ISim User Guide

UG660 (v13.3) December 7, 2011

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Description
10/19/2011	13.3	Updated with modifications to match release. Added: • Revision History (this topic) • Additional Resources Appendix • Determining the Ethernet Port • Tutorial References: Tutorials Updated: • Clarified .wcfg and .wdb in Saving the Results • Text throughout the document. • Removed supported boards from Fuse command. Added:Board Support with new supported boards listed. • ISim GUI description: GUI Overview
12/07/2011	13.3	Added: Supported properties, selection options, and new Type value to Board Support. Removed: References to OS Support.

Table of Contents

XILINX®

Chapter 1

Getting Started

ISim Overview

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.

Simulation Libraries

The Xilinx simulation device libraries are precompiled, and updated automatically when updates are installed.

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

Language Support

ISim supports the following languages.

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

Feature Support

ISim supports the following features.

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

ISim Operating System Support

You can run ISim as specified in the *ISE Design Suite: Installation and Licensing Guide (UG798)*.

Steps in Simulation

ISim Modes of Operation

There are three modes of operation available in ISim:

- Graphical User Interface (GUI)
- Interactive Command Line
- Non-Interactive Batch

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

Steps in a Simulation Overview

The basic steps for simulating a design in ISim are as follows:

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

Step 1: Gathering Files and Mapping Libraries

The required files to run a simulation in ISim are:

- Design files, including stimulus file
- Any user libraries
- Any other miscellaneous data files

Stimulus File

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

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

User Libraries

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

- When launching Project Navigator, define user libraries in the ISE software. 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 to point to the user logical/physical libraries.
- When launching ISim from PlanAhead™ software, define the user libraries in that software. See the *PlanAhead User Guide (UG632)* for more information.

Step 2: Parsing and Elaborating the Design

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

Graphical User Interface Mode

When you launch the ISim graphical user interface:

- The design is parsed and design components are elaborated for you when you invoke ISim from either the ISE® software or the PlanAhead™ software. For details, see "Simulation from ISE Software" 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. And then the generated simulation executable is invoked with `-gui` to launch graphical user interface.

Interactive Command Line Mode

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

Creating a Project File -

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

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

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

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

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

   where:

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

Using fuse -

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

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

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

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

Step 3: Simulating the Design

After design compilation and elaboration, the next step is to run the simulation executable, and simulate the design.

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

Graphical User Interface Mode

Simulation at the Command Line

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

Optionally, you can also invoke the simulation executable, launch the GUI, and run simulation with a Tool Command Language (Tcl) file by leveraging the **`-tclbatch`** option, for example, **`my_sim.exe -gui -tclbatch my_sim.tcl`**.

You can use the wave add command (example: **`wave add /`** to add all signals at top level) in your `my_sim.tcl` file to automatically trace the signals and display the signals in the GUI upon launch.

Simulation from the ISE software

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

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

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

See "Simulation Properties" in ISE Help for details.

To run for an additional time, use one of the **run** simulation commands described in Running a Simulation in ISim.

For more information, see Launching the ISim GUI.

Interactive Command Line Mode

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

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

Make sure that this step runs 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 in two ways:

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

You can save the results; see Saving the Results.

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

Step 5: Debugging the Design

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

Examining Error Messages

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

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

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

Examining Log Files

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

- **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 `isim/<simulation_executable> .sim` directory. Provide this file to Xilinx Technical Support for further assistance. This file also does not disclose any design data, and is safe to share with Xilinx Technical Support if you report a problem.

Using Tcl Simulation Commands

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

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

For more commands, see Simulation Command Overview.

Debugging in the Graphical User Interface

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

Tutorials

For tutorials on how to use ISim, see the following table:

ISE Simulator (ISim) In-Depth Tutorial (UG682)	This tutorial demonstrates how to use ISim for design simulation and debugging.
ISE Hardware Co-Simulation Tutorial: Accelerating Floating Point FFT Simulation (UG817)	This tutorial shows how to use the ISim Hardware Co-Simulation to accelerate Floating Point FFT simulation.
ISE Hardware Co-Simulation Tutorial: Interacting with Spartan-6 Memory Controller and On-Board DDR2 Memory, (UG818)	This tutorial shows to use the ISim Hardware Co-Simulation feature to interact with the Spartan®-6 device Memory Controller and the on-board DDR2 memory.
ISE Hardware Co-Simulation Tutorial: Processing Live Ethernet Traffic through Virtex-5 Embedded Ethernet MAC, (UG819)	This tutorial shows to use the ISim Hardware Co-Simulation feature to process live Ethernet traffic through the Embedded Ethernet MAC on the Virtex®-5 device.

XILINX®

Chapter 2

Exploring the ISim Graphical User Interface

Graphical User Interface Overview

The ISim Graphical User Interface (GUI) consists of the main window, which contains panels, the Workspace, toolbars, and the status bar. In the main window, you can:

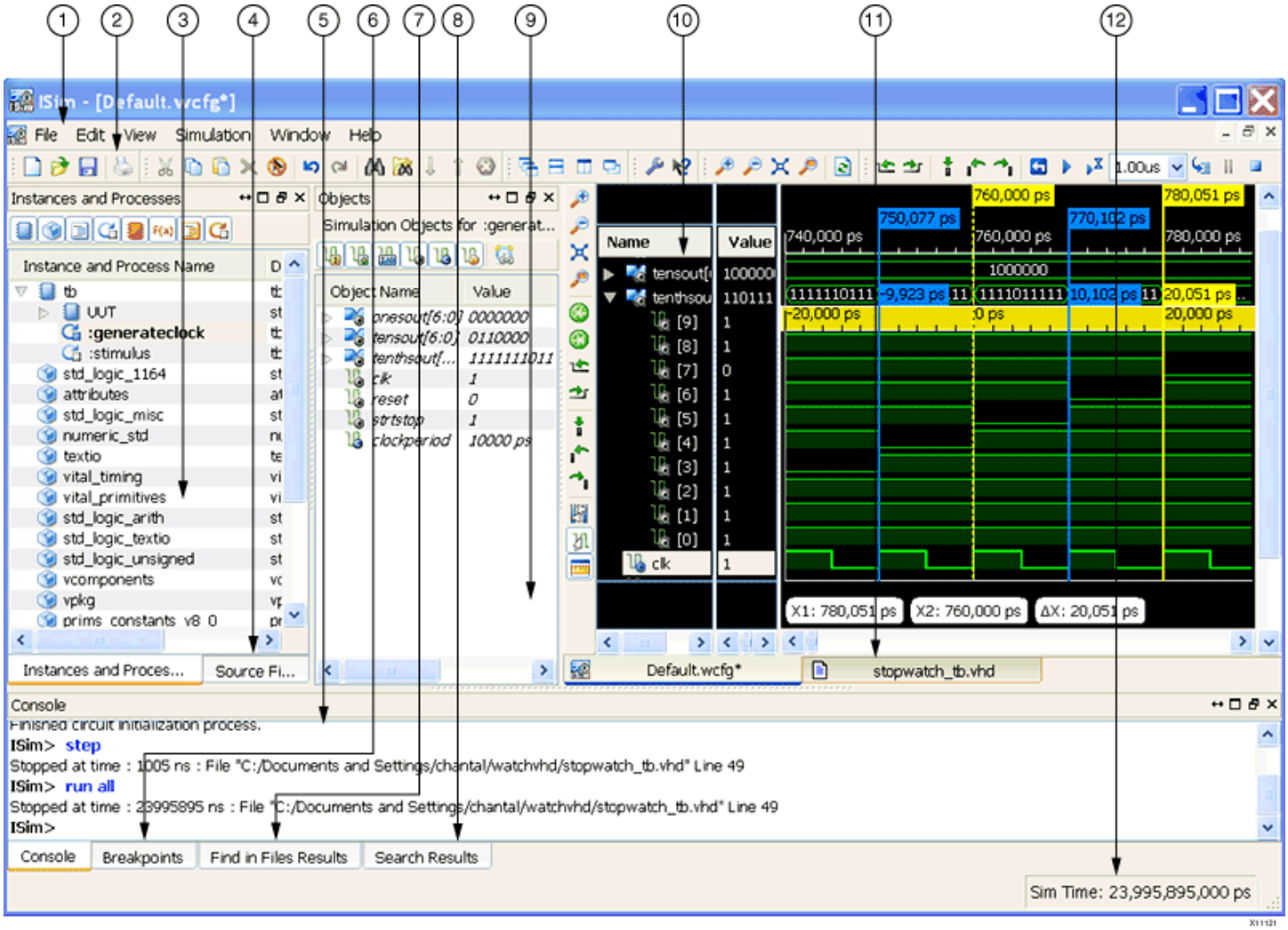
- View the parts of the design that can be simulated
- Add and view signals in the wave configuration
- Use commands to run simulation
- Examine the design, and debug as necessary

Launching the GUI

The ISim GUI launches when you run the simulation executable, from the ISE® software, from the command line, or from the PlanAhead™ application. For details, see Step 3: Simulating the Design .

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

Parts of the GUI



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

#	Part of the GUI	Description
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 Simulation Time.

Design Objects and Icons

Design Hierarchy Icons

Design entities (Verilog) and modules (VHDL) display in a design hierarchy in the Instances and Processes Panel.


VHDL Entity


VHDL Package


VHDL Block


VHDL Process


Verilog Module


Verilog Task or Function


Verilog Block


Verilog Process

Design Object Icons

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

Signals


Input Port


Output Port


InOut, Bidirectional Port


Internal Signal


Constants, parameters, and generics


Variable


Linkage Signal (VHDL only)


Buffer Signal

Buses


Input Bus


Output Bus


InOut, Bidirectional Bus


Internal Bus


Constant, Parameter and Generics Bus


Variable Bus


Linkage Bus


Buffer Bus

Arranging the Main Window

Arranging Windows

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

Using Window Commands

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

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.

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**.

Menu Commands

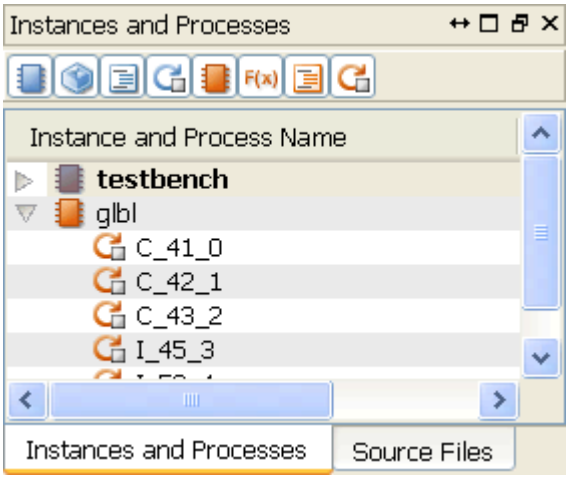
The View menu commands enable you to hide the main window panels, toolbar, and status bar.

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

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

Toggle Icons

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



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

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

Toggle Maximized - maximizes the panel.

Click again to restore the panel size.

Toggle Floating - floats the panel.

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

Close - closes the panel from view.

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

Expanding and Collapsing a Hierarchy

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

Clicking the Arrow

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

 - Click the arrow to collapse the hierarchy.

Menu Command

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

Using the Context Menu

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

Wave Window

Wave Window Overview

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

In the GUI, the signals and buses in the wave configuration are traced during simulation, and therefore, the wave configuration is used to examine the simulation results. The design hierarchy and the signal transitions are not part of the wave configuration but rather are stored in a separate database (`.wdb`) file.

Wave Configuration File (.wcfg)

A wave configuration comprises a list of signals, their properties; such as color and radix value, and other wave objects; such as dividers, 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 the Results.

Default.wcfg Details

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

Otherwise, the content of `Default.wcfg` is not saved on disk.

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 **`wave 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, the **`wcfg new`** and **`wcfg open`** commands change the active window to the newly created window just like **File > New** and **File > Open**.

Signals/Buses in the Wave Configuration

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

Signals


Input Port


Output Port


InOut, Bidirectional Port


Internal Signal


Constants, parameters, and generics


Variable


Linkage Signal (VHDL only)


Buffer Signal

Buses


Input Bus


Output Bus


InOut, Bidirectional Bus


Internal Bus


Constant, Parameter and Generics Bus


Variable Bus


Linkage Bus


Buffer Bus

Objects in the Wave Configuration

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

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

Markers - A marker is used to mark a particular time for future reference. A marker is a vertical line intersecting the waveform. A marker 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 Adding Markers and Displaying Waveform Values With Markers.

Hollow/Filled-in Circle - When placing or moving cursors and markers, the **Snap to Transition** button assists with placing the cursor/marker more precisely on a signal transition.

– When placing or moving a cursor or marker, the mouse displays a hollow circle ○ when hovering between transitions.

– The mouse displays a filled in circle when hovering over a transition of a signal ●.

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

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

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

Wave Window Toolbar Icons

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








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

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

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

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

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

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

Working With Wave Configurations

Creating New Wave Configurations

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

To Create a Wave Configuration

1. Select **File > New**.

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

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

If more than one wave configuration is open:

- Use the wave configuration tab to locate a particular wave configuration, or
- select **Window > Next** or **Window > Previous** to navigate through open wave configurations.

Adding Signals to the Wave Configuration

You can populate the Wave window with the signals from your design using menu commands or drag and drop capabilities in the 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.

