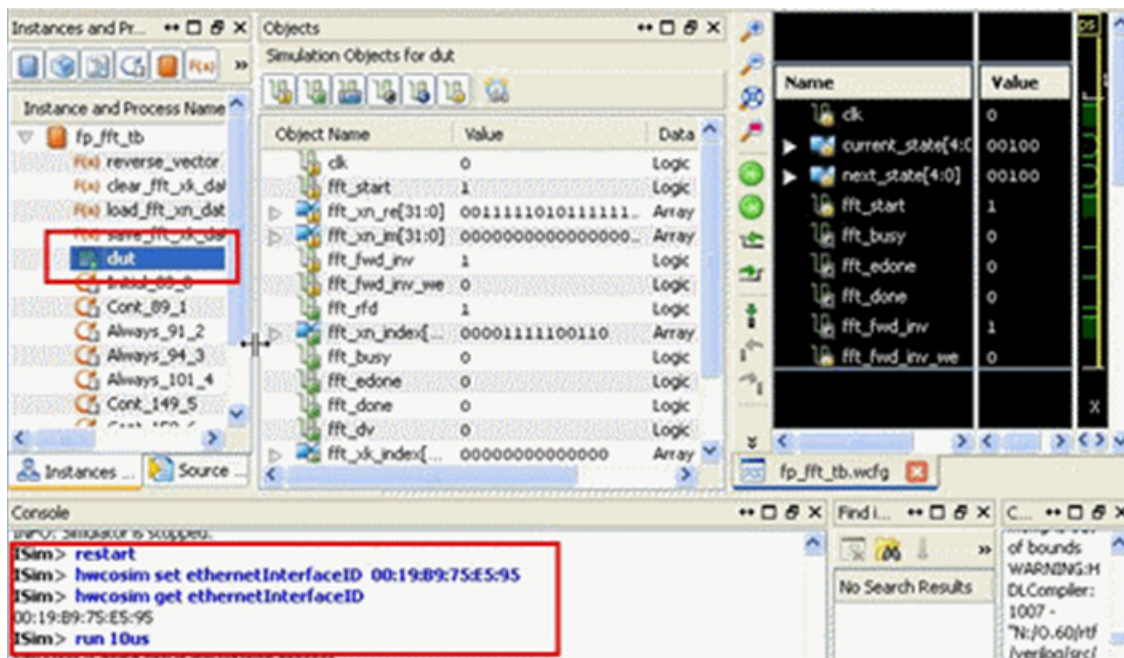


Figure 8-6: HwCoSim Process in GUI Console

Frequently Asked Questions

In the following description, *design under test (DUT)* refers to the portion of design that is co-simulated in FPGA.

General

- Q: Does HwCoSim support any kind of designs?

A: Certain limitations, described in [Limitations, page 84](#), apply.
- Q: Is IHwCoSim a functional simulation or a timing simulation?

A: It is a functional simulation assisted with hardware, which is bit-and-cycle accurate with respect to the pure tool simulation.
- Q: How do I use my own board?

A: You can add your own board in the board support file as described in [Board Support, page 92](#).

Compilation

1. Q: Can I co-simulate multiple instances in the design in hardware?
A: No. Only one instance can be selected for hardware co-simulation. You can co-simulate the parent instance of the multiple instances or group the multiple instances into one instance.
2. Q: What happens if the DUT already instantiates a BSCAN primitive (such as the ChipScope debugging tool ICON)?
A: The hardware co-simulation interface uses a BSCAN primitive at location 1, which could result in an error in MAP if the DUT also instantiates another BSCAN primitive at location 1.

You might need to change the ChipScope ICON to use a different location for the BSCAN primitive. Some device families, such as Spartan-3, have only one BSCAN primitive, where the hardware co-simulation interface cannot coexist with a ChipScope ICON module.

To share the JTAG cable with ChipScope Analyzer or the XMD, run the Tcl command:
hwcosim set shareCable 1.

Simulation

1. Q: Can I skip the FPGA configuration and reuse the last downloaded bitstream for multiple co-simulation runs?
A: There is no command line or GUI option for that; however, you can run this Tcl command: **-hwcosim set skipConfig 1**. Be aware that, by skipping the bitstream configuration, the design running in the FPGA maintains its previous states across simulation runs. You might need to appropriately reset the design in the test bench when a new simulation run starts.
2. Q: Can I probe a signal inside the DUT that is co-simulated in hardware?
A: No. Only the port interface of the DUT can be accessed from ISim through the hardware co-simulation interface. To debug an internal signal, you need to route the signal to a port of the DUT.

Clocking and I/Os

1. Q: Which clock is supplied to the DUT? How is the clocking of the DUT being handled during co-simulation?
A: The clock pin of the hardware board specified in the board support file is the master clock source but is not used directly to drive the DUT. This clock source is scaled to a particular clock frequency (typically around 25 to 100 MHz) through a DCM/PLL. The scaled clock further goes through a gated clock buffer (BUFGCTRL) before driving the clock port of the DUT. The gated clock buffer generates a clock pulse to the DUT per simulation cycle to synchronize the tool simulation and the DUT execution.
2. Q: Can I let the DUT clock be free-running?
A: In hybrid co-simulation, you can let a portion of the DUT be free-running by mapping one or multiple clock ports to external I/Os. However, at least one clock of the DUT must be clocked in a lockstep fashion synchronous to the tool simulation. That effectively partitions the DUT into two portions; one running in lockstep and the other free-running. Also note that ISim hardware co-simulation does not insert any clock domain crossing between the two portions.

Some asynchronous buffer or additional synchronization logic is expected to properly handle clock domain crossing between the two portions.

3. Q: Can I run the DUT at a particular clock frequency?

A: You can supply external clocks for the free-running portion of the DUT in hybrid co-simulation. However, the clock in the lockstep portion is controlled by ISim, and is synchronous to the tool simulation and thus not fixed to a particular clock frequency. The effective clock rate of the lockstep portion is slow regardless of its input clock frequency.

Driving the DUT with a faster clock does not improve the simulation performance as the major bottleneck is the communication between tool and hardware. The DUT is constrained with a lower clock frequency to make the compilation process (such as map and place-and-route) faster.

4. Q: Can I connect some ports of the DUT to external I/Os such as DDR memory modules?

A: External I/Os and clocks are supported in the hybrid co-simulation mode through the use of a custom constraints file as described in [Hybrid Co-Simulation Flow](#), page 88.

ISim Tcl Commands

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

Note: All commands are case sensitive.

Simulation Command Entry Methods

Simulation commands can be entered as follows:

- **ISim GUI**
Type simulation commands in the ISim Console panel.
- **Command Line**
Type simulation commands at the command line Tcl prompt.
- **Tclbatch**
Type simulation commands in a Tcl file, and reference the Tcl file with the `-tclbatch` option of the 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:

1. Click **Edit > Language Templates**.
2. In the Language Templates, expand the **Tcl > Tools > 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](#), page 102.

Note: For information on using Tcl, see documentation at <http://www.tcl.tk/>.

Simulation Command Summary

Table 9-1 lists the available simulation Tcl commands with a brief description, and links to a more complete description of the command, syntax, options, and examples.

In your PDF reader, turn on **Previous View** and **Next View** buttons to navigate back and forth.

Table 9-1: Simulation Command Summary

Command	Description
<code>bp</code>	Sets and deletes breakpoints in your HDL source code for debugging purposes.
<code>describe</code>	Displays information about the given HDL data or block object.
<code>divider add</code>	Adds a new divider.
<code>dump</code>	Displays a list of variables, generics, parameters, and nets along with their values for the current scope of the design hierarchy.
<code>group add</code>	Adds a new group.
<code>help</code>	Displays a description with usage and syntax of the specified ISim command.
<code>isim condition</code>	Enables the execution of a set of commands based on a specified condition on one or more nets or Verilog regs.
<code>isim force</code>	Forces or removes a value on a VHDL signal, Verilog wire, or Verilog reg.
<code>isim set arraydisplaylength</code>	Displays the limit of numbers of elements for an array type HDL object.
<code>isim get radix</code>	Gets the global radix being used.
<code>isim get userunit</code>	Displays the current unit of measurement for all time values where unit is unspecified.
<code>isim ltrace</code>	Turns on and off line tracing.
<code>isim ptrace</code>	Turns on and off tracing of execution of processes.
<code>isim set arraydisplaylength</code>	Sets the limit on the number of elements for an array type HDL object.
<code>isim set radix</code>	Sets the global radix to use.
<code>isim set userunit</code>	Sets the default unit of measurement for all time values where unit is unspecified.
<code>marker add</code>	Adds a new marker.
<code>onerror</code>	For batch mode, controls the behavior immediately following a failed Tcl simulation command.
<code>put</code>	Assigns a value to a specified bit, slice, or all of a variable or signal.
<code>quit</code>	Exits either the simulation or the tool, depending on the command options.
<code>restart</code>	Restarts simulation, setting the simulation time back to 0.

Table 9-1: Simulation Command Summary (Cont'd)

Command	Description
<code>resume</code>	With the <code>onerror</code> command, continues executing commands after an error is encountered.
<code>run</code>	Starts simulation.
<code>saif</code>	Creates a Switching Activity Interchange Format (SAIF) file and records estimated power usage.
<code>scope</code>	Navigates the design hierarchy.
<code>sdfanno</code>	Back-annotates delays from an SDF file to the HDL design.
<code>show</code>	Displays selected aspects of the design in the Simulation Console panel.
<code>step</code>	Executes simulation through your HDL design, line by line, to assist with debug.
<code>test</code>	Compares the actual value of a net or bus with a supplied value.
<code>vcd</code>	Generates simulation results in VCD format.
<code>virtualbus add</code>	Adds a new virtual bus.
<code>wave add</code>	Adds simulation objects or blocks to the specified wave configuration in the ISim graphical user interface.
<code>wave log</code>	Logs simulation output of HDL objects to a waveform database.
<code>wcfg new</code>	Creates a new wave configuration.
<code>wcfg open</code>	Opens a wave configuration from a file.
<code>wcfg save</code>	Saves a wave configuration.
<code>wcfg select</code>	Selects the wave configuration file to be displayed in the active window.