In the GUI

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

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

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 by entering the scope command in the Console panel.
2. In the Console panel, enter the wave add command to enter an individual object or a group of objects.

Wave configurations and .wcfg files

Though 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.
- 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 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. In saving to the file, the wave configuration name is changed to match the `.wcfg` file name
- When loading a wave configuration from a `.wcfg` file, the wave configuration uses the name of the `.wcfg` file.

Saving Wave Configurations

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

To Save a Wave Configuration

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

The current simulation wave configuration is saved.

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

Opening and Closing Wave Configurations

To Open a Wave Configuration File

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

- Opening a Live Simulation
- Opening a Static Simulation

To Close a Wave Configuration File

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

Instances and Processes Panel

Instances and Processes Panel Overview

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

Three columns display in this panel, as follows:

- The first column displays the instance, process, and static tasks or functions in a tree structure showing the block hierarchy of the design.
- The second column displays the names of the design units (Verilog module or VHDL entity (architecture) corresponding to the instance, static task/function, or process from the first column.
- The third column displays the type of the instance, static task/function, or process.

The following describes the icons used for the items displayed in this panel:


VHDL Entity


VHDL Package


VHDL Block


VHDL Process


Verilog Module

Verilog Task or Function


Verilog Block


Verilog Process

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

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

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

Searching For Objects

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

To Search For Objects

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

6. Click **OK**.

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

Opening HDL Source Files

In ISim, you can open Hardware Description Language (HDL) source file for viewing only in the text editor. Files open in read-only mode.

Opening HDL Source Files in Read Mode

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

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

Opening HDL Source Files in Write Mode

You can also open a file using the **File > Open** menu command. In the **Open** dialog box, change **Files of type** file to **Verilog** or **VHDL**, select the file, and click **Open**. This method opens the files in write mode.

Objects Panel

Objects Panel Overview

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

The top of the panel displays which instance 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 toolbar icon.
- **Data Type -** Displays the data type of the corresponding simulation object, logic or an array.

Objects Panel Toolbar Icons


Hides/displays input ports.


Hides/displays output ports.


Hides/displays inout, bidirectional ports.


Hides/displays internal signals.


Hides/displays constants, parameters, and generics.


Hides/displays variables.


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

Searching For Objects

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

To Search For Objects

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

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

Using Show Drivers

You can use the Show Driver command to display the driver for a change in signal, or object value. This command is used to determine the cause of a value change, which 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 (`show driver` command)

To Show Drivers

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

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

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

Showing Display Elements

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

To Display all Child Elements

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

The number of children in the object hierarchy display.

To Limit the Display of Child Elements

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

To Change the Preset Maximum Number of Child Elements

Set the preference settings as follows:

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

The number of children displayed in the object hierarchy updates immediately.

Selecting an Object in the Wave Window

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

To Select an Object in the Wave Window

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

The signal associated with the object highlights.

Source Files Panel

Source Files Panel Overview

The Source Files panel displays the 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. The HDL source files are available for quick access to the read-only source code.

Opening the Source Code

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

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

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

Text Editor Window

Text Editor Window Overview

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

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

Do Not edit your Hardware Description Language (HDL) files in the ISim text editor. Editing files could result in a conflict with the files in your ISE® software project.

Modifying Source Files

Follow these steps to modify source files.

Note HDL files must be edited outside of ISim to avoid design conflicts.

1. In ISE, open the source file in ISE Text Editor.
2. Make the appropriate edits.
3. Run the design through ISE tools to update the design.
4. Simulate the design.

Opening HDL Source Files

In ISim, you can open Hardware Description Language (HDL) source file for viewing only in the text editor. Files open in read-only mode.

Opening HDL Source Files in Read Mode

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

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

Opening HDL Source Files in Write Mode

You can also open a file using the **File > Open** menu command. In the **Open** dialog box, change **Files of type** file to **Verilog** or **VHDL**, select the file, and click **Open**. This method opens the files in write mode.

Viewing HDL Source Files

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

Standard text editor operations are available to assist you with viewing your Hardware Description Language (HDL) files, such as Zoom, and Find.

Information about specific operations is available in the ISE Text Editor Help.

Never edit your HDL files in the ISim text editor. Editing files could result in a conflict with the files in your ISE® software project.

To View Source File Contents

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

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

Setting Breakpoints

In ISim, 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 to Debug Your Design.

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

Setting and Removing Breakpoints

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

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

To remove a breakpoint, click the breakpoint to remove it.

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

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

A list of breakpoints is available in the Breakpoints panel.

Memory Editor Window

Memory Editor Window Overview

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 where memory objects could be listed: Memory Panel, Object Panel, and Search Result Panel. 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.
- Right-click a two-dimensional array of logic type in Objects Panel, and select **Memory Editor** from the context menu.
- Run a search on a memory name in the Instance and Processes panel. After the searched memory displays on the Search Results Panel, 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 corresponding to the two-dimensional array shows up in the main window as a tab along with the Waveform Viewer or Text Editor.

ISim Memory Editor GUI Descriptions -

- **Address -** The Address is used for going to a particular location in the Memory Editor.
- **Columns -** The Columns combo box is to control the display of the number of elements per row. The auto column displays the maximum of 2 to power *N* of elements.
- **Address Radix -** The address Radix combo box controls the radix of the address displayed in the Memory Editor.
- **Value Radix -** The Value Radix combo box 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

Console Panel Overview

The Console panel lets you view a log of commands generated by ISim, and enter standard and ISim-specific Tools Command Language (Tcl) commands at the command prompt.

- **Messages -** Messages that are generated by the ISim include errors, warnings, and informational messages, and display in the Console panel. 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 Entering Simulation Commands for more information.

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

Breakpoints Panel

Breakpoints Panel Overview

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 Debugging Your Design Using Breakpoints.

See also Setting Breakpoints and Deleting Breakpoints.

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

Breakpoint Toolbar Icons


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


Deletes all breakpoints from the HDL source files.


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

Search Results Panel

Search Results Panel Overview

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

Search Results Toolbar Commands

The following functions are available in the Search Results panel.


Clears the contents of the Search Results panel.


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


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


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


Stops the search.

Find in Files Results Panel

Using Find in Files

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

To Use Find in Files

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

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

Toolbar Commands and Shortcuts

ISim Toolbar Commands

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

Standard Toolbar

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

New — Opens the **New** dialog box and lets you select the type of file you want to create.

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

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

Print — Opens the **Print** dialog box and lets you print the active file.

Edit Toolbar

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

Unselect All — Unselects everything in the active window.

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

Redo — Reverses your latest "undone" operation.

Find — Searches for a text string in the active window.

Help Toolbar

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

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

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

Window Toolbar

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

View Toolbar

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

 Refresh — Cleans up the display of the file in focus.

ISim Toolbar

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

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

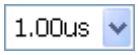
Moves the main cursor to the nearest marker to the left of the main cursor's current position.

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

Resets the simulation time to zero.

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

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



Specifies the amount of time the simulation runs when you click the **Run For** button.

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.

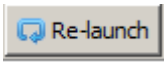
Ends the current simulation, leaving the simulation data open.

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

Re-launching ISim Simulation

Re-launching Simulation in the ISim GUI

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 software or the PlanAhead™ application.

What to Expect

Successful Simulation Re-launch

When you **Re-launch** a simulation in the ISim GUI successfully your simulation is complete without errors.

Unsuccessful Simulation Re-launch

When you **Re-launch** an unsuccessful simulation in the ISim GUI a dialog box opens with the syntax error failure to compile source code. The links take you directly 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

Force Selected Signal Dialog Box

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. Assignments made from within HDL code or any previously applied constant or clock force are overridden by the newly applied constant force.

Click **Apply** to apply all changes.

Signal Name

Displays the default signal name. The default signal name is the full path name of the item selected in the Object Window 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 will turn 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

You can specify a force constant value. The Force to Value uses the radix defined in Value Radix.

Starting at Time Offset

The force command starts after the specified time. The default starting time is 0. Time can be a string, such as 10 or 10 ns. When a number is entered without a unit, the default simulation time unit is used.

Cancel after Time Offset

The force command 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

Define Clock Dialog Box

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). Assignments made from within HDL code or any previously applied constant or clock force are overridden by the newly applied clock pattern.

Click **Apply** to apply changes.

Signal Name

Displays the default signal name. The default signal name 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: Binary, Hexadecimal, Unsigned Decimal, Signed Decimal, Octal, and ASCII.

Leading Edge Value

You can specify the first edge of the clock pattern. The Leading Edge Value uses the radix defined in Value Radix.

Trailing Edge Value

You can specify the second edge of the clock pattern. The Trailing Edge Value uses the radix defined in Value Radix.

Starting at Time Offset

The **force** command starts 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 a number is entered without a unit, ISim uses the default user unit as returned by Tcl command `isim get userunit`.

Cancel after Time Offset

The **force** command cancels after the specified time from the current simulation time. Time can be a string, such as 10 or 10 ns. When a number is entered without a unit, ISim uses the default simulation time unit.

Duty Cycle (%)

The percentage of time that the clock pulse is in an active state. The acceptable value ranges from 0 to 100.

Period

The length of the clock pulse, defined as a time value. Time can be a string, such as 10 or 10 ns.

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 1 us), set the following fields:

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

ISim Preferences

Setting ISim Preferences

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

To Set Preferences

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

The Preference settings are saved and are effective immediately in your 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

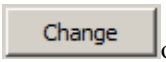
Use the ISim Simulator page in the **Preferences** dialog box. To access this page, select **Edit > Preferences**, and select **ISim Simulator** in the left pane.

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

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

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

Console text font
The window to the right of the field shows example text for the specified font.

Clicking the **Change** button 

opens a dialog box in which you can specify the font used in the Console tab.

ISim Color Preferences

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

Click **Apply** to apply changes.

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

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

Delete

Deletes the custom scheme that you have selected.

You can edit the color of this scheme

Contains the color selection for each item in the ISim main window that can be customized. Click **Apply** to apply changes.

Time Format Preferences

Use the Time Format Preferences to customize the appearance of displayed time values in the ISim GUI.

To access this page, select **Edit > Preferences >Time Format**, option in the left pane. There are two categories of time formats presented: **Waveform Window** and **Other GUI Elements**.

Waveform Window

Time format options under this category apply to the GUI elements inside the waveform viewer window. There are three time format options under this category:

Link All Waveform Time Units To Ruler

This option 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

This option applies to the main ruler at the top of the waveform window as well as to the floating ruler. Most users will need to deal with this option only.

Cursors/Markers

This option applies to the time values displayed for the all cursors and markers.

Measure Bubbles

This option applies to the cursor value bubbles displayed at the bottom of the waveform window.

Other GUI Elements

The time format options under this category apply to the GUI elements outside the Waveform window.

All Time Values

This option applies to the current simulation time shown at the bottom right of the main window and time values shown in the Objects panel.

The above 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 setting for the Other GUI elements is **Default** and for the Waveform Window it 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**.

Following button applies to the entire time format preference setting:

Reset To Defaults

This setting resets the values to the factory default settings.

XILINX®

Chapter 3

VHDL Simulation

VHDL Simulation Overview

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

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

Go to the appropriate simulation topic for details:

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

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

Method 1: Using a Project File - Recommended

Parsing the Design Files

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

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

where:

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

Example:

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

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

Generating the ISim Simulation Executable - Running fuse

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

Syntax:

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

where:

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

Example fuse command with project file:

```
fuse mylib_for_testbench.testbench -prj proj_name.prj
```

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

Method 2: Parsing the Files Using vhpcomp

Parsing the Design Files

Syntax:

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

where

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

Example:

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

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

Generation of the ISim Simulation Executable - Running fuse

Syntax:

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

where:

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

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

- **`-o`** switch lets you define the simulation executable name (for example, **`my_sim.exe`**). Without `-o`, the default name for the simulation executable is `x.exe`.

Example:

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

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

Simulation

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

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

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

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

Generating Timing Simulation Model

Before launching a timing simulation, a timing simulation model and delay file for back-annotation are required. Use the NetGen tool to generate these files. See "Generating Gate-Level Netlist (Running NetGen)" in the *Synthesis and Simulation Design Guide (UG626)*.

Method 1: Using a Project File - Recommended

File Compilation -

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

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

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

Generation of the ISim Simulation Executable - Running fuse -

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

Syntax:

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

where:

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

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

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

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

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

Method 2: Using vhpcomp

File Parsing -

Syntax:

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

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

where:

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

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

Generating the ISim Simulation Executable - Running fuse -

Syntax:

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

where:

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

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

Simulation

After compiling and generating the ISim executable, the final stage is simulation. You run the ISim simulation executable generated by the fuse command to perform simulation.

To execute Tcl commands in a file after the simulation has begun, use the **-tclbatch** switch.

It is also possible to instruct the simulator to use the correct timing delays from the Standard Delay Format (SDF) file.

Syntax:

```
<executable_name>.exe -tclbatch <tcl_file_name>
-sdfmin|-sdftyp|-sdfmax [<instance>=]<sdf file name>
```

where

- <*executable_name*>.exe is the simulation executable called x.exe unless otherwise specified with the **fuse -o** switch.
- **-sdfmin|-sdftyp|-sdfmax** is the type of delay (minimum, typical, or maximum) that ISim should use.
- <*instance*> is the hierarchical path name of the instance at which SDF back annotation needs to be done.
- <*sdf filename*> is the filename of the SDF you want to annotate.

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

Library Mapping File

Note The following information is intended for advanced users.

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

Search Order

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

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

Syntax

The xilinxsim.ini file has the following format:

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

Example

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

VHDL

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

Verilog

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

Features/Limitations

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

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

Interactive Simulation in Command Line Mode

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

XILINX®

Chapter 4

Verilog Simulation

Verilog Simulation Overview

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

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

Go to the appropriate simulation topic for details:

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

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

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

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

Method 1: Using a Project File - Recommended

Parsing the Design Files

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

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

where:

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

Example:

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

Generating the ISim Simulation Executable - Running fuse

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

Syntax:

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

where:

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

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

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

Example:

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

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

Method 2: Parsing the Files Using vlogcomp

Parsing the Design Files

Syntax:

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

where

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

Example:

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

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

Generation of the ISim Simulation Executable - Running fuse

Syntax:

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

where:

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

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

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

Example:

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

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

Simulation

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

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

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

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

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

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

Generating Timing Simulation Model

Before launching a timing simulation, a timing simulation model and delay file for back-annotation are required. Use the NetGen tool to generate these files. See "Generating Gate-Level Netlist (Running NetGen)" in the *Synthesis and Simulation Design Guide (UG626)*.

Method 1: Using a Project File - Recommended

Compilation of the Files

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

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

where:

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

Example:

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

Generation of the ISim Simulation Executable - Running fuse

The HDL linker, `fuse`:

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

Syntax:

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

where:

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

Example:

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

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

Method 2: Parsing the Files Using vlogcomp

Parsing the Files

Syntax:

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

where

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

Example:

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

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

Generation of the ISim Simulation Executable - Running fuse

Syntax:

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

where:

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

Example:

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

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

Simulation

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

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

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

Syntax:

```
<executable_name>.exe -tclbatch <tcl_file_name>
-sdfmin|-sdftyp|-sdfmax [<instance>=]<sdf file name>
```

where

- *<executable_name>.exe* is the simulation executable called `x.exe` unless otherwise specified with the **`fuse -o`** switch.
- `-sdfmin|-sdftyp|-sdfmax` is the type of delay (minimum, typical, or maximum) that ISim should use.
- *<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 you want to annotate.

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

Search Order for Instance of Verilog Design Units

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

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

Supporting Source Libraries

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

To use this feature, the following command options must be passed to the `vlogcomp` Verilog compiler.

Library Location (-sourcelibdir)

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

After the source files on the command line are compiled, if there are any unresolved references to modules, the compiler searches the source libraries for resolution. During this search, the compiler attempts to match the *name* of any unresolved instantiated design unit with a file of the *same name* in the specified **-sourcelibdir** directory. If such a file exists, the compiler analyzes that file.

Note By default, the compiler ignores any files with extensions such as `.v` or `.h`, unless **-sourcelibext** is also used.

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

Source File Extension (-sourcelibext)

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

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

```
-sourcelibext .v
```

Source File (-sourcelibfile)

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

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

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

Examples

The following examples demonstrate the use of these command options.

vlogcomp

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

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

fuse

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

where `test.prj` contains:

```
verilog work test.v
```

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

***<proj_name>* .prj**

```
fuse –prj test.prj test
```

where `test.prj` contains:

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

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

Library Mapping File

Note The following information is intended for advanced users.

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

Search Order

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

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

Syntax

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

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

Example

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

VHDL

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

Verilog

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

Features/Limitations

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

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

Predefined XILINX_SIM Macro for Verilog Simulation

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

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

Interactive Simulation in Command Line Mode

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

XILINX®

Chapter 5

Mixed Language Simulation

Mixed Language Simulation Overview

Note The following information is intended for advanced users.

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

Restrictions on Mixed Language in Simulation

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

Key Steps in a Mixed Language Simulation

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

  Use the **`fuse -L`** option to specify the binding order of a VHDL entity or a Verilog module in the design libraries of a mixed language project.

  Note The library search order specified by **`-L`** is used for binding Verilog modules to other Verilog modules as well. More Info
- Run the simulation.

Instantiating Mixed Language Components

Instantiating a Verilog Module in a VHDL Design Unit

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

To Instantiate a Verilog Module in a VHDL Design Unit

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

   For example,

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

2. Use named association to instantiate the Verilog module.

   For example,

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

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

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

Instantiating a VHDL Module in a Verilog Design Unit

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

To Instantiate a VHDL Module in a Verilog Design Unit

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

For example,

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

Mixed Language Binding and Searching

Note The following information is intended for advanced users.

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

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

VHDL Instantiation Unit

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

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

Verilog Instantiation Unit

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

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

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

Mixed Language Boundary and Mapping Rules

Note The following information is intended for advanced users.

General

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

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

Port Mapping

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

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

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

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

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

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

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

Generics (Parameters) Mapping

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

- integer
- real
- string
- boolean

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

VHDL/Verilog Values Mapping

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

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

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

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

bit	Verilog
'0'	0
'1'	1

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

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

XILINX®

Chapter 6

Waveform Analysis

Before Analysis

Launching the ISim GUI

To launch the ISim GUI in read-only mode to view or analyze the data from a previous simulation, see Opening a Static Simulation .

What to Expect

Simulation from ISE or PlanAhead software

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

Simulation at the Command Line

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

Adding Signals to the Wave Configuration

You can populate the Wave window with the signals from your design using menu commands or drag and drop capabilities in the 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.

In the GUI

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

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

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 by entering the scope command in the Console panel.
2. In the Console panel, enter the wave add command to enter an individual object or a group of objects.

Wave configurations and .wcfg files

Though 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.
- 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 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. In saving to the file, the wave configuration name is changed to match the `.wcfg` file name
- 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/bus in a wave configuration for comparing waveforms. Copies of the same signal /bus can be placed anywhere in the wave configuration, such as in groups, or virtual buses.

To Add a Copy of a Signal or Bus

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

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

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

Running a Simulation in ISim

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

To Run a Simulation From the ISim GUI

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

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

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

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

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 close the session of ISim, see Closing ISim.

To Pause a Running Simulation

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

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

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

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

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

Closing ISim

You can terminate a simulation and close the ISim session.

To Close ISim

- Select **File > Exit**.
- Enter the `quit -f` command in the Console panel at the prompt. This will prevent an "are you sure" dialog box from opening.
- Click the **X** at the top-right corner of the main window.

The simulation terminates and the session of ISim closes.

Customizing the Wave Configuration

Placing Cursors

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

Note Cursors are used primarily for temporary indicators of time and are expected to be moved frequently, such as when 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 Adding a Marker for more information.

To Place the Main Cursor

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

To Place the Secondary Cursor

Place the secondary cursor as follows:

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

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

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

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

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

To Move a Cursor

To move a cursor hovering 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 toolbar button is selected, which is the default behavior.

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

To Hide the Secondary Cursor

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

Setting Markers

Adding a Marker

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

To Add a Marker

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

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

Moving a Marker

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

To Move a Marker

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

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

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

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

The marker is moved to the new location.

Deleting a Marker

You can delete one or all markers with one command.

To Delete Markers

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

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

The marker or markers are removed from the waveform.

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

Adding Dividers

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

To Add a Divider

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

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

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

Following changes can be made to a divider:

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

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

Adding Groups

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

To Add a Group

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

   Note A group can also comprise dividers, virtual buses, and other groups.

2. Select **Edit > New Group**, or right-click and select **New Group** from the context menu.

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

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

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

Following changes can be made to a group:

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

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

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

Adding Virtual Buses

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

To Add a Virtual Bus

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

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

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

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

Following changes can be made to a virtual bus:

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

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

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

Renaming Objects

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

To Rename an Object

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

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

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

Changing the Display Names

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

If the name is hidden:

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

To Change the Display Name

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

The name changes immediately according to your selection.

Changing the Radix

To Set the Default Radix

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

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

To Change an Individual Radix

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

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

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

Reversing Bus Bit Order

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

To Reverse a Bus' Bit Order

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

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

Navigating the Wave Configuration

Expanding and Collapsing a Hierarchy

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

Clicking the Arrow

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

 - Click the arrow to collapse the hierarchy.

Menu Command

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

Using the Context Menu

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

Zooming In and Out

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

Using Commands

Zooming Behavior	Command	Shortcut Key
Zoom in	Select **View > Zoom > In**, or click the Zoom In toolbar button.	• Press **F8**. • Press **Ctrl**, and in the wave configuration, draw a line *down* to the *left*. **Note** The middle-mouse button can be used in place of the Ctrl key.
Zoom out	Select **View > Zoom > Out**, or click the Zoom Out toolbar button.	• Press **F7**. • Press **Ctrl**, and in the wave configuration, draw a line *up* and to the *right*. **Note** The middle-mouse button can be used in place of the Ctrl key.
Zoom to display the entire waveform in the window	Select **View > Zoom > To Full View**, or click the Zoom to Full toolbar button.	• Press **F6**. • Press **Ctrl**, and in the wave configuration, draw a line *up* and to the *left*. **Note** The middle-mouse button can be used in place of the Ctrl key.
Zoom to an area you select	Not applicable.	Press **Ctrl**, and draw a box around an area of the wave configuration from the top left corner of the area to the bottom right. **Note** The middle-mouse button can be used in place of the Ctrl key.
Zoom to display the time range between the two cursors	1. Place the cursors in the waveform. 2. Select **View > Zoom > To Cursors**, or click the Zoom to Selected toolbar button.	Not applicable.

Using the Middle Mouse Wheel

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

Displaying the Floating Ruler

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

To Display the Floating Ruler

1. Either:
   - Place the secondary cursor.

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

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

The floating ruler displays.

Select the command again to hide the floating ruler.

Displaying Waveform Values With Markers

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

To Display Waveform Values With Markers

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

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

   For multiple markers, continue.
2. Select **Edit > Markers > Next Marker**, or click the Next Marker toolbar button  .

   The cursor advances through markers in the wave configuration.
3. Observe the values in the Value column at each marker.
4. Select **Edit > Markers > Previous Marker**, or click the Previous Marker toolbar button  .

   The cursor moves backward through markers in the wave configuration.
5. Observe the values in the Value column at each marker.

Displaying Waveform Values at Signal Transitions

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

To Use Next and Previous Transition Commands

1. Select a signal.

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

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

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

3. Repeat step 2 as necessary.

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

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

5. Repeat step 4 as necessary.

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

Measuring Time with Cursors

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

To Measure Time With Cursors

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

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

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

   These actions place the secondary cursor on the first transition, and the main cursor on the second transition.

4. Examine the values at the bottom of the wave configuration:
   - X1 represents the time of the main cursor.
   - X2 represents the time of the secondary cursor.
   - Delta X represents the time range between the cursors.

   If the floating ruler is displayed, the time values of the cursors display just above the floating ruler.

5. *Optional*. You can switch the cursors using the Swap Cursors icon .

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

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

Measuring Time with Markers

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

To Measure Time with Markers

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

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

   The time base can be switched quickly from one marker to another 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 enables you to jump the main cursor to a particular time in the wave configuration. Two Go To Time toolbar button also enables you to jump to time 0 (zero) and the latest simulation time. All signal and bus values are updated for the location of the main cursor.

To Go To a User-Specified Time

With a wave configuration open:

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

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

To Go to Time 0 or Latest Time

With a wave configuration open:

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

Using Show Drivers

You can use the Show Driver command to display the driver for a change in signal, or object value. This command is used to determine the cause of a value change, which 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 (`show driver` command)

To Show Drivers

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

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

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

Printing Wave Configurations

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

To Print Preview

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

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

To Print

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

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

The wave configuration prints according to the print settings.

Using Custom Colors

Creating Custom Color Schemes

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

To Create a Custom Color Scheme

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

The custom waveform scheme takes effect immediately.

Changing Signal Colors

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

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

To Change a Signal Color

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

The signal or bus waveform displays the new color selection.

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

XILINX®

Chapter 7

Saving and Opening Simulation Results

Saving the Results

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.

Saving a Database to a .wdb File

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

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

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

Saving a Wave Configuration to a WCFG File

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

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

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

Relationship Between .wdb and .wcfg

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

Opening a Live Simulation

A live simulation consists of the following:

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

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

Opening a Waveform Database

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

Opening a Wave Configuration

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

To open a wave configuration in the ISim GUI:

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

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

To open a wave configuration from the command prompt:

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

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

where

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

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

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

Opening a Static Simulation

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

Opening a Wave Configuration and Waveform Database

Opening an Existing WCFG With Associated Waveform Database

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

In a command prompt, either:

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

Or:

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

   where:

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

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

Opening an Existing WCFG and Non-Associated Waveform Database

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

In a command prompt:

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

   where:

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

Opening a Waveform Database and a New Default .wcfg

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

In a command prompt:

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

   where:

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

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

Opening a Waveform Database Only

If you would like to open a .wdb from a previous simulation and no .wcfg, use this method:

In a command prompt, either:

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

Or:

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

   where:

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

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

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

XILINX®

Chapter 8

Debugging

Source Level Debugging Overview

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

Some strategies available for debugging in ISim are:

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

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

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

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

Stepping

Stepping Through a Simulation

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

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

To Step Through a Simulation

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

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

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

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

Using Breakpoints

Setting Breakpoints

In ISim, 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 to Debug Your Design.