Using Object Names with Special Characters in Tcl

Some commands, such as `wave add`, can take arguments that contain characters that have special meaning to Tcl. Those arguments must be surrounded with curly braces `{}` to avoid unintended processing by Tcl. The most common cases are as follows.

Bus Indexes

Because square brackets `[]` have special meaning to Tcl, an indexed (bit- or part-selected) bus using square bracket notation must be surrounded with curly braces. For example, when adding element 4 of `bus` to the Wave window using square bracket notation, you must write the command as `wave add {bus[4]}`.

Parentheses can also be used for indexing a bus, and because parentheses have no special meaning to Tcl, the command can be written without curly braces. For example:

```
wave add bus(4)
```

Verilog Escaped Identifiers

Verilog identifiers containing characters that are special to Verilog need to be “escaped” both in Verilog source code and on the ISim command line by prefixing the identifier with a backslash `\` and appending a space.

Additionally, on the Tcl command line the escaped identifier must be surrounded with curly braces. For example, to add wire `my wire` to the Wave window, you must write the command as `wave add {\my wire}`, taking care to supply a space between the final character and the closing curly brace.

Note: Verilog allows any identifier to be escaped. On the ISim command line do not escape Verilog identifiers that are not required to be escaped. For example, to add wire `w` to the Wave window, ISim would not accept `wave add {\w}` as a valid command. Instead, the command must be written as `wave add w`, or optionally, `wave add {w}`.

If the escaped identifier contains a curly brace, then the technique of surrounding the identifier with curly braces does not work, because Tcl interprets curly braces as special characters even within curly braces. Instead, you must use the technique demonstrated in VHDL Extended Identifiers, below.

VHDL Extended Identifiers:

VHDL extended identifiers contain backslashes `\`, which are special characters to Tcl. Because Tcl interprets a backslash next to a close curly brace `\}` as being a close curly brace character, VHDL extended identifiers cannot be written with curly braces.

Instead, the curly braces must be absent and each special character to Tcl must be prefixed with a backslash. For example, to add the signal `\my sig\` to the Wave window, you must write the command as `wave add \\my\ sig\\`.

Both the backslashes that are part of the extended identifier, and the space inside the identifier are prefixed with a backslash.

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*.

Setting a Variable at the Simulation Console Panel

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

To then run the variable (and thereby execute the simulation command), type:

```
eval $svc.
```


ISim Wave Viewer Tcl Commands Overview

The ISim Wave Viewer Tcl commands operate on the active window. When ISim starts, the first active window is `Default.wcfg`.

You can change the active window by:

- Clicking the window tab or using the `wcfg select` command.
- In the GUI, select **File > New** and **File > Open** to change the active window to the newly created waveform configuration window.
- In Tcl, you can use the `wcfg new` and `wcfg open` commands to change the active window to the newly created window.

Command Line Conventions

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

Table 9-2 provides a summary of the syntax used for the simulation commands.

Table 9-2: Simulation Command Syntax

Syntax	Description
[]	Indicates an optional command option. Note: The [] can be used in Tcl for nesting commands. A command put inside [] is executed and result of that is returned as a value to be used in another Tcl command. For example, <code>set var <show time></code> sets <code>var</code> to the current time.
	Indicates a choice of possible parameters. If one term is divided into a subset of parameters that can be entered separately or together, each sub-parameter occurs between square brackets.
...	Indicates that one or more occurrences of the option separated by space is accepted.
< >	Encloses variables for which you must supply values.

Tcl Commands

Engine Commands

bp

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.

bp Command Syntax

```
bp add <file_name> <line_number> clear
del <index> [<index>... ]list
remove <file_name> <line_number>
```

bp Command Options

Table 9-3: bp Command Options

Option	Description
add <file_name> <line_number>	Adds a breakpoint at the given line in the HDL file. The <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 loaded HDL files. 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 <code>bp list</code> command to obtain the index numbers for your breakpoints. See the <code>list</code> 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 <i>not</i> the line number of the breakpoint in the source file.

Table 9-3: bp Command Options (Cont'd)

Option	Description
list	<p>Lists all of the breakpoints in your design and shows their index number to be used with the <code>bp del</code> command. If you have set breakpoints in multiple files, all breakpoints are listed. <code>bp list</code> returns the following:</p> <pre><index> <directory_path> <file_name>/ <line_number></pre> <p>Where:</p> <ul style="list-style-type: none"> • <code><index></code> is the index number to use with the <code>bp del</code> command. • <code><directory_path></code> is the fully qualified path to your source files. • <code><file_name></code> is the name of the source file in which there is a breakpoint. • <code><line_number></code> is the line number in the source file of the breakpoint.
remove <file_name> <line_number>	<p>Deletes a breakpoint at the <code><line_number></code> in the <code><file_name></code>.</p> <ul style="list-style-type: none"> • The <code><file_name></code> is the name of the HDL source file that you are simulating and where you want to remove a breakpoint. • The <code><line_number></code> is the number of the line of HDL code where the breakpoint has been set.

bp Command Examples

Set a breakpoint at line 2 for the file `statmach.vhd`:

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

List all Breakpoints

List all breakpoints in all files in your simulation, and give them a unique index number:

```
bp list
```

Delete all breakpoints in your simulation:

```
bp clear
```

To delete 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
```

Delete a breakpoint at line 2 in `statmach.vhd`:

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

describe

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

describe Command Syntax

```
describe <object_name>
```

The `<object_name>` option displays a description about either an HDL object or an HDL block in the current simulation scope.

describe Command Example

The `describe` command can be used as follows.

```
describe param
```

Returns:

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

dump

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 the `isim set radix` command.

dump Command Syntax

```
dump
```

dump Command Example

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

```
dump
```

help

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 ISim Tcl commands.

Help Command Syntax

```
help [command_name]
```

help Command Options

`help` displays a description for the specified command.

help Command Examples

The `help` command can be used as follows.

For a description of all of ISim commands:

help

For a description of the bp command:

help bp

isim condition

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` block. When added, it starts monitoring signals (those that appear in the `isim condition` expression) continually during simulation. The condition expression is evaluated anytime a signal change is detected. When a specified condition expression is met, the specified command runs.

- `isim condition remove` stops monitoring signals.
- `isim condition list` lists all active conditional actions added and their labels and IDs.

isim condition Command Syntax

```
isim condition add|remove|list
[<condition expression> <command>]
[-radix <radix_type>]
[-label <label_name>]
[-index <index_name>]
-all
```

isim condition Command Options

Table 9-4: `isim condition` Command Options

Option	Description
add remove list	Specifies to add a condition, remove one or more conditions, or list all active conditions.
<condition expression> <command>	<p>The <condition expression> is associated with the add function, and it determines when to run the specified <command>.</p> <p>Operators used in the condition expression include != (not equal), == (equal), && (and), and (or).</p> <p>A space is required between words in the expression and the operator, such as <code>clk == St1</code>.</p> <p>The <command> is a Tcl command or script that is executed when the condition expression is true. This command is surrounded by {} (braces). The command can include standard Tcl commands and simulation Tcl commands, except <code>run</code>, <code>restart</code>, <code>init</code>, and <code>step</code>. Tcl variables used in the condition expression are surrounded by quotes "" instead of {}.</p> <p>Refer to the syntax examples below.</p>

Table 9-4: isim condition Command Options (Cont'd)

Option	Description
-radix <radix_type>	An optional argument used to read the value in the condition expression. The supported radix types are <code>default</code> , <code>dec</code> , <code>bin</code> , <code>oct</code> , <code>hex</code> , <code>unsigned</code> , and <code>ascii</code> . When no radix type is specified, the global radix type set with the <code>isim set radix</code> command is used, or if none is set there, <code>default</code> is used as radix.
-label <label_name>	An optional argument that specifies the name of a condition. For <code>isim condition add</code> , 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 <code>isim condition remove</code> only.
-all	An optional argument that is used to remove all conditions in the current simulation.

isim condition Command Examples

isim condition add Examples

Adding 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
```

Adding 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
```

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

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

Adding 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
```

Adding 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

Remove all conditions in the current simulation:

```
isim condition remove
```

or

```
isim condition remove -all
```

Remove the conditions `label0`, `label1`, and `label2`:

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

Remove a single statement:

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

or

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

ISim condition list Examples

Listing all ISim conditions added to the design at the current time.

```
isim condition list
```

```
isim force
```

The `isim force` command forces a VHDL signal, Verilog wire, or Verilog reg to a constant value or a repeating pattern over time. The value applied by the `isim force` command overrides any value assigned from within the HDL code, or any value applied by a previous force. The force remains active until cancel time, if a cancel time is specified, or until an explicit `isim force remove` command is issued.

- For a VHDL signal or a Verilog wire, removal of a force restores the value of the signal or the wire to the current driven value.
- For a Verilog reg, the forced value is retained even after the applied force has been removed until the time one of the HDL processes that write into the Verilog reg gets to assign a new value to the reg.

isim force Command Syntax

```
isim force add|remove
[<object_name>]
[-value <value>]
[-radix <radix_type>]
[-time <time>]
[-cancel <time>]
[-repeat <time>]
```

isim force Command Options

Table 9-5: **sim force Command Options**

Option	Description
add remove	Specifies whether you wish to assign or remove a value from a bus or signal.
<object_name>	Specifies name of the VHDL signal, Verilog wire, or Verilog reg to be forced.
-value <value>	Specifies one or more values to add.
-radix <radix_type>	This option specifies the radix. 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 <code>isim set radix</code> command is used, or if none is set, default is used.