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

Setting and Removing Breakpoints

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

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

To remove a breakpoint, click the breakpoint to remove it.

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

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

A list of breakpoints is available in the Breakpoints panel.

Debugging Your Design Using Breakpoints

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

To Use Breakpoints for Design Debugging

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

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

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

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

Deleting Breakpoints

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

To Remove a Breakpoint

Use one of the following methods:

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

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

The single breakpoint is deleted.

To Remove All Breakpoints

Use one of the following methods:

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

All breakpoints in all HDL files are deleted.

XILINX®

Chapter 9

Writing Activity Data for Power Consumption

Writing Activity Data of the Design

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

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

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

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

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

Activity File Types

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

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

To Write Activity Files for Power

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

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

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

XILINX®

Chapter 10

Using Tcl Simulation Commands

Simulation Command Overview

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

Simulation Command Entry Methods

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

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

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

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

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

Simulation Command Summary

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

Note These commands are case sensitive.

- bp: Sets and deletes breakpoints in your HDL source code for debugging purposes.
- describe: Displays information about the given HDL data or block object.
- divider add: Adds a new divider.
- dump: Displays a list of variables, generics, parameters and nets along with their values for the current scope of the design hierarchy.
- group add: Adds a new group.
- help: Displays a description with usage and syntax of the specified ISim command.
- isim condition: Enables the execution of a set of commands based on a specified condition on one or more nets or Verilog regs.
- isim force: Forces or removes a value on a VHDL signal, Verilog wire, or Verilog reg.
- isim get arraydisplaylength: Displays the limit of numbers of elements for an array type HDL object.

- isim get radix: Gets the global radix being used.
- isim get userunit: Displays the current unit of measurement for all time values where unit is unspecified.
- isim ltrace: Turns on and off line tracing.
- isim ptrace: Turns on and off tracing of execution of processes.
- isim set arraydisplaylength: Sets the limit on the number of elements for an array type HDL object.
- isim set radix: Sets the global radix to use.
- isim set userunit: Sets the default unit of measurement for all time values where unit is unspecified.
- marker add: Adds a new marker.
- onerror: For batch mode, controls the behavior immediately following a failed Tcl simulation command.
- put: Assigns a value to a specified bit, slice or all of a variable or signal.
- quit: Exits either the simulation or the software, depending on the command options.
- restart: Restarts simulation. It sets simulation time back to 0.
- resume: With the **`onerror`** command, continues executing commands after an error is encountered.
- run: Starts simulation.
- saif: Creates a Switching Activity Interchange format (SAIF) file and records estimate power usage.
- scope: Navigates the design hierarchy.
- sdfanno: Back-annotates delays from an SDF file to the HDL design.
- show: Displays selected aspects of the design in the Simulation Console tab.
- step: Executes simulation through your HDL design, line by line, to assist with debug.
- test: Compares the actual value of a net or bus with a supplied value.
- vcd: Generates simulation results in VCD format.
- virtualbus add: Adds a new virtual bus.
- wcfg new: Creates a new wave configuration.
- wcfg open: Opens a wave configuration from a file.
- wcfg save: Saves a wave configuration.
- wcfg select: Selects the wave configuration file to be displayed in the active window.
- wave log: Logs simulation output of HDL objects to a waveform database.
- wave add: Adds simulation objects or blocks to the specified wave configuration in the ISim graphical user interface.

Entering Simulation Commands

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

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

To Enter Commands in GUI Mode

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

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

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

To Enter Commands in Interactive Command Line Mode

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

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

   A Tcl prompt appears:

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

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

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

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

Example of a typical sequence.

1. **`scope`** - Shows current scope in design hierarchy.
2. **`dump`** - Shows you current value of signals in current scope.
3. **`show value clk`** - Shows the value of a signal named clk.
4. **`run 1000 ns`** - Runs simulation for 1000 ns.
5. **`restart`** - Resets simulation to time 0.
6. **`wave log /`** - Log simulation output of all VHDL signal, Verilog wire and Verilog reg at top level to the waveform database (wdb) file.
7. **`run 1000 ns`** - Runs simulation for 1000 ns.
8. **`wave log /tb/UUT/clk`** - Log simulation output of /tb/UUT/clk to waveform database (wdb) file starting at current simulation time (i.e. 1000 ns).
9. **`run 1000 ns`** - Runs simulation for an additional 1000 ns.
10. **`quit`** - Quits simulation.

    After you have quit simulation, you will see a file named **`isim.wdb`** in your working directory. The **`isim.wdb`** is the waveform database file containing simulation output of the signals that were logged during simulation. You can run **`isimgui -view isim.wdb`** to open and view the transitions in a waveform viewer.

To Enter Commands in Non-Interactive Batch Mode

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

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

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

   a. Open Project Navigator.

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

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

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

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

   ```
   wave add /
   run 1000 ns
   ```

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

   `stopwatchsim.exe -tclbatch isim.tcl`

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

To Use 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]}`** because square brackets have special meaning to Tcl. 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: **`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 e character and the closing curly brace.

Note Although Verilog allows any identifier to be escaped, on the ISim command line Verilog identifiers that are not required to be escaped must not 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 will not work, as 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 backslash next to a close curly brace (\}) as being simply 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, as well as 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*.

To Set a Variable at the Simulation Console

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

```
set svc "show value count"
```

where,

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

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

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

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 on the window tab or using "wcfg select" command. In the GUI, **File** > **New** and **File** > **Open** change the active window to the newly created waveform configuration window. In Tcl, "wcfg new" and "wcfg open" commands change the active window to the newly created window just like **File** > **New** and **File** > **Open** do.

ISim Wave Viewer Command Groups

The ISim Wave Viewer Tcl commands are divided into three groups:

- **Wave Configuration I/O Commands -** Used to create, save, and select a wave configuration.
- **Wave Configuration Commands -** Used for editing wave configuration.
- **Marker Commands -** Used to add new markers.

Command Line Conventions

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

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

Syntax	Description
( )	When an option occurs with parentheses "( )" next to the command name in the syntax, it is required.
[ ]	When an option appears in square brackets "[ ]," it is optional. **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, `set var <show time>` will set `var` to current time.
|	If the options are separated by a vertical bar |, you must choose one of the possible parameters. If one term is divided into a subset of parameters that can be entered separately or together, each sub parameter occurs between square brackets.
...	When an option has … it means that one or more occurrences of the option separated by space is accepted.
{ }	When an option has { } around it, it is treated as one single argument even though it has embedded spaces inside it. This allows passing of an argument with special characters including spaces to a command. For example: `set var {I have spaces}` sets var to "`I have spaces`."
< >	Less than and greater than symbols "< >" enclose variables for which you must supply values.

Note *Italics* are used to indicate variables.

Tcl Commands

Engine Commands

bp Command

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

Note

Syntax

bp (*options*)

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

Examples

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

Set a Breakpoint

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

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

List all Breakpoints

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

```
bp list
```

Delete Breakpoints

This example deletes all breakpoints in your simulation.

```
bp clear
```

This example deletes a breakpoint by index number.

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

   ```
   bp list
   ```

   The simulator returns the following:

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

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

   ```
   bp del 1
   ```

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

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

describe Command

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

Note

Syntax

`describe` *<object_name>*

Options

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

Examples

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

```
ISim> describe param
```

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

dump Command

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

Note

Syntax

dump

Example

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

dump

Obsolete Command Options and Replacements

Note The following options are obsolete:

Obsolete Tcl Command	Command to Use
dump -p	**show child**
dump -p<*process_name*>	**scope** <*process_name*>**dump**

help Command

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

Note

Syntax

help [*command_name*]

Options

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

Examples

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

Help for All Commands

For a description of all of ISim commands:

help

Help for One Command

For a description of the bp command:

help bp

isim condition Command

The **isim condition** command adds, removes, or generates a list of conditional actions. A conditional action is equivalent to a VHDL process or a Verilog always process. When added, it 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.

Note

Syntax

isim condition (**add|remove|list**) [*<condition expression>* *<command>*] [*<radix_type>*] [*<label_name>*] [*<index_name>*]

Options

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

Examples

ISim condition add Examples

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

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

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

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

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

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

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

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

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

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

ISim condition remove Examples

To remove all conditions in the current simulation:

```
isim condition remove
```

or

```
isim condition remove -all
```

To remove the conditions label0, label1 and label2:

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

To remove a single statement:

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

or

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

ISim condition list Examples

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

```
isim condition list
```

isim force Command

The **isim force** command 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.

Note

Syntax

isim force (**add|remove**) < *object_name* > < *value* > [*options*]

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 **isim set radix** command is used, or if none is set there, **default** is used as radix.
-time <*time*>	Time can be a string such as 10, 10 ns, "10 ns". When a number entered without a unit, the simulator resolution unit is used, which is ps. Time is relative to the time of execution of the command.
-cancel <*time*>	Cancels the force command after the specified time.
-repeat <*time*>	Repeats the cycle after the specified time.

Examples

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

To Assign a Value

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

```
isim force rst 0
```

To 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
```

To 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, and set `data_in` back to 1 at current simulation time + 75 ns and then 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
```

To Remove a Value

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

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

ism get arraydisplaylength Command

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.

Note

Syntax

```
isim get arraydisplaylength
```

where *no* options are available.

Example

Enter

```
isim get arraydisplaylength
```

Returns

```
64
```

isim get radix Command

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

Note

Syntax

```
isim get radix
```

where *no* options are available.

Example

To return the radix:

```
isim get radix
```

Returns the current global radix.

isim get userunit Command

The **`isim get userunit`** command 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.

Note

Syntax

```
isim get userunit
```

where *no* options are available.

Example

Enter

```
isim get userunit
```

Returns

```
1 ps
```

isim ltrace Command

The **`isim ltrace`** command let 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, file path, filename, and line number.

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

Note

Syntax

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

Options

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

Example

To see which line is currently executing.

```
isim ltrace on
run
```

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

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

isim ptrace Command

The **`isim ptrace`** command 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 tab. 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.

Note

Syntax

`isim ptrace [on | off]`

Options

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

Example

To see which process is currently executing.

`isim ptrace on`

isim set arraydisplaylength Command

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

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

The default is 64.

Note

Syntax

`isim set arraydisplaylength` *<size>*

Options

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

Examples

Enter

```
isim set arraydisplaylength 2
show value xcountout
```

Returns

```
00
00
```

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

Enter

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

Returns

```
0001000000
0001000000
```

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

isim set radix Command

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

Note

Syntax

`isim set radix` *<radix_type>*

Options

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

Examples

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

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

Returns `a`.

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

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

Returns `-4`.

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

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

Returns `10`.

isim set userunit Command

The **`isim set userunit`** command 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 1 ps.
- As defined by the `'timescale` simulator directive in Verilog.

Note

Syntax

`isim set userunit` (*options*)

Options

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

Example

Set the simulator timescale to 1ps:

```
isim set userunit 1 us
```

onerror Command

The **onerror** command lets you control the behavior immediately following a failed Tools Command Language (Tcl) simulation command. See examples for various applications, such as printing the error and resuming the next command.

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

Note

Syntax

onerror(*options*)

Options

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

Examples

In this example, ISim exits after printing the time that the error occurred and all values in the current scope.

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

In this example, ISim continues reading the next command in the Tcl script after showing the time the error occurred and all values in the current scope.

```
onerror { show time;dump;resume }
```

In this example, ISim reads the sourced Tcl file and executes its command upon encountering an error.

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

put Command

The **put** command enables you to modify values of signals or buses during simulation. The **put** command can be used to assign:

- a specified signal or bus
- an array of signals/buses
- a record or array of records containing signals/buses
- a value with the specified radix is put to an object

To use the **put** command, the signal or bus must be declared as a signal in your 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.

Note

Syntax

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

Options

Option	Description
`<signal_name|vhdl_process_name/process_variable_name> [element reference, element reference, ...] <value>`	*<signal name>* refers to the name of the signal or bus to which you want to assign a value. This can also be the name of an array of signals/buses, an array of records containing signals and buses, or a record of arrays of buses/signals. *<vhdl_process_name/process_variable_name>* is the name of the process and process variable to assign a value. To assign a value to a process variable, you must also supply the name of the process that contains the variable. The process name and process variable names must be separated by a slash symbol (/). [*element reference*] refers to the different ways in which the sub-elements of the signal name can be referenced when used with the **put** command. This provides the ability to further refine the signals by referencing the individual sub-elements of a signal. See the following examples for more information. The accepted *value* depends upon the signal type as follows. • Integer type can be a positive or negative integer. • A bit_vector can be 0 or 1 or a series of 0s and 1s. • For VHDL, `std_logic` values can be U, X, 0, 1,... • For Verilog, `bit_values` can be U, X, 0, 1.

	• Strength values are not supported. <*value*> refers to the value assigned.
<*object*> <*value*> [**-radix** <*radix_type*>]	Takes a value with the specified radix, converts the value to the data type of the object, and writes the value to the object. <*object*> specifies the signals, buses or objects to modify. <*value*>refers to the value you wish to add to the object. 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.

Examples

Assign a Value To a Bus or Signal

To assign a value to a signal called `clk`:

```
put clk 1
```

or

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

or

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

To assign a value to a 4-bit bus called `busx`:

```
put busx 0101
```

To assign a value FF to signal A:

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

To assign a bit value of 1 to a signal called `count(6)` in the module `u1` that is instantiated under your current scope:

```
put u1/count(6) 1
```

Assign a Value To a Standard Logic Vector

For a standard logic vector called `sigx` that is declared as follows:

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

To set bit 0 of `sigx` to 1:

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

To set slice 1 to 2 of `sigx` to 11:

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

To set all of `sigx` to 101010:

```
put sigx 101010
```

Assign a Value To an Array of Objects

For an array of standard logic vectors that is declared as follows:

```
signal sigarray :  (0 to 3) vectorarray(0 to 5, 1 to 4, 2 to 6);
```

To set every bit of the array element of `sigarray` to 1:

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

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

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

To set bits three of the array element of `sigarray` to 1:

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

For an array of records which in turn contain an array of standard logic vectors that is declared as follows:

```
type ram_3d_vector is array(0 to 10, 7 downto 0, 0 to
2) of std_logic_vector(1 to 4);

type rectype is record
a: integer;
b: string(1 to 7);
c: std_logic_vector(0 to 3);
d: ram_3d_vector;
end record;

type recarray is array(0 to 3, 4 downto 1) of rectype;

signal recarrsig : recarray;
signal recsig : rectype;
```

To set the second element (*b*) of the record *recsig* to the string `abc`:

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

To set the four bit wide vector represented by the coordinates 2,3,1 in the three dimensional array *d* in the record `recsig` to 0110:

```
put recsig.d(2,3,1) 0110
```

To set the first two bits of the four bit wide vector represented by the coordinates 2,3,1 in the three dimensional array *d* in the record `recsig` to 01:

```
put recsig.d(2,3,1)(1:2) 01
```

To set the four bit wide vector represented by the coordinates 2,3,1 in the three dimensional array *d* in the record `recsig` represented by the coordinates 2,2 in the two dimensional array `recarrsig` to 0011:

```
put recarrsig(2,2).d(2,3,1)0011
```

quit Command

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

Note

Syntax

`quit` [*`options`*]

Options

`-f`	Quits the current simulation and quits the ISim software. You are not prompted to save the wave configuration even when changes have been made to wave configurations. The switch has no effect at the command line.
`-s`	Quits the current simulation while keeping the graphical user interface open. At this point, ISim can only be used for loading static wave databases via **File > Open** to open a .wdb file. The switch has no effect at the command line.

Examples

To use this command to exit ISim and leave the Tcl prompt, type:

```
quit
```

To use this command to exit ISim and save your waveform:

```
quit -f
```

To use this command to quit the current simulation:

```
quit -s
```

restart Command

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.

Specifically, **`restart`** resets the following command settings:

- onerror — Removes onerror script.
- scope — Resets scope to `/top`.
- saif — Closes file.
- vcd — Closes file.
- isim force — Removes force.
- put — Returns to initial value.

Note

Syntax

```
restart
```

Example

To use this command to set simulation time back to 0, and start simulation:

```
restart
```

resume Command

The **`resume`** command is used with the **`onerror`** command to continue executing commands after an error is encountered.

Note This command has no effect when entered alone.

Note

Syntax

`resume`

where *no* options are available.

Example

In this example, ISim continues reading the next command in the Tools Command Language (Tcl) script after showing the time the error occurred and all values in the current scope.

```
onerror { show time;dump;resume }
```

run Command

The **`run`** command starts simulation. With no options, the **`run`** command runs simulation for 100 ns. The equivalent GUI commands are Simulation > Run All and Simulation > Run.

Note

Syntax

`run` [*`options`*]

Options

Option	Description
`all`	Runs simulation until there are no more events or until ISim encounters a breakpoint. See bp Command for information about setting and removing breakpoints.
`continue`	Resumes simulation after ISim has stopped at a breakpoint. This option is the same as running **`run all`**.
`<time> <unit>`	*Time* is the length of time you want simulation to run. It can be any positive integer or decimal. *Unit* is the unit of time. Possible values are **`fs`**, **`ps`**, **`ns`**, **`us`**, **`ms`** and **`sec`**. Default is **`ps`**.

Examples

The **`run`** command can be used as follows.

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

```
run all
```

To run simulation for 2000 nanoseconds:

```
run 2000 ns
```

To run simulation for 1.2 nanoseconds:

```
run 1.2 ns
```

To run simulation for 100 ns.

```
run
```

saif Command

The **saif** command lets you create a Switching Activity Interchange format (SAIF) file and record port and signal switching rates. See also Writing Activity Data of the Design.

Note

Syntax

saif *<options>*

Options

Option	Description
open [**-scope** *<path_name>*] [**-file** *<file_name>*] [**-allnets**]	**open** creates a file for SAIF power estimation. **-scope** *<path_name>* 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. **-file** *<file_name>*creates a new SAIF file. The default name of the SAIF file is `xpower.saif`. Only one SAIF file is allowed to be opened during one run of simulation. **-allnets** includes internal nets and port signals in the power estimation. Without this switch, only port signal changes are monitored.
close	Stops monitoring and flushes the SAIF file.
-level *<number_of_levels>*	When **-level 0** is used, signal transitions in all levels below the specified hierarchy are tracked. When **-level 1** is used, only the signal transitions in the specified hierarchy. When **-level 2** is used, signals in two levels of hierarchy are tracked and dumped.

Examples

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

In this example, all ports of a design starting at the current scope recursively are written to a file called `xpower.saif`.

```
saif open
```

In this example, the ports and internal nets of a design starting at the current scope recursively are written to a file called `xpower.saif`.

```
saif open -allnets
```

In this example, ports and internal nets of a design starting at `uut` recursively are written to a file called `uut_backward.saif`.

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

scope Command

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

Note

Syntax

scope [*options*]

Options

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

Examples

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

To move up one level in the design hierarchy:

```
scope ..
```

To move to the module UUT that is instantiated in the current module:

```
scope UUT
```

To use the scope Tools Command Language (Tcl) command on child instances in a post route netlist.

For example:

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

where, \CLK/PAD is an extended identifier.

Enter:

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

Notice that you must use a backslash (\) before \CLK and a backslash after PAD at the end.

sdfanno Command

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.

Note

Syntax

sdfanno (**-min** | **-typ** | **-max**) <*file_name*> [*options*]

where **-min | -typ | -max** and *file_name* are required, and *options* is optional.

Options

Option	Description
(-min | -typ | -max)	A delay switch type (**-min**, **-typ** or **-max**) must be specified in the **sdfanno** command. • **-min** - Annotates VHDL/Verilog file with fastest possible delay values. Gives you a Hold Time timing simulation. • **-typ** - Annotates VHDL/Verilog file with typical delay values. • **-max** - Annotates VHDL/Verilog file with longest possible delay values. Gives you a Setup Time timing simulation.
<file_name>	Name of the SDF file that contains the delay information. A filename must be specified in the **sdfanno** command.
-nowarn	Turns off warning messages.
-noerror	Turns error messages into warning messages. This allows you to continue simulation even though there are errors in SDF back-annotation.
-root *<root_path>*	Specifies the place in the design to apply annotation. The paths specified in the SDF file are relative to the place in the design hierarchy specified by *root_path*. By default, the root is at the top level of the design.

Examples

sdfanno -typ Examples

To annotate the submodule "subdesign" with the typical delay values from `mysubdesign.sdf`:

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

To annotate the current top-level design with the typical delay values from `design.sdf` and ignore all errors or warnings:

```
sdfanno -typ design.sdf -noerror -nowarn
```

sdfanno -min Example

To annotate the submodule "subdesign" with the minimum delay values from `mysubdesign.sdf`:

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

sdfanno -max Example

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

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

show Command

The **show** command displays selected aspects of the design.

Note

Syntax

show (*options*)

Options

Option	Description
child | **child -r**	Without the **-r** option, **child** displays the children blocks (one level only) from the current block. **child -r** option recursively lists all children blocks, including the child processes from the current block.
constant	Lists constants, generics and parameters within the current block.
driver	Displays the processes that are driving the signal specified by *signal_name*. If possible, it prints the line numbers of the HDL code that is contributing to the driver.
load	Displays all of the load for the signal specified by *signal_name*.
port	Displays 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*> | <*signal_name*> [*element reference, element reference, ...*] | <*object*> [**-radix** <*radix_type*>]	<*generic_name*> is the name of the VHDL generic to be queried. <*parameter_name*> is the name of the VHDL parameter to be queried. <*process_name/process_variable_name*> is the name of the process and process variable to be queried. To query the value of a process variable, you must also supply the name of the process that contains the variable. The process name and process variable names must be separated by "/". <*signal_name*> - Name of the signal or bus to be queried. This can also be the name of an array of signals/buses, an array of records containing signals and buses or a record of arrays of buses/signals. • Use a dot (.) to separate the record hierarchy, as in:

	show value recsig.c • Use a slash (/) to delimit the hierarchical name of the signal to query a value to a signal on a lower level of hierarchy, as in: **show value mymod/mysig** [*element reference*] refers to the different ways in which the sub-elements of the signal name can be referenced when used with the **show** command. This allows further refinement to the signals by referencing the individual sub-elements of a signal. Please see the following examples for more information. • two integers separated by a colon, enclosed in parenthesis, displays a range of values in the vector specified by *signal_name*. For example: **show value(3:0)** • one or more integers separated by commas, enclosed by parenthesis, displays the values of elements in a multidimensional array specified by *signal_name*. For example: **show value(2,3)** <*object*> [**-radix** <*radix_type*>] displays values with the specified radix. For <*object*>, set the HDL object data type. The supported radix types are **default**, **dec**, **bin**, **oct**, **hex**, **unsigned** and **ascii**. When no radix type is specified, the global radix type set with the **isim set radix command** is used, or if none is set there, **default** is used as radix.
variable	Displays all of the variables within the current block. To see variables within a VHDL process, use **scope** to navigate to the VHDL process and then run **show variable**.

Examples

Show Child

If you are located at the top level of the hierarchy in a design called `fifo_controller` and enter **show child**. The following hierarchy information displays:

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

Enter **show child -r** for the same level of the design, the current and recursive hierarchy information displays.

Show Driver

If you are located at the top level of the hierarchy in a design called `fifo_count` and enter **show driver fifocount**. The following information displays for the signal `fifocount`:

```
<Driver for fifocount>
   fifoctlr_cc_v2.v:221
```

The number 221 at the end of the lines refers to the code line in the source file.

Show Load

If you are located at the top level of the hierarchy in a design called `fifo_count` and enter **`show load fifocount`**. The following information displays for the signal `fifocount`:

```
<Load for fifocount>
Signal <Hex(0)> (Block: fifo_count/Lsbled/)
Signal <Hex(1)> (Block: fifo_count/Lsbled/)
Signal <Hex(2)> (Block: fifo_count/Lsbled/)
Signal <Hex(3)> (Block: fifo_count/Lsbled/)
```

Show Scope

If you are located at the top level of the hierarchy in a design called `fifo_count` and enter **`show scope`**. The following information displays:

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

Show Signal Value

To query a value to a signal called *clk*:

```
show value clk
```

To query a value to a 4-bit bus called *busx*:

```
show value busx
```

To query the value of *addr*:

```
show value addr
```

Displays `0111010101011101`.

```
show value addr -radix hex
```

Displays `755D`.

```
show value addr -radix dec
```

Displays `30045`.

Show Object Value

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

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

To query bit 0 of `sigx` :

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

To query slice 1 to 2 of `sigx`:

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

To query all of `sigx`:

```
show value sigx
```

Show Value of Array of Objects

The following examples refer to an array of standard logic vectors that is declared as follows:

```
signal sigarray :  vectorarray(0 to 5, 1 to 4, 2 to 6);
```

To query every bit of a vector array element of `sigarray`:

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

To query the first two bits of each array element of `sigarray`:

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

To query bits three of each array element of `sigarray`:

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

For an array of records which in turn contains an array of standard logic vectors that is declared as follows:

- `type ram_3d_vector is array(0 to 10, 7 downto 0, 0 to 2) of std_logic_vector(1 to 4);`
- `type rectype is record`
- `a:  integer;`
- `b:  string(1 to 7);`
- `c:  std_logic_vector(0 to 3);`
- `d:  ram_3d_vector;`
- `end record;`
- `type recarray is array(0 to 3, 4 downto 1) of rectype;`
- `signal recarrsig :  recarray;`
- `signal recsig :  rectype;`

To query the second element (`b`) of the record `recsig`:

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

To query the four bit wide vector represented by the coordinates 2,3,1 in the three dimensional array `d` in the record `recsig`:

```
show value recsig.d(2,3,1)
```

To query the first two bits of the four bit wide vector represented by the coordinates 2,3,1 in the three dimensional array `d` in the record `recsig`:

```
show value recsig.d(2,3,1)(1:2)
```

To query the four bit wide vector represented by the coordinates 2,3,1 in the three dimensional array `d` in the record `recsig` represented by the coordinates 2,2 in the two dimensional array `recarrsig`:

```
show value recarrsig(2,2).d(2,3,1)
```

step Command

After you run an initial simulation, you can step through your HDL design one line of source code at a time to verify that the design is working as expected. The **`step`** command advances to the next line of executable code in the Verilog or VHDL file. The equivalent GUI command is Simulation > Step.