Table 9-5: **sim force Command Options (Cont'd)**

Option	Description
-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>	Cancels the force command after the specified time.
-repeat <time>	Repeats the cycle after the specified time.

isim force Examples

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

Assigning a Value

Force signal `rst` to 0 starting at the current simulation time:

```
isim force rst 0
```

Force signal `rst` to 1 starting at 10 ns from the current simulation time and cancel forcing after 50 ns from the current simulation time:

```
isim force rst 1 -time 10 ns -cancel 50 ns
```

Apply a clock to the signal `clk` such that `clk` goes to 1 at current simulation time, goes back to 0 at 20 ns later, and then repeats this every 40 ns until 1 us from the current simulation time (for example, to generate a clock with 50% duty cycle and 40 ns period for a duration of 1 us):

```
isim force clk 1 -value 0 -time 20 ns -repeat 40 ns -cancel 1 us
```

To:

- Force signal `data_in` to 1 at current simulation time
- Set `data_in` to 0 at current simulation time + 50 ns
- Set `data_in` back to 1 at current simulation time + 75 ns
- Repeat this 101 pattern every 100 ns for a duration of 5000 ns:

```
force add data_in 1 -value 0 -time 50 ns -value 1 -time 75 ns -repeat 100 ns -cancel 5000 ns
```

Removing a Value

Remove the values on signals `s`, `s1`, and `s2`:

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

```
isim get arraydisplaylength
```

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

isim get arraydisplaylength Command Syntax

```
isim get arraydisplaylength
```

where *no* options are available.

isim get arraydisplaylength Command Example

Type the following command to determine the array display length, and the array is 64-bit:

```
isim get arraydisplaylength
```

Returns: 64

```
isim get radix
```

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

isim get radix Command Syntax

```
isim get radix
```

No options are available.

isim get radix Command Example

Return the radix:

```
isim get radix
```

Returns the current global radix.

```
isim get userunit
```

The `isim get userunit` command displays 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.

isim get userunit Command Syntax

```
isim get userunit (where no options are available)
```

isim get userunit Example

```
isim get userunit
```

Returns: 1 ps

```
isim ltrace
```

The `isim ltrace` command lets you turn line tracing on or off. When line tracing is turned on, you can do a line-by-line analysis for debugging. Executed HDL lines print to the screen with the following information: simulation time, filepath, filename, and line number.

Note: `isim ltrace on` can slow the simulation.

isim ltrace Command Syntax

```
isim ltrace [on | off]
```

isim ltrace Command Options

The `isim ltrace` command options are `[on | off]`. The default is off.

isim ltrace Command Example

To see which line is currently executing:

```

    isim ltrace on
    run

```

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

```

1005 ns "C:/Data/ISE_Projects/freqm/watchver/
stopwatch_tb.v":261005 ns "C:/Data/ISE_Projects/freqm/watchver/
stopwatch_tb.v":271005 ns(3) "C:/Data/ISE_Projects/freqm/watchver/
statmach.v":631005 ns(3) "C:/Data/ISE_Projects/freqm/watchver/
statmach.v":64

```

```
isim ptrace
```

The `isim ptrace` command lets you 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 panel. This is 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.

isim ptrace Command Syntax

```
isim ptrace [on | off]
```

isim ptrace Command Options

The options are `[on | off]`. The default is `off`.

isim ptrace Command Example

See which process is currently executing.

```
isim ptrace on
```

```
isim set arraydisplaylength
```

The `isim set arraydisplaylength` command sets the limit on the number of elements for an array type HDL object. The default is 64. The following are affected by the limit set:

- The display of values in the GUI Objects
- The response to the `show value` command

isim set arraydisplaylength Syntax

```
isim set arraydisplaylength <size>
```

The available option is `<size>`. Type a number of elements of an array type HDL object to display. Use 0 for unlimited length. The default is 64.

isim set arraydisplaylength Command Examples

Enter:

```

isim set arraydisplaylength 2
show value xcountout

```

Returns:

```

00
00

```

Examine the **Value** column for arrays in the Objects :

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

Returns:

```
0001000000
0001000000
```

```
isim set radix
```

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`.

isim set radix Syntax

```
isim set radix <radix_type>
```

Where:

The `<radix_type>` option sets the global radix for the current simulation. The simulator uses `<radix_type>` 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`.

isim set radix Examples

Set a radix of hex, and see a count value:

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

Returns: a.

Set a radix of dec, and see a count value:

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

Returns -4.

Set a radix of unsigned, and see a count value.

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

Returns: 10.

```
isim set userunit
```

The `isim set userunit` command let you set the default unit of measurement for all time values where unit is unspecified. The default time unit is the same as the time-unit of the engine 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 1ps.
- As defined by the ``timescale` simulator directive in Verilog.

isim set userunit Command Syntax

```
isim set userunit <1|10|100> <fs|ps|ns|us|ms|s>
```

Example

Set the simulator timescale to 1 ps:

```
isim set userunit 1 ps
```

onerror

The `onerror` command lets you 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.

Note: This command is not intended for users who type one Tcl command at a time at the Tcl prompt.

onerror Command Syntax

```
onerror [<list_of_Tcl_commands>] [<source Tcl_script>]
```

Where:

- `<list_of_Tcl_commands>` Is a list of simulation Tcl commands to control behavior upon encountering a simulation command error.
- `<source Tcl_script>` Is the sourced Tcl script file that contains listed Tcl commands.

onerror Command Examples

Print the that an error occurred and all values in the current scope, then exit:

```
onerror {showtime;dump;quit -f}
```

Continue 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}
```

Read the sourced Tcl file and executes its command upon encountering an error:

```
onerror {source myerror.tcl}
```

put

The `put` command lets you 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 or buses
- A record or array of records containing signals or 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 Hardware Description Language (HDL) source code.

You cannot use the `put` command to assign a value to a VHDL variable, a VHDL generic, or a 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; consequently, the command is temporary.

put Command Syntax

```
put <signal_name>|<vhdl_process_name>| <process_variable_name>
[element reference...] <value> [<object> <value>] [-radix
<radix_type>]
```

put Command Options

Table 9-6: put Command Options

Option	Description
<pre><signal_name> <vhdl_process_name> <process_variable_name> [element reference...] <value></pre>	<ul style="list-style-type: none"> • <signal name> is 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 or 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 a slash symbol, /. • The optional <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. You can use it to further refine the signals by referencing the individual sub-elements of a signal. See the following examples for more information. <p>The accepted <value> depends upon the signal type as follows:</p> <ul style="list-style-type: none"> • 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,... • For Verilog, <bit_values> can be U, X, 0, 1. • Strength values are not supported.
<pre>[<object> <value>] [-radix <radix_type>]</pre>	<p>Takes a value with the specified radix, converts the value to the data type of the object, and writes the value to the object.</p> <p><object> specifies the signals, buses or objects to modify. <value> refers to the value you wish to add to the object.</p> <p>Supported radix types: -radix [radix_types]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.</p>

put Command Examples

Assign a Value To a Bus or Signal

Assign a value to a signal called `clk`:

```
put clk 1
```

or

```
put clk 1 -radix bin
```

or

```
put clk "1" -radix bin
```

Assign a value to a 4-bit bus called `busx`:

```
put busx 0101
```

Assign a value FF to signal `A`:

```
put A FF -radix hex
```

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);
```

Set bit 0 of `sigx` to 1:

```
put sigx(0) 1
```

Set slice 1 to 2 of `sigx` to 11:

```
put sigx(1:2) 11
```

Set all of `sigx` to 101010:

```
put sigx 101010
```

Assign a Value To an Array of Objects

To assign a value to 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);
```

Set every bit of the array element of `sigarray` to 1:

```
put sigarray(0,1,2) 1111
```

Set the first two bits of the array element of `sigarray` to 10:

```
put sigarray(0,1,2) (1:2) 10
```

Set bits three of the array element of `sigarray` to 1:

```
put sigarray(0,1,2) (3) 1
```

Assign a value to 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;
```

Set the second element (b) of the record recsig to the string abc:

```
put recsig.b(2:4)abc
```

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
```

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
```

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
```

The quit command exits either the simulation or the tool, depending on the command options. With no options, the quit command closes the ISim tool after being prompted to do so (for graphic user interface (GUI) mode) or simply closes ISim (in command-line mode).

quit Command Syntax

```
quit [-f] [-s]
```

Where:

- -f
Quits the current simulation and quits the ISim tool. 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 using **File > Open** to open a WDB file. The switch has no effect at the command line.

quit Command Examples

Exit ISim and leave the Tcl prompt:

```
quit
```

Exit ISim and save your waveform:

```
quit -f
```

Quit the current simulation:

```
quit -s
```

restart

The `restart` command stops simulation and sets simulation time back to 0. This lets you start simulation over again within the same simulation run without reloading the design. The equivalent GUI command is **Simulation > Restart**.

restart Command Syntax

```
restart [onerror] [scope] [saif] [vcd] [isim force] [put]
```

restart Command Options

Table 9-7: restart Command Options

Option	Description
<code>onerror</code>	Removes the <code>onerror</code> script.
<code>scope</code>	Resets scope to <code>/top</code> .
<code>saif</code>	Closes file.
<code>vcd</code>	Closes file.
<code>isim force</code>	Removes force.
<code>put</code>	Returns to initial value.

restart Command Example

To set simulation time back to 0, and start simulation:

```
restart
```

resume

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.

resume Command Syntax

```
resume
```

There no available options to this command.

resume Command Example

To continue 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

The `run` command starts simulation. With no options, the `run` command runs simulation for 100 ns. The equivalent GUI commands are:

- **Simulation > Run All**
- **Simulation > Run**

run Command Syntax