Note

Syntax

`step`

where *no* options are available.

Example

To step through one line of HDL source code:

```
step
```

test Command

The **test** command compares the actual value of a VHDL signal, Verilog wire, Verilog reg, VHDL generic, Verilog parameter or VHDL process variable in the current scope with a supplied value. If the two values match, nothing is displayed. Otherwise the current correct value is displayed, and ISim reports an error. You can test one bit, a slice of a vector element or a whole value.

Note

Syntax

```
test <signal_name|vhdl_process_name/process_variable_name >
{element reference, element reference, ...} <value> | <object>
<value> -radix <radix_type>
```

Options

Option	Description
<signal_name | vhdl_process_name/process_variable_name> {*element reference, element reference, ...*}*<value>*	*<signal_name>* refers to the name of the signal or bus to be compared. This can also be the name of an array of signals/buses, an array of records containing signals and buses or a record of arrays of buses/signals. *<vhdl_process_name/process_variable_name>* is the name of the process and process variable to be compared. To compare the value of a process variable, you must also supply the name of the process that contains the variable. The process name and process variable names must be separated by "/". {*element reference*} refers to the different ways in which the sub-elements of the signal name can be referenced when used with the **test** command. This provides the ability to further refine the signals by referencing the individual sub-elements of a signal. Please see Examples below for more information. *<value>* refers to the supplied value to compare with actual value on net or bus.
<object> <value> **-radix** *<radix_type>*	Compares the specified value with the specified radix to object value. *<object>* specifies the signals, buses or objects to test. *<value>* refers to the value you wish to add to the object. The supported radix types are **default**, **dec**, **bin**, **oct**, **hex**, **unsigned** and **ascii**. When no radix type is specified, the global radix type set with the isim set radix command is used, or if none is set there, **default** is used as radix.

Examples

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

To compare a bit values of 1 to a signal called `count(6)` in the module `u1` that is instantiated under your current scope:

```
test u1/count(6) 1
```

To compare the value of signal A to FF:

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

To compare value with "clk" value:

```
test clk 'U'
```

Returns `1`.

To compare value with clk value:

```
test clk '0'
```

Returns `0`.

To stop simulation if /top/rst is 0.

```
if {[test /top/rst 0 ] } {stop } else ...
```

For the signal Reset in Block Uut, the command **test Reset 1** displays the following message:

test failed Command failed: test Reset 1 1 Net Reset has value 0 not 1 as expected.

For the bus Lsbcnt in Uut,
the command **test Lsbcnt 1001** displays the following message:
test passed 0

For a standard logic vector called `sigx` and declared as follows:

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

To compare bit 0 of `sigx` to 1:

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

To compare slice 1 to 2 of `sigx` to 11:

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

To compare all of `sigx` to 101010:

```
test sigx 101010
```

For an array of standard logic vectors that is declared as follows:

```
signal sigarray :  vectorarray(0 to 5, 1 to 4, 2 to 6);
```

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

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

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

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

To compare bits three of each array element of `sigarray` to 1:

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

For an array of records that contains an array of standard logic vectors and is declared as follows:

```
type ram_3d_vector is array(0 to 10, 7 downto 0, 0 to
2) of std_logic_vector(1 to 4);

type rectype is record
a: integer;
b: string(1 to 7);
c: std_logic_vector(0 to 3);
d: ram_3d_vector;
end record;

type recarray is array(0 to 3, 4 downto 1) of rectype;

signal recarrsig : recarray;

signal recsig : rectype;
```

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

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

To compare the four bit wide vector represented by the coordinates 2,3,1 in the three dimensional array `d` in the record `recsig` to 0110:

```
test recsig.d(2,3,1) 0110
```

To compare the first two bits of the four bit wide vector represented by the coordinates 2,3,1 in the three dimensional array `d` in the record `recsig` to 01:

```
test recsig.d(2,3,1)(1:2) 01
```

To compare the four bit wide vector represented by the coordinates 2,3,1 in the three dimensional array `d` in the record `recsig` represented by the coordinates 2,2 in the two dimensional array `recarrsig` to 0011:

```
test recarrsig(2,2).d(2,3,1)0011
```

vcd Command

The **`vcd`** command generates simulation results in Value Change Dump (VCD) format. This command enables you to dump specified instances to a VCD file, to name the VCD file, to start and stop the dump process, and other functions. See also Writing Activity Data of the Design.

Note

Syntax

`vcd` (*`options`*)

Options

Option	Description
dumpfile <*file_name*>	Gives a name to VCD file. Default file name is `dump.vcd`. Invokes Verilog `$dumpfile` directive.
dumpvars -m <*module_name*> [**-l** <*level*>]	Dumps the specified variables and their values to the VCD file. **-m** <*module_name*>dumps the module of that name. **-l** <*level*> **0** - Level 0 causes a dump of all variables in the specified module and in all module instances below the specified module. The argument 0 applies only to subsequent arguments which specify module instances, and no to individual variables. **1** -Level 1 dumps all variable within the module specified by -m; it does not dump variables in any of the modules instantiated by the module specified by -m. Invokes Verilog `$dumpfile` directive.
dumpoff	Temporarily suspends the dumping process, and dumps all selected variables as an 'X' value. Invokes Verilog `$dumpoff` directive.
dumpon	Resumes the dumping process after the **dumpoff** option has been invoked. Dumps all selected values at the time **dumpon** is invoked. Invokes Verilog `$dumpon` directive.
dumpall	Creates a checkpoint in the VCD file that dumps the current value of all selected variables. Invokes Verilog `$dumpall` directive.
dumplimit <*file_size*>	Limits the size of the VCD file. <*file_size*> specifies the maximum size of the VCD file in bytes. When the size of VCD file reaches the limit, the dump process stops and a comment is inserted in the VCD file indicating that the dump limit was reached. Invokes Verilog `$dumplimit` directive.
dumpflush	Empties the operating system VCD file buffer to ensure that all the data in that buffer is stored in the VCD file. After executing, dump process resumes as before so no value changes are lost. Invokes Verilog `$dumpflush` directive.

Examples

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

Following are the commands you would use to write the VCD simulation values of the module UUT to a VCD file after running simulation for 1000 ns.

Specify which file to write:

```
vcd dumpfile adder.vcd
```

Specify which module net activities to write:

```
vcd dumpvars -m /UUT
```

Run simulation for given time:

```
run 1000 ns
```

Dump the activity data to the VCD file.

```
vcd dumpflush
```

wave log Command

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.

Note

Syntax

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

Options

Option	Description
`-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 will be 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 same as `wave add /UUT/* if *` were to be supported).

Examples

To 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
```

To log all signals in the design:

```
wave log –r /
```

Waveform Window Commands

wcfg new Command

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

Note

Syntax

```
wcfg new
```

Example

To create a new wave configuration:

```
wcfg new
```

wcfg open Command

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

Note

Syntax

```
wcfg open <filename>
```

Options

<filename>	Specifies the name of the wcfg file to open.

Example

To open a wcfg file named "toplevel.wcfg":

```
wcfg open toplevel.wcfg
```

wcfg save Command

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

Note

Syntax

```
wcfg save <filename>
```

Options

<filename>	Specifies the name of the file in which to save the currently active wave configuration.

Example

To save the active wave configuration to a wcfg file named "toplevel.wcfg":

```
wcfg save toplevel.wcfg
```

wcfg select Command

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

Note

Syntax

```
wcfg select <wave_config_name>
```

Options

<wave_config_name>	Specifies the wave configuration to activate. The *<wave_config_name>* must be the name of an open existing wave configuration or the command will report an error.

Example

To activate the wave configuration named "design":

```
wcfg select design
```

wave add Command

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 –wdb switch supplied to the simulation executable. The wave configuration displays in the Wave window.

The equivalent GUI command is Add to Wave Window.

Note

Syntax

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

Options

Option	Description
-into *<ID>*	Specifies object ID of the group or virtual bus into which the object should be added.
-wcfg *<wave_config_name>*	Specifies the wave configuration name to which the object should be added. If no such configuration of that name exists already, a new one is created. This option is being deprecated. Please refrain from using this option. Instead, use *wcfg new* or *wcfg select* to add simulation objects to a desired wave configuration and wcfg save *<filename>* to save the wave configuration into a particular file.
-reverse	Reverses the bus order.
-radix *<radix>*	Uses the specified radix when displaying signal values. The value of <radix> is one of default, bin, oct, hex, dec, unsigned, or ascii.
-color *<color>*	This option sets the color of the simulation object(s) to *<color>*. The value of *<color>* is defined in Red/Green/Blue (RGB) format. For example #0000FF is blue, #FF0000 is red, and #00FF00 is green. Textual name of color can also be specified for some of the popular colors. 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.
-name *<custom_name>*	Names the wave object with a custom name.
-r	This option applies to the specified block name(s). With **-r**, the objects associated with each block are added to the waveform, down to the lowest level of hierarchy. Without **-r**, the first level objects of the block(s) entered as *object_name* are added.
<object_name>	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. Wild card 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 to be supported)

Examples

The **wave add** command can be used as follows.

To add the signals associated with object UUT to the current wave configuration:

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

To add all top-level signals:

```
wave add /
```

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

```
wave add –r / -color #00FF10
```

To add specific signals with radix hex and in color red:

```
wave add /tb/clk /tb/UUT/data –radix hex –color red
```

divider add Command

The **divider add** command adds a new divider.

Note

Syntax

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

Options

Option	Description
-into <*ID*>	This option specifies the object ID of the group into which the divider should be added.
-color <*color*>	This option sets the divider color to <*color*>. The value of <color> is defined in RGB format. For example #0000FF is blue, #FF0000 is red, and #00FF00 is green. Textual name of color can also be specified for some of the popular colors. 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.

Examples

To add a divider with name “Inputs” to the current wave configuration:

```
divider add Inputs
```

To add red divider with name “Outputs”:

```
divider add Outputs –color red
```

To 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 Command

The **group add** command adds a new group.

Note

Syntax

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

Options

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

Examples

To add a group with name “Inputs” to the current wave configuration:

```
group add Inputs
```

To 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
```

To 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 Command

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

Note

Syntax

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

Options

Option	Description
<name>	Name of the virtual bus.
-into <ID>	This option specifies object ID of an existing virtual bus into which the newly created virtual bus should be added.
-reverse	This option reverses the bus order.
-radix <radix>	This option uses the specified radix when displaying signal values. The value of <radix> is one of bin, oct, hex, signed, dec, or ascii.
-color <color>	This option sets the color of the virtual bus to <color>. The value of <color> is defined in RGB format. For example: #0000FF is blue, #FF0000 is red, and #00FF00 is green. Textual name of a color can also be specified for some of the popular colors. 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.

Examples

To add a virtual bus with name “mybus" having radix as hexadecimal to the current wave configuration:

```
virtualbus add mybus –radix hex
```

To 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 Command

The **marker add** command adds a new marker.

Note

Syntax

marker add <*time*>

Options

Option	Description
<time>	Specify time location at which the new marker should be added. If the time unit is not specified, then the default user time unit is returned by running the `isim get userunit` command.

Examples

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

```
marker add 10 ns
```

XILINX®

Chapter 11

ISim Hardware Co-Simulation

Introduction

Hardware Co-Simulation (HWCoSim) is integrated into ISim as a complementary flow to the software-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. This document describes the system requirements and usage of the HWCoSim in ISim. Submit feedback about this flow to hwc_feedback@xilinx.com.

Prerequisites

Hardware co-simulation in ISim has the following requirements:

- Xilinx® ISE Design Suite 13x (any edition)
- Windows 32-bit and 64-bit, Linux 32-bit and 64-bit
- 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

Hardware Co-Simulation (HWCoSim) in ISim 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 software testbench (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 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 software 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 offers 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 software 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 software testbench

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

Hardware co-simulation (HWCoSim) in ISim has the following limitations:

- Only one instance in a design can be selected for hardware co-simulation, and it cannot be the top-level testbench itself.
- The selected instance for hardware co-simulation must be able to be synthesized using XST, and must be implementable 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 is controlled by ISim, and is asynchronous to the software 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

Compilation

As with software-based HDL simulation, you first compile a design into a simulation executable before performing Hardware Co-Simulation (HWCoSim). Invoke the ISim compiler `fuse` for compilation, using the command line or through Project Navigator 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>
   -hwcosim_instance <instance>
   -hwcosim_clock <clock>
   -hwcosim_board <board>
   -hwcosim_incremental [0|1]
   -hwcosim_constraints <constraints file>
```

- `-hwcosim_instance` specifies the full hierarchical path of the instance to co-simulate in hardware
- `-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_board` specifies the identifier of the hardware board to use for co-simulation.

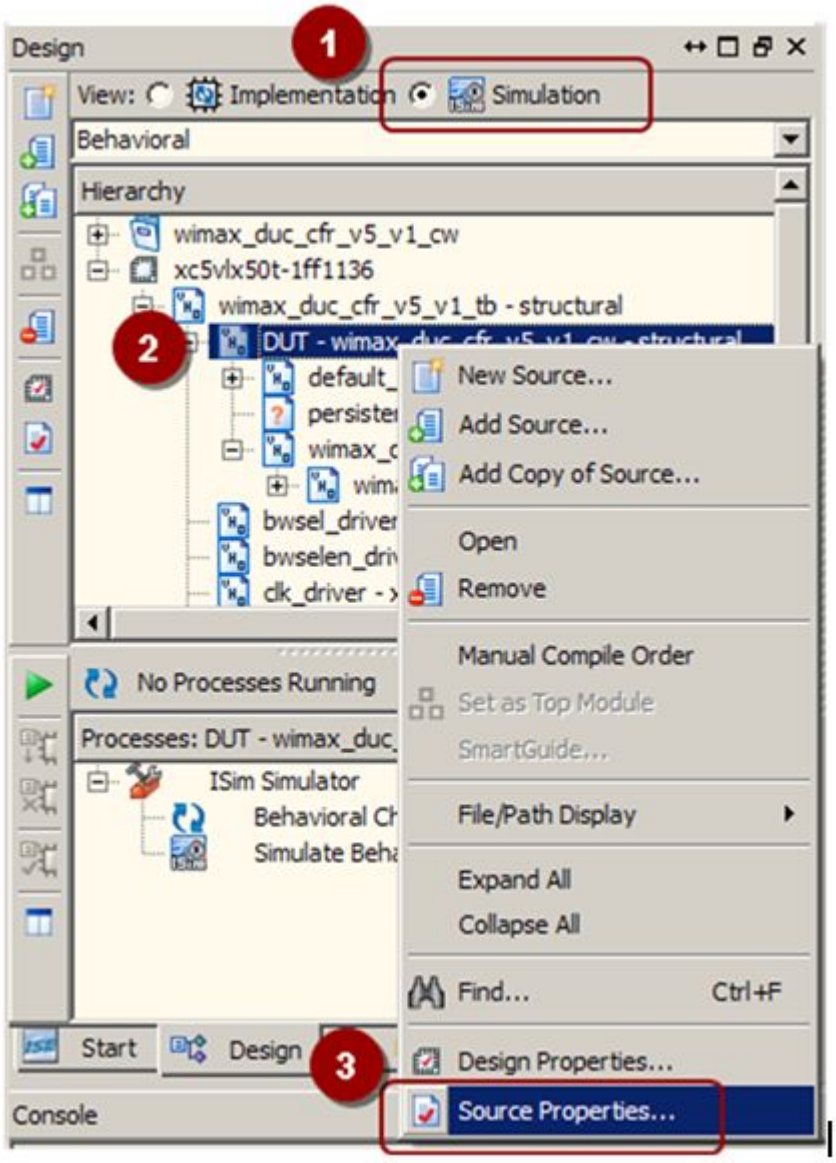
  See Board Support for a list of supported boards.
- `-hwcosim_incremental` (optional) specifies whether `fuse` should reuse the last generated hardware co-simulation bitstream and skip the implementation flow.
- `-hwcosim_constraints` (optional) specifies the custom constraints file that provides additional 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.

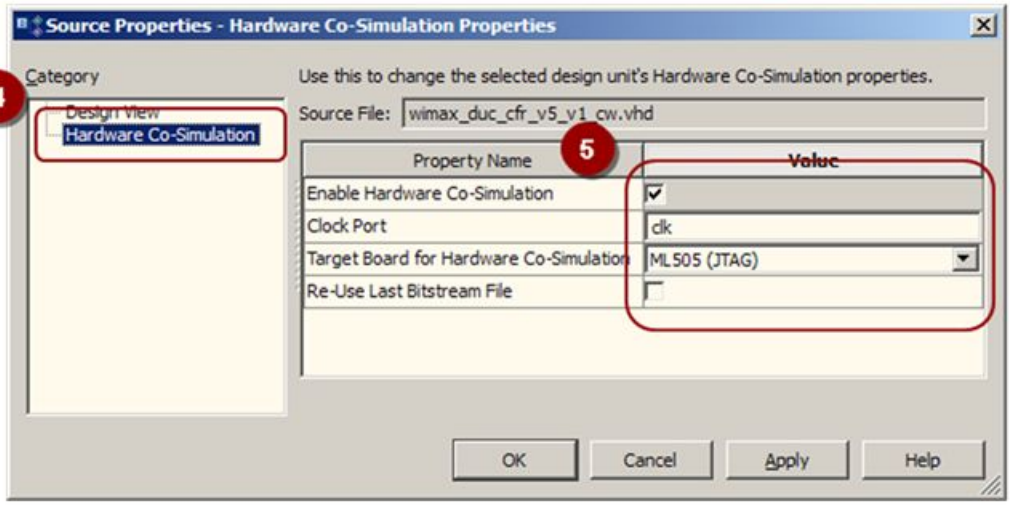
Project Navigator Flow

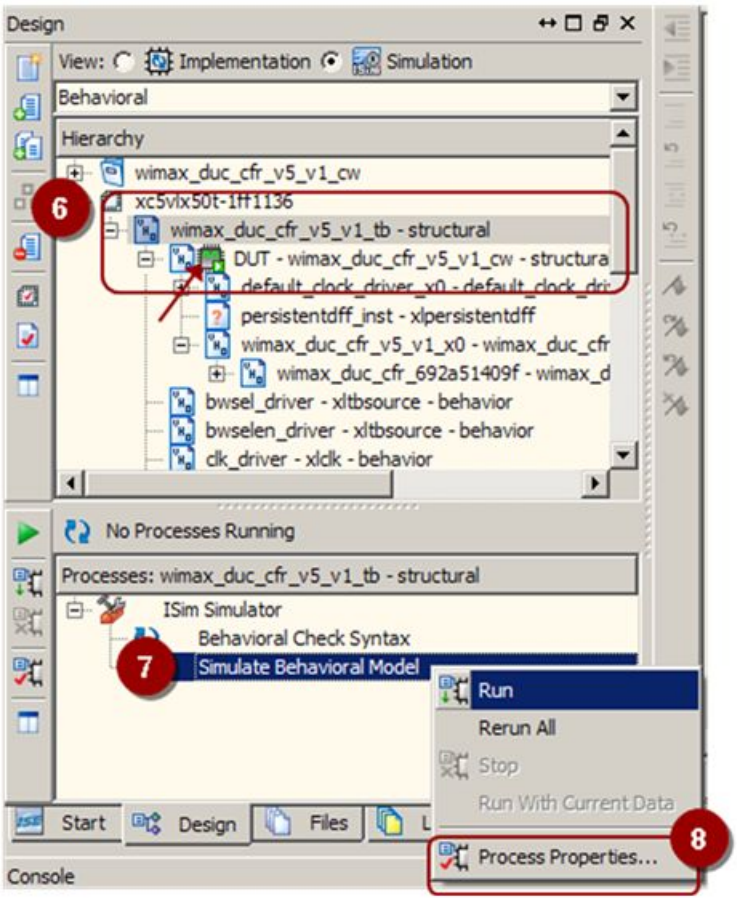
1. Ensure you have ISim selected as the simulator for the project. Switch to the Simulation view.
2. From the Hierarchy pane, select the instance to co-simulate in hardware and right-click to show the popup menu.
3. Select **Source Properties** from the popup menu to open the Source Properties dialog 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** checkbox. 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.
   - Select a board from the **Target Board for Hardware Co-Simulation** pulldown list. The list shows only the boards with an FPGA that is under the same device family chosen for the project.
   - If a previous hardware co-simulation bitstream is available and the instance under co-simulation remains unchanged, check the **Reuse Last Bitstream File** checkbox 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 .

6. Select the testbench 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.
7. Go to the Instances and Processes panel of the testbench and double-click on **Simulate Behavior Model** to start the compilation and simulation process.
8. Open the Process properties for the **Simulate Behavior Model** to specify any additional options for ISim before starting the compilation and simulation process. The following example displays steps 1 — 8:



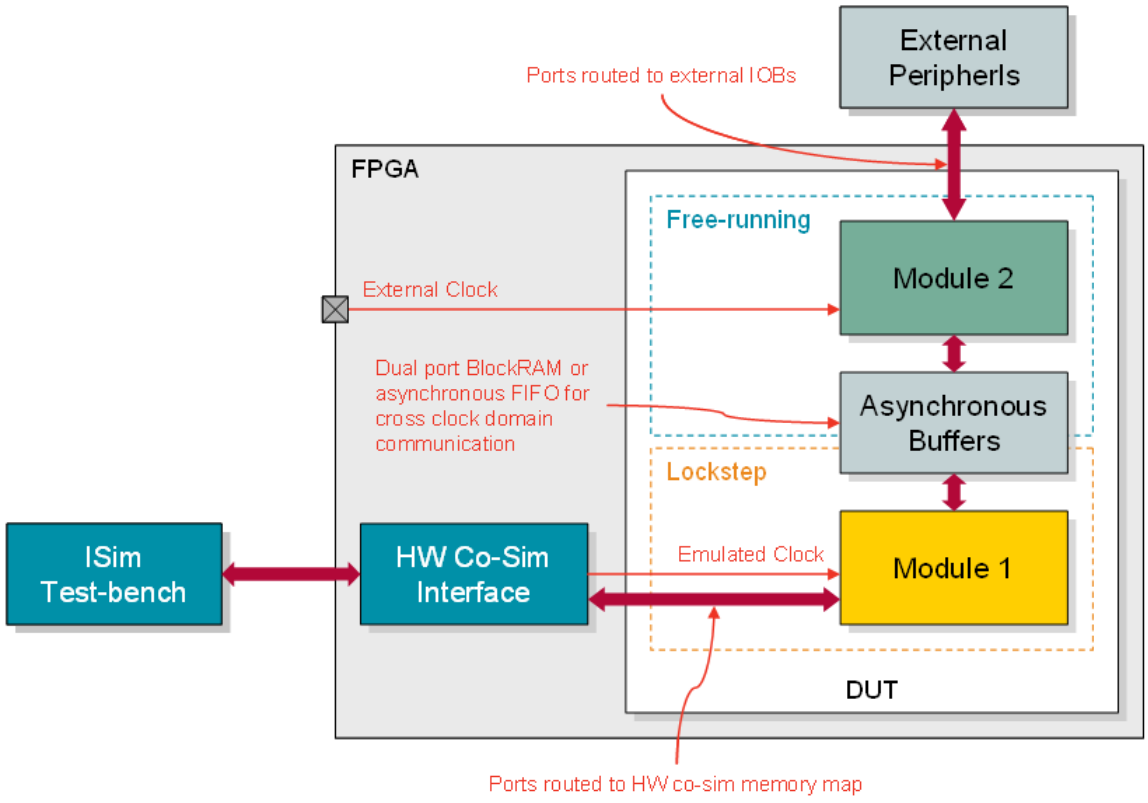

Source Properties - Hardware Co-Simulation Properties
Category
Design View
Hardware Co-Simulation
Use this to change the selected design unit's Hardware Co-Simulation properties.
Source File: wimax_duc_cfr_v5_v1_cw.vhd
Property Name
Value
Enable Hardware Co-Simulation
Clock Port
clk
Target Board for Hardware Co-Simulation
ML505 (JTAG)
Re-Use Last Bitstream File
OK
Cancel
Apply
Help
4
5


Design
View: Implementation Simulation
Behavioral
Hierarchy
wimax_duc_cfr_v5_v1_cw
xc5vlx50t-1ff1136
wimax_duc_cfr_v5_v1_tb - structural
DUT - wimax_duc_cfr_v5_v1_cw - structura
persistentdff_inst - xlpersistentdff
wimax_duc_cfr_v5_v1_x0 - wimax_duc_cfr
wimax_duc_cfr_692a51409f - wimax_d
bwsel_driver - xltbsource - behavior
bwselen_driver - xltbsource - behavior
clk_driver - xlclk - behavior
No Processes Running
Processes: wimax_duc_cfr_v5_v1_tb - structural
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model
Run
Rerun All
Stop
Run With Current Data
Process Properties...
Start
Design
Files
Console
6
7
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. The following diagram outlines the concept:



2. Make a copy of 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 a simple example, suppose you have a FIFO design and you want to single step the write side and free run the read side.

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

   The `isim_hwcosim.ucf` would look like:

```
# 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

Setting Up a Hardware Board

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

1. Make sure 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 make sure the LED on the cable is green. Install the Xilinx cable drivers when prompted. For more information about Ethernet, see Determining the Ethernet

Hardware Co-Simulation

Unlike the executable for software simulation, the executable for hardware co-simulation 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 software simulation flow:

- Invoking the executable launches a Tcl shell interface for controlling the simulation
- Invoking the executable with the **-gui** option launches the ISim 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 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, which means you need to first use the scope command to select an instance, which 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`.

  ```
  isim> scope /mytestbench/top/hwcosim_inst

  isim> 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, and thus 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.

  ```
  isim> init
  isim> scope /mytestbench/top/hwcosim_inst
  isim> hwcosim set skipConfig 1
  isim> run 1000 ns
  isim> restart
  isim> scope /mytestbench/top/hwcosim_inst
  isim> hwcosim set skipConfig 1
  isim> 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 may 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.
- **shareCable**: Default is 0. This property is only available for JTAG-based co-simulation interfaces. Set to 1 if the JTAG cable should be shared with the EDK External Memory Debugger (XMD) or ChipScope 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.

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 Hardware Co-Simulation (HWCoSim) in ISim. 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, which must be named as `hwcosim.bsp`, in the directory where `fuse` is invoked. The board support file defines a list of board specification in the following format.