```
run [all] [continue] [<time> <unit>]
```

run Command Options

Table 9-8: run Command Options

Option	Description
all	Runs simulation until there are no more events or until ISim encounters a breakpoint. See <code>bp</code> 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 <code>run all</code> .
<time> <unit>	<ul style="list-style-type: none"> • <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 <code>fs</code>, <code>ps</code>, <code>ns</code>, <code>us</code>, <code>ms</code>, and <code>sec</code>. Default is <code>ps</code>.

run Command Examples

The `run` command can be used as follows.

Run simulation until there are no more events or until ISim reaches a breakpoint:

```
run all
```

Run simulation for 2000 nanoseconds:

```
run 2000 ns
```

Run simulation for 1.2 nanoseconds:

```
run 1.2 ns
```

Run simulation for 100 ns:

```
run
```

saif

The `saif` command lets you create a Switching Activity Interchange format (SAIF) file and record port and signal switching rates. See [Chapter 7, Writing Activity Data for Power Consumption](#).

saif Command Syntax

```
saif open [-scope <path_name>] [-file <file_name>] [-allnets]
close [-level <number_of_levels>]
```

saif Command Options

Table 9-9: **saif Command Options**

Option	Description
<code>open</code> <code>[-scope <path_name>]</code> <code>[-file <file_name>]</code> <code>[-allnets]</code>	<p><code>open</code> creates a file for SAIF power estimation.</p> <ul style="list-style-type: none"> <code>-scope <path_name></code> creates power estimation data starting at specified scope and recursively. You can use a relative or an absolute path. If no path is specified, the current scope is used. <code>-file <file_name></code> creates a new SAIF file. The default name of the SAIF file is <code>xpower.saif</code>. Only one SAIF file can be opened during one run of simulation. <code>-allnets</code> includes internal nets and port signals in the power estimation. Without this switch, only port signal changes are monitored.
<code>close</code>	<p>Stops monitoring and flushes the SAIF file.</p>
<code>[-level <number_of_levels>]</code>	<p>You can have as many levels as you have in your design hierarchy, as follows:</p> <ul style="list-style-type: none"> <code>-level 0</code> tracks and dumps all levels of signal transitions below the specified hierarchy into the SAIF file. <code>-level 1</code> tracks and dumps only the signal transitions in the specified hierarchy into the SAIF file. <code>-level 2</code> tracks and dumps two levels of signal transitions in the specified hierarchy into the SAIF file, and so forth.

saif Command Examples

The `saif` command can be used as follows.

Write all ports of a design starting at the current scope recursively to a file called `xpower.saif`:

```
saif open
```

Write the ports and internal nets of a design starting at the current scope recursively to a file called `xpower.saif`:

```
saif open -allnets
```

Write ports and internal nets of a design starting at the UUT recursively to a file called `uut_backward.saif`:

```
saif open -scope uut -file uut_backward.saif -allnets
```

scope

The `scope` command lets you navigate the design hierarchy. With no options, the `scope` command displays the current module information.

scope Command Syntax

```
scope.. <path_name>
```

Where:

- `..`
Moves the current scope to the hierarchy directly above the current one.
- `<path_name>`
is the path to the module for which you want to display the module information. You can use a relative or an absolute path.

scope Command Examples

The `scope` command can be used as follows.

Move up one level in the design hierarchy:

```
scope..
```

Move to the module UUT that is instantiated in the current module:

```
scope UUT
```

Use the `scope` command on child instances in a post route netlist:

```
X_IPAD \CLK/PAD (  
  .PAD(CLK)  
);
```

Where, `\CLK/PAD` is an extended identifier:

```
scope /testbench/UUT/\CLK/PAD\
```

Notice that you must constrain the extended identifiers `CLK` and `PAD` with a backslash at the beginning and at the end.

sdfanno

The `sdfanno` command back-annotates VITAL delays from a Verilog Standard Delay Format (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.

sdfanno Command Syntax

```
sdfanno -min | -typ | -max | <file_name>  
[<file_name>] [-nowarn] [-noerror] [-root <root_path>]
```

Where:

`-min | -typ | -max` and `<file_name>` are required. The [options] are any options listed in [Table 9-10, page 122](#).

sdfanno Command Options

Table 9-10: sdfanno Command Options

Option	Description
<code>-min</code> <code>-typ</code> <code>-max</code>	<p>A delay switch type (<code>-min</code>, <code>-typ</code> or <code>-max</code>) must be specified in the <code>sdfanno</code> command.</p> <ul style="list-style-type: none"> <code>-min</code> - Annotates VHDL/Verilog file with fastest possible delay values, and gives you a Hold Time timing simulation. <code>-typ</code> - Annotates VHDL/Verilog file with typical delay values. <code>-max</code> - Annotates VHDL/Verilog file with longest possible delay values, and gives you a Setup Time timing simulation.
<code><file_name></code>	Name of the SDF file that contains the delay information. A filename must be specified in the <code>sdfanno</code> command.
<code>-nowarn</code>	Turns off warning messages.
<code>-noerror</code>	Turns error messages into warning messages. This lets you continue simulation even though there are errors in SDF back-annotation.
<code>-root <root_path></code>	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 <code><root_path></code> . By default, the root is at the top level of the design.

sdfanno Command Examples

sdfanno -typ Examples

Annotate the submodule called `subdesign` with the typical delay values from `mysubdesign.sdf`:

```
sdfanno -typ mysubdesign.sdf -root /subdesign
```

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

Annotate the submodule “`subdesign`” with the minimum delay values from `mysubdesign.sdf`:

```
sdfanno -min mysydesign.sdf -root /subdesign
```

sdfanno -max Example

Annotate the current top-level design with the maximum delay values found in `design.sdf` and ignore warnings:

```
sdfanno -max design.sdf -nowarn
```

show

The show command displays selected aspects of the design.

show Command Syntax

```
show [child | child -r] [constant] [driver <signal_name>]
[load <signal_name>] [port] [scope] [signal] [time]
value [<generic_name> | <parameter_name> |
<process_name>/<process_variable_name>] [<signal_name>
[element references...]<object> [-radix <radix_type>] variable
```

show Command Options

Table 9-11: show Command Options

Option	Description
child child -r	The child displays the children blocks (one level only) from the current block. The child -r option recursively lists all children blocks, including the child processes from the current block.
constant	Lists constants, generics, and parameters within the current block.
driver <signal_name>	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 <signal_name>	Displays all of the load for the signal specified by <signal_name>.
port	Displays 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 view, but not alter, the hierarchy location. This command is identical to the scope command without the <path_name> option.
signal	Displays signals within the current module including port signals.
time	Displays the simulator current time.
value [<generic_name> <parameter_name> <process_name>/<process_variable_name>]	<ul style="list-style-type: none"> • <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 a forward slash/.

Table 9-11: show Command Options (*Cont'd*)

Option	Description
<pre>[<signal_name> [element references...]</pre>	<ul style="list-style-type: none"> • <signal_name> is the name of the signal or bus to be queried. This can also be the name of an array of signals or buses, an array of records containing signals and buses, or a record of arrays of buses/signals. • Use a dot (.) to separate the record hierarchy, as in: <pre>show value recsig.c</pre> • Use a forward slash / to delimit the hierarchical name of the signal to query a value to a signal on a lower level of hierarchy, as in: <pre>show value mymod/mysig [element reference]</pre> refers to the different ways in which the sub-elements of the signal name can be referenced when used with the show command. This allows further refinement to the signals by referencing the individual sub-elements of a signal. See the following examples: <ul style="list-style-type: none"> • Two integers separated by a colon and enclosed in parenthesis displays a range of values in the vector specified by <signal_name>. For example: <pre>show value (3:0)</pre> • One or more integers separated by commas and enclosed by parenthesis displays the values of elements in a multidimensional array specified by <signal_name>. For example: <pre>show value (2,3)</pre>
<pre><object> [-radix <radix_type>]</pre>	<ul style="list-style-type: none"> • <object> [-radix <radix_type>] displays values with the specified radix. For <object>, set the HDL object data type. • Supported <radix_type> 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 with that command, default is used
<pre>variable</pre>	<p>Displays all of the variables within the current block. To see variables within a VHDL process, use the scope command to navigate to the VHDL process and then run show variable.</p>

show Command Examples

show child

If you are located at the top level of the hierarchy in a design called `fifo_controller` and type `show child`, the following hierarchy information displays:

```
Block Name: <fifo_controller>
```

Type `show child -r` for the same level of the design, and 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 type `show driver fifocount`, the following information displays for the signal `fifocount`:

```
<Driver for fifocount>
fifoclr_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 type `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 type `show scope`, the following information displays:

```
<Block> /tb_cc_func/
```

show value (Signal)

Query a value to a signal called `clk`:

```
show value clk
```

Query a value to a 4-bit bus called `busx`:

```
show value busx
```

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 value (Object)

The following examples refer to a standard logic vector called `sigx` that is declared as follows:

```
signal sigx : std_logic_vector(0 to 5);
```

Query bit 0 of `sig`:

```
show value sigx(0)
```

Query slice 1 to 2 of `sigx`:

```
show value sigx(1:2)
```

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);
```

Query every bit of a vector array element of sigarray:

```
show value sigarray(0,1,2)
```

Query the first two bits of each array element of sigarray:

```
show value sigarray(0,1,2)(1:2)
```

Query bits three of each array element of sigarray:

```
show value sigarray(0,1,2)(3)
```

An array of records which in turn contains an array of standard logic vectors 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;
```

Query the second element b of the record recsig:

```
show value recsig.b(2:4)
```

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)
```

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)
```

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
```

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.

step

The `step` command advances to the next line of executable code in the Verilog or VHDL file. The equivalent GUI command is **Simulation > Step**.

step Command Syntax

```
step
```

where no options are available.

step Command Example

Step through one line of HDL source code:

```
step
```

test

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.

test Command Syntax

```
test <signal_name>| <vhdl_process_name /<process_variable_name>]  
[element reference...]<value>  
<object> <value> [-radix <radix_type>]
```

test Command Options

Table 9-12: test Command Options

Option	Description
<pre><signal_name>/ <vhdl_process_name / <process_variable_name> [element reference...] <value></pre>	<ul style="list-style-type: none"> • <code><signal_name></code> is the name of the signal or bus to be compared. This can also be the name of an array of signals or buses, an array of records containing signals and buses or a record of arrays of buses/signals. • <code><vhdl_process_name/ process_variable_name></code> 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 a forward slash <code>/</code>. • <code>element reference</code> are 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. • <code><value></code> is the supplied value to compare with actual value on net or bus.
<pre><object> <value> [-radix <radix_type>]</pre>	<p>Compares the specified value with the specified radix to object value.</p> <ul style="list-style-type: none"> • <code><object></code> specifies the signals, buses, or objects to test. • <code><value></code> is the value to add to the object. The supported radix types are: <code>default</code>, <code>dec</code>, <code>bin</code>, <code>oct</code>, <code>hex</code>, <code>unsigned</code>, and <code>ascii</code>. When no radix type is specified, the global radix type set with the <code>isim set radix</code> command is used, or if none is set, <code>default</code> is used.

test Command Examples

The test command can be used as follows.

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
```

Compare the value of signal `A` to `FF`:

```
test A FF -radix hex
```

Compare value with `clk` value:

```
test clk "U"
```

Returns: 1

Compare value with `clk` value:

```
test clk "0"
```

Returns: 0

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` that is declared as follows:

```
signal sigx: std_logic_vector(0 to 5);
```

Compare bit 0 of `sigx` to 1:

```
test sigx(0) 1
```

Compare slice 1 to 2 of `sigx` to 11:

```
test sigx(1:2) 11
```

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);
```

Compare every bit of a vector array element of `sigarray` to 1:

```
test sigarray(0,1,2)1111
```

Compare the first two bits of each array element of `sigarray` to 10:

```
test sigarray(0,1,2)(1:2)10
```

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;
```

Compare the second element (b) of the record `recsig` to the string `abc`:

```
test recsig.b(2:4)abc
```

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
```

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
```

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

The `vcd` command generates simulation results in Value Change Dump (VCD) format. This command lets you:

- Dump specified instances to a VCD file
- Name the VCD file
- Start and stop the dump process

and other functions. See [Chapter 7, Writing Activity Data for Power Consumption](#).

vcd Command Syntax

```
vcd [dumpfile <file_name>] [dumpvars -m <module_name> [-l <level>]]
[dumpoff] [dumpon] [dumpall] [dumplimit <file_size>] [dumpflush]
```

vcd Command Options

Table 9-13: vcd Command Options

Option	Description
<code>dumpfile <file_name></code>	Gives a name to VCD file. The default file name is <code>dump.vcd</code> . Invokes the Verilog <code>\$dumpfile</code> directive.
<code>dumpvars -m <module_name> [-l <level>]</code>	Dumps the specified variables and their values to the VCD file. <code>-m <module_name></code> dumps the module of that name. <code>-l <level> {0 1}</code> <ul style="list-style-type: none"> • 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 not to individual variables. • 1 dumps all variable within the module specified by <code>-m</code>; it does not dump variables in any of the modules instantiated by the module specified by <code>-m</code>. Invokes the Verilog <code>\$dumpfile</code> directive.
<code>dumpoff</code>	Temporarily suspends the dumping process, and dumps all selected variables as an X value. Invokes the Verilog <code>\$dumpoff</code> directive.

Table 9-13: vcd Command Options (*Cont'd*)

Option	Description
<code>dumpon</code>	Resumes the dumping process after the <code>dumpoff</code> option is invoked, and dumps all selected values at the time <code>dumpon</code> is invoked. Invokes the Verilog <code>\$dumpon</code> directive.
<code>dumpall</code>	Creates a checkpoint in the VCD file that dumps the current value of all selected variables. Invokes the Verilog <code>\$dumpall</code> directive.
<code>dumplimit <file_size></code>	Limits the size of the VCD file. <code><file_size></code> 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 <code>\$dumplimit</code> directive.
<code>dumpflush</code>	Empties the operating system 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 <code>\$dumpflush</code> directive.

vcd Command 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 log
```

The `wave log` command logs simulation output of HDL object(s) to a waveform database (wdb) file. VHDL signal, Verilog wire, and Verilog reg type HDL objects can be logged. Logging of VHDL variables is not supported.

wave log Command Syntax

```
wave log [-r] [<object_name>]
```

Where:

- `-r`
Recursively adds all child modules of the specified block
- `<object_name>`
HDL object whose simulation output is to be logged to the waveform database. The `<object_name>` can also be a hierarchical instance name

(for example: `/tb/UUT`) of a block in which case all HDL objects within the given block are logged.

Wild characters such as `*` are not supported in the `<object_name>`.

To add all HDL objects inside instance of a block, the instance name of the block can be used (for example: `wave add /UUT` is the same as `wave add /UUT/*` if `*` were supported).

wave log Command Examples

Log the signals associated with the module instances `/tb/UUT` and `/tb/child/gt` to the waveform database:

```
wave log /tb/UUT /tb/child/gt
```

Log all signals in the design:

```
wave log -r /
```

Waveform Window Commands

`wcfg new`

The `wcfg new` command creates a new wave configuration and opens it in a new window.

wcfg new Command Syntax

```
wcfg new
```

wcfg new Command Example

Create a new wave configuration:

```
wcfg new
```

`wcfg open`

The `wcfg open` command opens a wave configuration from a file into a new window.

wcfg open Command Syntax

```
wcfg open <filename>
```

The `<filename>` command option specifies the name of the WCFG file to open.

wcfg open Command Example

Open a WCFG file of the name, `toplevel.wcfg`:

```
wcfg open topLevel.wcfg
```

`wcfg save`

The `wcfg save` command saves the active wave configuration to a file.

wcfg save Command Syntax

```
wcfg save <filename>
```

wcfg save Command Option

The `wcfg save` command option, `<filename>`, specifies the name of the WCFG file to save.

wcfg save Command Example

Save the active wave configuration to a WCFG file named `toplevel.wcfg`:

```
wcfg save topLevel.wcfg
```

wcfg select

The `wcfg select` command makes the specified wave configuration the active window.

wcfg select Command Syntax

```
wcfg select <wave_config_name>
```

wcfg select Command Option

The `wcfg select` command option `<wave_config_name>` specifies the wave configuration to activate. The `<wave_config_name>` must be the name of an open, existing wave configuration; otherwise, the command reports an error.

wcfg select Command Example

Activate the wave configuration named `design`:

```
wcfg select design
```

wave add

The `wave add` command adds HDL object(s) to the currently active wave configuration in the ISim graphical user interface and logs simulation output of the HDL object(s) to a waveform database (wdb) file. The waveform database file is named as `isim.wdb` by default and can be changed using the `-wdb` switch with the simulation executable. The wave configuration displays in the Wave window.

The equivalent GUI command is to right-click in the wave window for the context menu and select the **Add to Wave Window** option.

wave add Command Syntax

```
wave add [-into <ID>] [-reverse] [-radix <radix>] [-color  
<color>] [-name <custom_name>] [-r] [<object_name>]
```

wave add Command Options

Table 9-14: Wave add Command Options

Option	Description
<code>-into <ID></code>	Specifies object ID of the group or virtual bus into which the object should be added.
<code>-reverse</code>	Reverses the bus order (MSB to LSB or vice versa).
<code>-radix <radix></code>	Uses the specified radix when displaying signal values. The value of <code><radix></code> is one of <code>default</code> , <code>bin</code> , <code>oct</code> , <code>hex</code> , <code>dec</code> , <code>unsigned</code> , or <code>ascii</code> .

Table 9-14: Wave add Command Options (Cont'd)

Option	Description
-color <color>	<p>Sets the simulation object(s) to the specified <color>. The value of is defined in RGB format.</p> <p>For example: #0000FF is blue, #FF0000 is red, and #00FF00 is green.</p> <p>You can also specify the textual name of color can for some of the popular colors. The following color names are accepted: black, red, darkred, green, darkgreen, blue, darkblue, cyan, darkcyan, magenta, darkmagenta, darkyellow, gray, darkgray, lightgray. The RGB values for these colors are defined in the RGB table.</p>
-name <custom_name>	Names the wave object with a custom name.
-r	Adds objects associated with each block 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.
<object_name>	<p>Specifies HDL object whose simulation output is to be logged to the waveform database. The <object_name> can also be a hierarchical instance name (for example: /tb/UUT) of a block, in which case all HDL objects within the given block are logged. This option does not support the use of wild card characters. To add all HDL objects inside instance of a block, use the block instance name.</p>

wave add Command Examples

The wave add command can be used as follows.

Add the signals associated with object UUT to the current wave configuration:

```
wave add /tb/UUT
```

Add all top-level signals:

```
wave add /
```

Add all signals in the design in color whose RGB value is #00FF10:

```
wave add -r / -color #00FF10
```

Add specific signals with radix hex and in color red:

```
wave add /tb/clock /tb/UUT/data -radix hex -color red
```


divider add

The divider add command adds a new divider.

divider add Command Syntax

```
divider add [-into <ID>] [-color <color>]
```

Where:

- `-into <ID>`
Specifies the object ID of the group into which the divider should be added
- `-color <color>`
Sets the divider color to the specified `<color>`. The value of is defined in RGB format.

For example: #0000FF is blue, #FF0000 is red, and #00FF00 is green.

You can also specify the textual name of color can for some of the popular colors. The following color names are accepted: black, red, darkred, green, darkgreen, blue, darkblue, cyan, darkcyan, magenta, darkmagenta, darkyellow, gray, darkgray, lightgray. The RGB values for these colors are defined in the RGB table.

divider add Command Examples

Add a divider with name `Inputs` to the current wave configuration:

```
divider add Inputs
```

Add red divider with name `Outputs`:

```
divider add Outputs -color red
```

Add dividers into a group:

```
set test_group_id [group add test_group]
wave add "dcm_clk_s" /tb/data2 -into $test_group_id
divider add data -color blue -into $test_group_id
wave add "addr1" /tb/UUT/addr2 -into $test_group_id
divider add address -color red -into $test_group_id
```

group add

The group add command adds a new group.

group add Command Syntax

```
group add [-into <ID>]
```

Where:

- `-into <ID>`
Specifies the object ID of an already existing group into which to add a newly created group.

group add Command Examples

Add a group with name `Inputs` to the current wave configuration:

```
group add Inputs
```

Create a group that adds a simulation object, `dcm_clk_s`, to the group:

```
set test_group_id [group add test_group]
wave add "dcm_clk_s" -into $test_group_id
```

Create groups within a group:

```
set group_id [group add test_group]
set group_id_1 [group add group_1 ?into $group_id]
set group_id_2 [group add group_2 ?into $group_id]
wave add clk read_ok -into $group_id_1
wave add data_w -into $group_id_2
```

`virtualbus add`

The `virtualbus add` command adds a new virtual bus to the currently active waveform configuration. The bus is empty when it is created. Subsequently, HDL objects can be added to populate the virtual bus as desired.

virtualbus add Command Syntax

```
virtualbus add <name> [-into <ID>] [-reverse] [-radix <radix>]
[-color <color>]
```

virtualbus add Command Options

Option	Description
<name>	Name of the virtual bus.
-into <ID>	Specifies the object ID of an existing virtual bus into which to add the new virtual bus.
-reverse	Reverses the bus order.
-radix <radix>	Uses the specified radix when displaying signal values. The value of <radix> is one of <code>bin</code> , <code>oct</code> , <code>hex</code> , <code>signed</code> , <code>dec</code> , or <code>ascii</code> .
-color <color>	Sets the virtual bus color as specified. The value of is defined in RGB format. For example: #0000FF is blue, #FF0000 is red, and #00FF00 is green. You can also specify the textual name of a color for some of the popular colors. The following color names are accepted: <code>black</code> , <code>red</code> , <code>darkred</code> , <code>green</code> , <code>darkgreen</code> , <code>blue</code> , <code>darkblue</code> , <code>cyan</code> , <code>darkcyan</code> , <code>magenta</code> , <code>darkmagenta</code> , <code>darkyellow</code> , <code>gray</code> , <code>darkgray</code> , <code>lightgray</code> . The RGB values for these colors are defined in the RGB table.

virtualbus add Command Examples

Add a virtual bus with name `mybus` having radix as hexadecimal to the current wave configuration:

```
virtualbus add <mybus> -radix hex
```

Create a virtual bus that adds two simulation objects, `sigA` and `sigB`, to the virtual bus:

```
set vbusId [virtualbus add mybus -radix hex]
wave add sigA -into $vbusId
wave add sigB -into $vbusId
```

`marker add`

The `marker add` command adds a new marker.

marker add Command Syntax

```
marker add <time>
```

The `<time>` option lets you specify the time location at which to add the new marker. If you do not specify the time unit, you can get the default user time unit by running the `isim get userunit` command.

marker add Command Examples

Add a marker at 10 ns to the current wave configuration: `marker add 10 ns`

Library Mapping File (*xilinxim.ini*)

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

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

1. `$(XILINX)/vhdl/hdp/<platform>`.
2. User-file specified through the `-initfile` switch. If `-initfile` is not specified, the program searches for `xilinxim.ini` in the current working directory.

The `xilinxim.ini` file has the following syntax:

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

The following is an example `xilinxim.ini` file:

```
VHDL
std=C:/libs/vhdl/hdp/
stdieee=C:/libs/vhdl/hdp/ieee
work=C:/workVerilog
unisims_ver=$(XILINX)/rtf/verilog/hdp/nt/unisims_ver
xilinxcorelib_ver=C:/libs/verilog/hdp/nt/xilinxcorelib_ver
mylib=./mylib
work=C:/work
```

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

- There must be no more than one library path per line inside the `xilinxim.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 the `$` character.
- The default physical directory for a logical library is `isim/<logical_library_name>`.
- File comments must start with `--`

Appendix B

Exceptions to VHDL and Verilog Language Support

VHDL Language Support Exceptions

ISim supports:

- VHDL IEEE-STD-1076-1993
- Verilog IEEE-STD-1364-2001

with exceptions as noted in the Exceptions Column in the following tables.

Table B -1: VHDL Language Support Exceptions

Supported VHDL Construct	Exceptions
abstract_literal	Floating point expressed as based literals are not supported.
aggregate	Mixing choice directions in an aggregate is not supported.
alias_declaration	Alias to non-objects are in general not supported; particularly the following: <ul style="list-style-type: none"> • Alias of an alias • Alias declaration without subtype_indication • Signature on alias declarations • Operator symbol as alias_designator • Alias of an operator symbol • Character literals as alias designators
alias_designator	<ul style="list-style-type: none"> • Operator_symbol as alias_designator • Character_literal as alias_designator
association_element	Globally, locally static range is acceptable for taking slice of an actual in an association element.
attribute_name	Signature after prefix is not supported.
binding_indication	Binding_indication without use of entity_aspect is not supported.
bit_string_literal.	Empty bit_string_literal ("") is not supported
block_statement	Guard_expression is not supported; for example, guarded blocks, guarded signals, guarded targets, and guarded assignments are not supported.

Table B -1: VHDL Language Support Exceptions (Cont'd)

Supported VHDL Construct	Exceptions
choice	Aggregate used as choice in case statement is not supported.
concurrent_assertion_statement	Postponed is not supported.
concurrent_signal_assignment_statement	Postponed is not supported.
concurrent_statement	Concurrent procedure call containing wait statement is not supported.
conditional_signal_assignment	Keyword guarded as part of options is not supported as there is no supported for guarded signal assignment.
configuration_declaration	Non locally static for generate index used in configuration is not supported.
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.
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.
function_call	In named parameter association in a function_call slicing, indexing or selection of formals is not supported.
instantiated_unit	Direct configuration instantiation is not supported.
mode	Linkage and Buffer ports are not supported completely.
options	Guarded is not supported.
primary	At places where primary is used, allocator is expanded there.
procedure_call	In named parameter association in a procedure_call slicing, indexing or selection of formals is not supported.
process_statement	Postponed processes are not supported.
selected_signal_assignment	The "guarded" keyword as part of options is not supported as there is no support for guarded signal assignment.
signal_declaration	Signal_kind is not supported. Signal_kind is used for declaring guarded signals, which are not supported.
subtype_indication.	Resolved subtype of composites (arrays and records) is not supported
waveform.	Unaffected is not supported.
waveform_element	Null waveform element is not supported as it only has relevance in the context of guarded signals.

Verilog Language Support Exceptions

The following table lists the exceptions to supported Verilog language support.

Table B -2: Verilog Language Support Exceptions

Verilog Construct	Exception
Compiler Directive Constructs	
<code>`celldefine</code>	not supported
<code>`endcelldefine</code>	not supported
<code>`undefs</code>	Supports parameterized <code>`define</code> macros.
<code>`unconnected_drive</code>	not supported
<code>`nounconnected_drive</code>	not supported
Attributes	
<code>attribute_instance</code>	not supported
<code>attr_spec</code>	not supported
<code>attr_name</code>	not supported
Primitive Gate and Switch Types	
<code>cmos_switchtype</code>	not supported
<code>mos_switchtype</code>	not supported
<code>pass_en_switchtype</code>	not supported
Generated Instantiation	
<code>generated_instantiation</code>	<p>The <code>module_or_generate_item</code> alternative is not supported.</p> <p>Production from 1364-2001 Verilog standard:</p> <pre>generate_item_or_null ::= generate_conditional_statement generate_case_statement generate_loop_statement generate_block module_or_generate_item</pre> <p>Production supported by Simulator:</p> <pre>generate_item_or_null ::= generate_conditional_statement generate_case_statement generate_loop_statement generate_blockgenerate_condition</pre>

Table B -2: Verilog Language Support Exceptions (Cont'd)

Verilog Construct	Exception
genvar_assignment	<p>Partially supported.</p> <p>All generate blocks must be named.</p> <p>Production from 1364-2001 Verilog standard:</p> <pre>generate_block ::= begin [: generate_block_identifier] { generate_item } end</pre> <p>Production supported by Simulator:</p> <pre>generate_block ::= begin: generate_block_identifier { generate_item } end</pre>
Source Text Constructs	
Library Source Text	
library_text	not supported
library_descriptions	not supported
library_declaration	not supported
include_statement	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	
config_declaration	not supported
design_statement	not supported
config_rule_statement	not supported
default_clause	not supported
System Timing Check Commands	
\$skew_timing_check	not supported
\$timeskew_timing_check	not supported
\$fullskew_timing_check	not supported
\$nochange_timing_check	not supported
System Timing Check Command Argument	
checktime_condition	not supported
PLA Modeling Tasks	
\$async\$nand\$array	not supported
\$async\$nor\$array	not supported
\$async\$or\$array	not supported

Table B -2: Verilog Language Support Exceptions (Cont'd)

Verilog Construct	Exception
\$sync\$and\$array	not supported
\$sync\$nand\$array	not supported
\$sync\$nor\$array	not supported
\$sync\$or\$array	not supported
\$async\$and\$plane	not supported
\$async\$nand\$plane	not supported
\$async\$nor\$plane	not supported
\$async\$or\$plane	not supported
\$sync\$and\$plane	not supported
\$sync\$nand\$plane	not supported
\$sync\$nor\$plane	not supported
\$sync\$or\$plane	not supported
Value Change Dump (VCD) Files	
\$dumpportson \$dumpports \$dumpportsoff, \$dumpportsflush, \$dumpportslimit \$vcdplus	not supported

Appendix C

Migrating from ModelSim XE to ISim

Retargeting from a ModelSim XE simulation environment to a Xilinx® ISim simulation environment can be accomplished without significantly modifying the existing environment.

This overview identifies and details the appropriate migration guidelines and other considerations for making the switch from ModelSim XE to ISim.

To get the best use of the underlying innovations of ISim, a video demonstration and tutorials are also available:

- Tutorials: http://www.xilinx.com/support/documentation/dt_ise.htm
- ISim Video Demos:
http://www.xilinx.com/products/design_resources/design_tool/resources/index.htm
- ISim Product Page: <http://www.xilinx.com/tools/isim.htm>

About ModelSim XE

ModelSim XE stands for ModelSim Xilinx Edition, which is an OEM product from Mentor Graphics. ModelSim XE provides a complete HDL simulation environment that lets you verify the functional and timing models of your design, and your HDL source code. ModelSim XE was discontinued in the Xilinx ISE® Design Suite 12.4 tool release. See the Product Discontinuance Notice at:

http://www.xilinx.com/support/documentation/customer_notices/xcn10028.pdf

ModelSim XE was shipped with each major Xilinx ISE Design Suite release through version 12.3. There were two versions:

- ModelSim XE Starter - a free version that can be downloaded from the Xilinx website. A starter license is required for using this product.
- ModelSim XE Full - an OEM version from Mentor Graphics, based on their PE product line.

About ISim

ISim is a Xilinx simulation product that provides a complete, full-featured HDL simulator integrated within the ISE tool and the PlanAhead™ tool, Embedded Development Kit (EDK), and System Generator.

ISim is available with all major Xilinx tools releases and comes in two versions:

- ISim Lite: A limited version of the ISE Simulator. In this version, when your design plus test bench exceeds 50,000 lines of HDL code, the simulator begins to derate the performance of the simulator for that invocation.
- ISim Full: The full version of ISE Simulator.

Feature Comparison

Table C-1: ModelSim and ISim Feature Comparison

Feature	ModelSim XE	Starter ModelSim XE	Full ISim Lite	ISim Full
Line Limit (Statements)	10,000	40,000	50,000	None
Performance	30% of ModelSim PE or ModelSim DE	40% of ModelSim PE or ModelSim DE	Same as ModelSim XE	Same as ModelSim XE
Mixed Language	No	No	Yes	Yes
VHDL	Yes	Yes	Yes	Yes
Verilog	Yes	Yes	Yes	Yes
System Verilog for Design	No	No	No	No
System Verilog for Verification	No	No	No	No
Debugging Environment	Yes	Yes	Yes	Yes
Standalone Waveform Viewer	Yes	Yes	Yes	Yes
Memory Viewer/Editor	Yes	Yes	Yes	Yes
Verilog PLI/VPI	Yes	Yes	No	No
VHDL FLI/VHPI	No	No	No	No
Code Coverage	No	No	No	No
SecureIP/HardIP Support	No	No	Yes	Yes
EDK Support	No	No	Yes	Yes
System Generator Support	No	No	Yes	Yes
CORE Generator Support	Yes	Yes	Yes	Yes
MIG Support	No	No	Yes	Yes

Table C-1: ModelSim and ISim Feature Comparison (Cont'd)

Feature	ModelSim XE	Starter ModelSim XE	Full ISim Lite	ISim Full
Floating License	No	No	Yes	Yes
32-bit OS Support	Windows	Windows	Windows/Linux	Windows/Linux
64-bit (native) OS Support	No	No	Windows/Linux	Windows/Linux

Simulation Process

This section describes the different modes of simulation and the steps involved in simulation. Each sub-section explains the differences between the two simulators.

Figure C-1 shows the different steps in simulation and the process for each step.

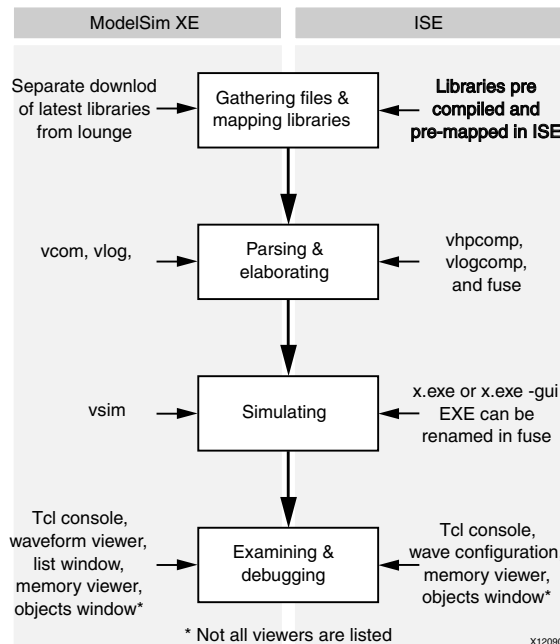


Figure C-1: Simulation Steps

Step 1: Gathering Files and Mapping Libraries

ModelSim XE Flow

The ModelSim XE libraries are downloaded from <http://www.xilinx.com/support/download/index.htm>.

Each time a new release of the ISE Design Suite is available; you must go to this area and download the libraries separately. These libraries are marked and must be used to ensure that the Xilinx libraries are not counted against the line count limits.

The `modelsim.ini` file delivered with ModelSim XE is pre-mapped with the correct Xilinx libraries.

ISim Flow

Libraries for ISim are updated as part of the standard Xilinx installation. No additional steps are needed. Mapping is also handled automatically by Xilinx. You do not need to know where to download from or how to map the Xilinx libraries to start simulating.

Step 2: Parsing and Elaborating the Design

ModelSim XE Flow

ModelSim XE uses the following commands for compilation and elaboration. VCOM options (VHDL Compiler) runs the VHDL compiler and compiles VHDL files to a specified directory. VLOG options (Verilog Compiler) runs the Verilog compiler and compiles Verilog files to a specified directory. VSIM options (VSIM simulator) elaborates the load for the simulation.

For each of these commands, multiple options give you additional control over compilation and elaboration. For a complete list of equivalent ModelSim XE commands, see [Appendix B, Exceptions to VHDL and Verilog Language Support](#).

ISim Flow

ISim uses the following commands for compilation and elaboration.

- `vhpcomp`: (VHDL Compiler) runs the VHDL compiler and compiles VHDL files to a specified directory.
- `vlogcomp`: (Verilog Compiler) runs the Verilog compiler and compiles Verilog files to a specified directory.
- `fuse`: (VSIM simulator) elaborates the load for the simulation and creates an executable that needs to be launched to run the simulation.

For each of these commands, multiple options give you additional control over compilation and elaboration.

Step 3: Simulating the Design

ModelSim XE Flow

Running VSIM elaborates the design and runs the simulation. By default, running `vsim` launches the GUI.

To run in command line mode use the `-c` switch.

ISim Flow

Running `fuse` creates a named executable. You must run this executable to launch the simulation. By default this executable is named `x.exe`, but you can change the name.

By default, running the executable runs the simulation in command line mode. To launch the GUI, use the `-gui` switch.

Step 4: Examining and Debugging the Design

Customizing Wave Operations

ModelSim XE and ISim provide the same capabilities for customizing the waveform window, but perform the customization differently. ModelSim XE uses standard Tools Command Language (Tcl) commands for all waveform operations. ISim uses a subset of Tcl commands, but the majority of the customizing is through the GUI, with results saved in the waveform configuration file.

The waveform configuration file for ISim is an XML-based file that you cannot edit, while the waveform Tcl commands in ModelSim XE can be modified. The load time for the wave configuration through the ISim implementation is faster because loading an XML file is faster than executing multiple Tcl commands.

Note: ISim does not have Tcl support for all wave configuration operations.

Measuring with Markers and Cursors

Measuring with markers and cursors is different between ModelSim XE and ISim. ModelSim XE provides cursors to measure between two points of interest. You can add cursors as needed, and each new cursor gets added under the existing cursors. The waveform viewer automatically shows the distance between the cursors. [Figure C-2](#) shows the ModelSim XE waveform with cursors.

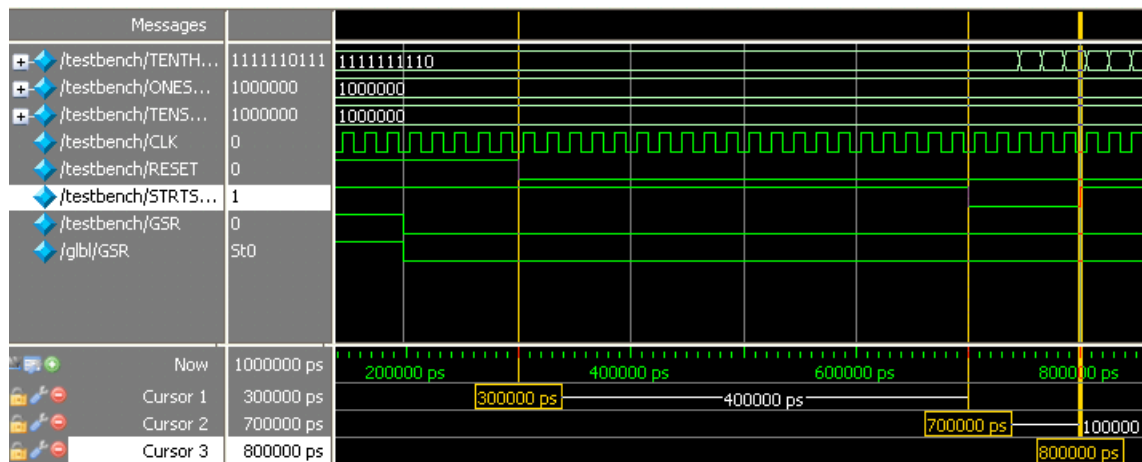


Figure C-2: ModelSim XE Waveform View with Cursor

ISim has a different approach to measuring. ISim uses both cursors and markers. While in ModelSim XE a cursor is a permanent measuring stick, ISim cursors are treated as temporary. ISim has a primary and a secondary cursor that together can be used to measure between two points. ISim markers let you measure between multiple points, including the primary cursor. [Figure C-3](#) shows the ISim measure in use.

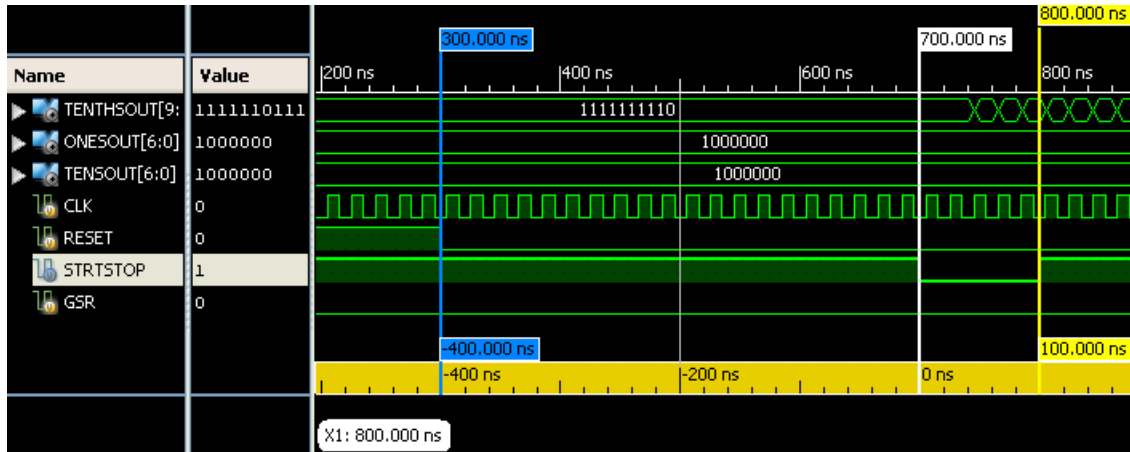


Figure C-3: ISim Measure in Use

ISim also provides a ruler for frame of reference. The selected marker or cursor is always the 0 location against which all other markers are measured. Figure C-3 shows how to measure between the edges of interest in ISim.

Note: You cannot rename markers in ISim.

Analog Waveforms

Contact Xilinx Technical Support for more information on availability.

Single-Click Compile and Reload

ModelSim XE has a true text editor built into the standalone GUI which lets you make changes to HDL code, recompile, and re-simulate.

The ISim GUI has a text viewer only for HDL files. When you make edits to the file, you cannot recompile and re-simulate. You must shut down the existing simulation, make HDL modifications in the ISE tool or PlanAhead tool text editor, then re-launch the simulation in ISim.

Project Navigator Integration

The ISE Project Navigator notifies you when ModelSim XE is not a valid simulator choice, and that you should select one of the other integrated simulators. The following figure shows the ISim selection in Project Navigator.

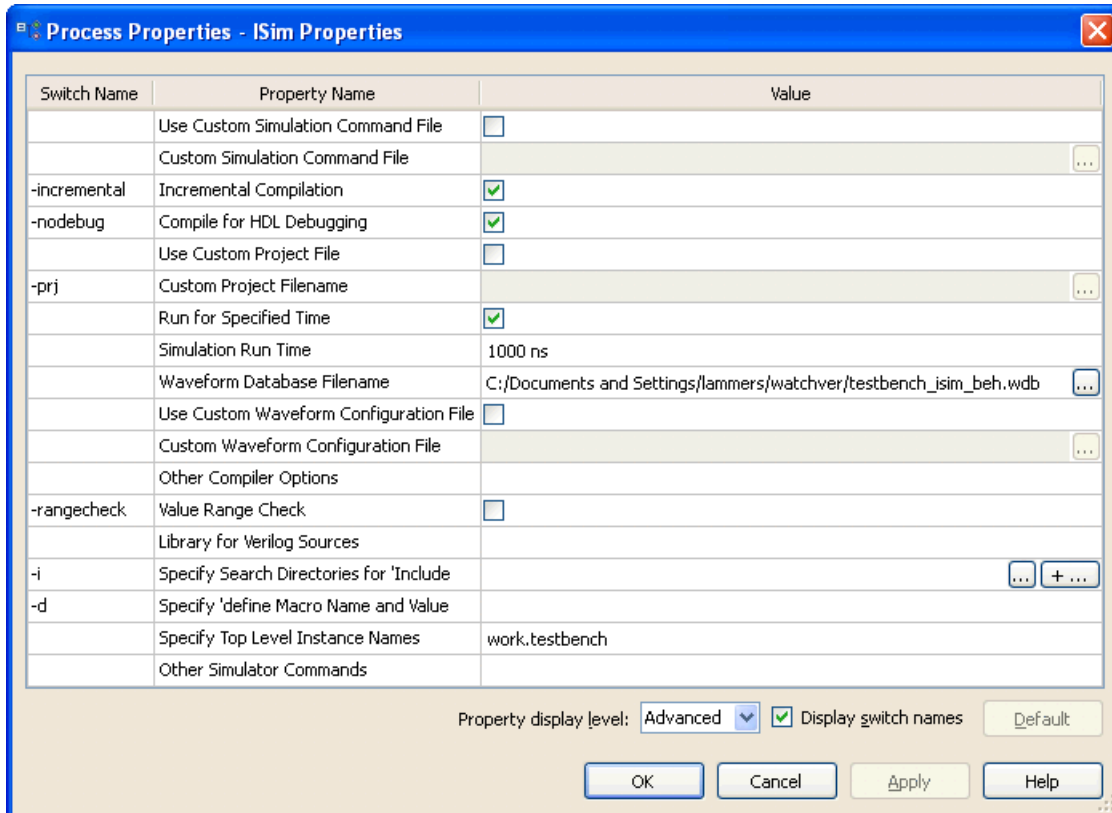


Figure C-4: Project Navigator Project Settings Dialog Box with ISim Selected

The simulation properties are similar between ModelSim XE and ISim; the following table lists the differences.

Table C-2: Simulation Properties in Project Navigator

ModelSim XE Property Name	ISim Property Name	Comments
Library Compilation		
Compiled Library Directory	N/A	Pre-compiled libraries delivered with ISE Design Suite installation for ISim.
Ignore Pre-compiled Library Warning Check	N/A	
Generate Verbose Library Compilation Messages	N/A	
Custom User Commands		
Use Custom Do File	Use Custom Simulation Command File Use Custom Wave Configuration File	ISim supports Tcl commands to control engine operation as well as to control most common GUI operations. In addition, it allows a faster way to set waveform window using wave configuration file.
Custom Do File	Custom Simulation Command File Custom Wave Configuration File	
Use Automatic Do File	N/A	You cannot prevent Project Navigator from creating the ISim script.
Custom Compile File List	Use Custom Project File Custom Project Filename	Lets you change the compile order of the file.
N/A	Waveform Database Filename	Lets you specify a different database for the simulation.
Custom Compiler Commands		
Other VSIM Command Line Options	Other Compiler Options Other Simulator Commands	ISim splits the VSIM commands into fuse commands and executable commands.

Table C-2: Simulation Properties in Project Navigator (Cont'd)

ModelSim XE Property Name	ISim Property Name	Comments
Other VLOG Command Line Options	Other Compiler Options	Passes the options to the fuse command in ISim.
Other VCOM Command Line Options		
Runtime Settings		
Simulation Run Time	Simulation Run Time	
Simulation Resolution	N/A	ISim defaults to 1ps.
Language Settings		
VHDL Syntax	N/A	The default in ISim is 93.
Use Explicit Declarations Only	N/A	N/A
Other VCOM Command Line Options	Value Range Check	ModelSim XE does not have specific options for this, but options can be specified with the "Other Command Line Options" property.
	Specify Search Directories for `include Incremental Compilation	
	Specify `define Macro Name and Value Incremental Compilation	
N/A	Compile for HDL Debugging	
Miscellaneous Settings		
Use Configuration Name	N/A	
Configuration Name	N/A	
Log All Signals In Simulation	N/A	
Other VSIM Command Line Options	Specify Top-Level Instance Name	

Appendix D

Additional Resources

Xilinx Resources

- **Device User Guides:**
http://www.xilinx.com/support/documentation/user_guides.htm
- **Xilinx Glossary:** <http://www.xilinx.com/company/terms.htm>
- **Xilinx Design Tools: Installation and Licensing Guide (UG798):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/iil.pdf
- **Xilinx Design Tools: Release Notes Guide (UG631):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/irn.pdf
- **Product Support and Documentation:** <http://www.xilinx.com/support>
- **Synthesis and Simulation Design Guide (UG626):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/sim.pdf
- **PlanAhead User Guide (UG632):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/PlanAhead_UserGuide.pdf
- **Command Line Tools User Guide (UG628):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/devref.pdf
- **ChipScope Pro Software and Cores User Guide (UG029):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/chipscope_pro_sw_cores_ug029.pdf
- **ISE Help:** http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/isehelp_start.htm
- **XPower Help:** http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/isehelp_start.htm#xpa_c_overview.htm

ISim Tutorials

- **ISim In-Depth Tutorial (UG682):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/ug682.pdf
- **ISE Hardware Co-Simulation Tutorial: Accelerating Floating Point FFT Simulation (UG817):**
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ug817_fft_sim_tutorial.pdf