In the following example, **ml402-jtag** is the board identifier that is provided to the `fuse` command to compile the design for the given board. The board identifier includes the following list of properties:

- **Description** provides the description of the board
- **Vendor** specifies the board vendor
- **Type** specifies the type of co-simulation interface to be used. . Allowed values are: **jtag** and**ppethernet** (for Point-To-Point Ethernet-based HWCoSim).
- **Part** specifies the part name of the FPGA on the board.
- **Clock** provides the system clock information, where:

  Period (and **VariablePeriods**) specifies the supported clock period(s) in nanoseconds.
- **Pin** specifies the clock pin location. For differential clock sources, provide both **Positive** and **Negative** clock pin locations.
- **BoundaryScanPosition** specifies 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® 13.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 <jtag/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 <jtag/ppethernet> Xilinx Virtex-6 ML605 Evaluation Platform
- **kc705-jtag**: Xilinx Kintex™-7 FPGA KC705 Evaluation Kit
- **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 <jtag/ppethernet> Xilinx Spartan®-6 SP601 Evaluation Platform
- **sp605-jtag**: sp605 <jtag/ppethernet> Xilinx Spartan-6 SP605 Evaluation Platform

Example Board Support File

```
Example Board Support File{
  'ml402-jtag' => {
    'Description' => 'ML402 (JTAG)',
    'Vendor' => 'Xilinx',
    'Type' => 'jtag',
    'Part' => 'xc4vsx35-10ff668',
    'Clock' => {
      'Period' => 10,
      'Pin' => 'AE14',
    },
    'BoundaryScanPosition' => 3,
  },
  's3adsp-3400a-jtag' => {
    'Description' => 'Spartan-3A DSP 3400A Development Platform (JTAG)',
    'Vendor' => 'Xilinx',
    'Type' => 'jtag',
    'Part' => 'xc3sd3400a-4fg676',
    'Clock' => {
      'Period' => 8,
      'Pin' => {
        'Positive' => 'AA13',
        'Negative' => 'Y13',
      },
    },
    'BoundaryScanPosition' => 4,
  },
}
```

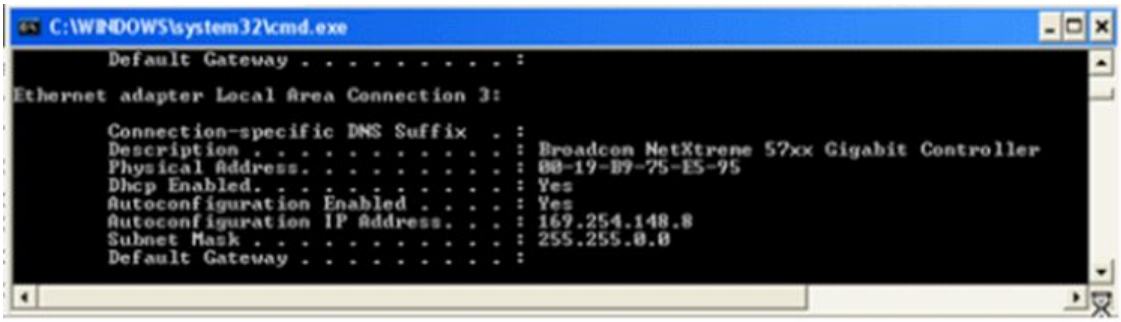
Determining the Ethernet

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. Refer to the following figure for Step 1.



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.
   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 in ISim, as follows:
   a. Open the ISim GUI.
   b. Select the Design under Test (DUT).
   c. Go to the Tcl console.
   d. In the Tcl console, enter the following commands:
      i. Set the Ethernet address:

      ```
      hwcosim set
      ethernetInterfaceID <##:##:L#:##:L#:##>  <physical
      address>
      ```

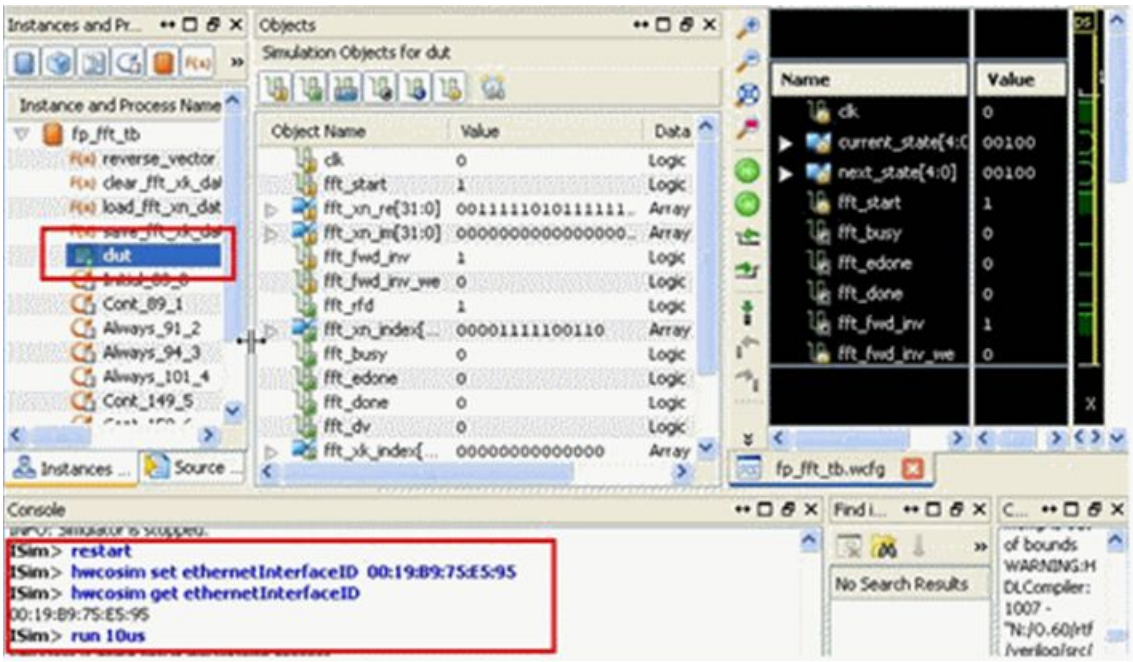
      ii. Verify the Ethernet address:

      ```
      hwcosim get ethernetInterfaceID
      ```

      iii. Verify that the simulation runs:

      ```
      run 10us
      ```

The following figure outlines the process within the ISim GUI.



Frequently Asked Questions

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

General

1. Q: Does ISim-based hardware co-simulation support any kind of designs?

   A: Certain limitations, described in Limitations, apply.

2. Q: Is ISim-based hardware co-simulation 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 software simulation.

3. 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.

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 happen if the DUT already instantiates a BSCAN primitive (such as a ChipScope™ debugging tool ICON or an Embedded Design Kit (EDK) MDM)?

   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 or EDK MDM 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 or EDK MDM module. To share the JTAG cable with ChipScope Analyzer or EDK XMD, you 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 testbench 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 software simulation and the DUT execution.

2. Q: Can I let the DUT clock free-running?

   A: In hybrid co-simulation, you can let a portion of the DUT free-running by mapping one or multiple clock ports to external I/Os. However, at least one clock of the DUT needs to be clocked in a lockstep fashion synchronous to the software 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 software 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 software 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.

XILINX®

Appendix A

Reference

Simulation Executable Commands

Simulation Executable Command Overview

Note The following information is intended for advanced users.

You can run ISim at the command line instead of using the ISE® software or the ISim interface.

Running Simulation From the Command Line

To simulate a Verilog design from the command line, see:

- Running a Functional Simulation From the Command Line With a Verilog Design
- Running a Timing Simulation From the Command Line With a Verilog Design

To simulate a VHDL design from the command line, see:

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

Executable Command Summary

There are a number of commands used to execute a simulation at the command line.

Compilation Commands

- **VHDL compiler (vhpcomp) -** The VHDL compiler, vhpcomp, parses VHDL source files for all design units in those files.
- **Verilog compiler (vlogcomp) -** The Verilog compiler, vlogcomp, parses Verilog source files for all design units in those files.

Elaboration Commands

HDL linker (fuse) - The HDL linker, `fuse`, performs static elaboration of a design in terms of parsed nodes, generates object code for each unique module instance, and links the generated object codes with ISim simulation engine library to create a simulation executable. A user runs this generated simulation executable to effect simulation of the design under consideration.

Simulation Commands

Simulation Executable - The ISE Simulation Executable is generated by the fuse command. To run the simulation of a design in ISim, the generated simulation executable needs to be invoked. When ISim is run inside the ISE interface, ISE takes care of invoking the generated simulation executable. A command-line user needs to explicitly invoke the generated simulation executable to effect simulation. The simulation executable effects event-driven simulation and has rich support for driving and probing simulation using Tcl.

Project Files

A project file can be used to invoke vhpcomp, vlogcomp and fuse. You can collect all of the design files for a particular project into a project file, and use the **-prj** option to specify the project file. The project file can contain VHDL or Verilog files.

Each line in a project file must be formatted as follows:

```
<language> [<library>] <filename> {[-d <macro>] [-i
<include_path>]}
```

where,

<language> is either **vhdl** or **verilog**.

<library> is an optional working library name.

Tcl Commands

The simulation executable provides a powerful Tcl interface that can be run in batch mode. Tcl commands can be used to control the simulator and view the results of simulation. More Info

Invoke the simulation executable using the **-tclbatch** option to list a number of Tcl commands in a batch file.

ISim Simulation Executable

ISim Simulation Executable Overview and Syntax

Note The following information is intended for advanced users.

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

The syntax for this command is as follows:

```
<executable_name> .exe (option)
```

where *<executable_name>.exe* is user-defined or is `x.exe` by default.

Note This command is case sensitive.

ISim Executable Options

The ISim executable command options are as follows.

Note This command is case sensitive.

-f *<cmd_file>*	You can save ISim engine command options in a text file for future use. This option reads and executes the saved options that are specified in *cmd_file*.
-gui	Launches the ISim GUI.
-h	Displays all command line options and their usage.
-intstyle **ise** | **xflow** | **silent**	Use one of the specified styles for printing messages. Specify **ise** to format messages for the ISE® Console window or **xflow** to format messages for XFLOW. Specify **silent** to suppress all messages.

-log *<file_name>*	Generates a log file with name specified by *<file_name>*.
-maxdeltaid *<number>*	Specifies the maximum delta number. *<Number>* is any integer.
-nolog	Blocks log file generation.
-sdfnowarn	Do not display SDF warnings.
-sdfnoerror	Treat errors found in SDF file as warnings.
-sdfmin|-sdftyp|-sdfmax *<root=file>*	Specifies the type of delays for ISim to use. • **-sdfmin** - SDF annotates <file> at <root> with minimum delay. • **-sdftyp** - SDF annotates <file> at <root> with typical delay. • **-sdfmax** - SDF annotates <file> at <root> with maximum delay.
-sdfroot *<root_path>*	Sets default place in the design hierarchy where SDF annotation is applied.
-tclbatch *<file_name>*	Turn batch mode on. By default batch mode is off. When batch mode off, Tcl commands can be issued from the Console panel even if the engine is busy simulating. When batch mode is on, all of the commands in the specified batch file are executed sequentially until completion, ignoring any commands entered from the command prompt. *file_name* specifies the name of file containing Tcl commands to be executed. For information about ISim Tcl commands, see Simulation Command Overview.
-testplusarg *<string|stringvalue>*	When the simulator matches this command line argument string with the `$test$plusarg` or `$value$plusarg` system function of a Verilog design file, the test or design behavior change associated with the system function is run. *string* is any string. For example, **`-testplusarg HELLO`**. If a Verilog file uses ($test$plusargs("HE")), the function will return true. *stringvalue* is an appropriate string for the Verilog format specifiers. It provides a value to the variable in the `$value$plusargs` system function call. For example, **`-testplusarg FINISH=10000`**. If a Verilog file uses `($value$plusargs("FINISH=%d", stop_clock))` and when Verilog format specifier %d matches 10000, and stop_clock get values 10000 and 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.
-vcdfile *<vcd_file>*	Specifies the VCD output file for Verilog projects. Default file name is `dump.vcd`.

-vcdunit *<unit>*	Specifies the VCD output time unit. Possible values are **fs**, **ps**, **ns**, **us**, **ms** and **sec**. Default is **ps**.
-view *<waveform_file.wcfg >*	Used in combination with the **-gui**switch to open the specified waveform file in the ISim graphic user interface.
-wdb *<waveform_database_file.wdb >*	Simulation data is saved to the specified file. For example: **x.exe –wdb my.wdb** saves the simulation data to `my.wdb` instead of the default `isimgui.wdb`.

ISim Simulation Executable Examples

The ISim simulation executable command can be used as follows.

To run the simulation executable named `watchvhd` in batch mode:

```
watchvhd.exe -tclbatch batchfile
```

To run the simulation executable named `watchvhd` with back-annotation using the delay information in the file `adder.sdf`:

```
watchvhd.exe -sdftyp adder.sdf
```

To run the simulation executable interactively:

```
watchvhd.exe
```

To run the simulation in the ISim GUI:

```
watchvhd.exe –gui
```

To run simulation for 1000 clock cycles and terminate simulation using the following Verilog code.

```
real frequency;
reg [8*32:1] testname;
integer stop_clock;
if ($value$plusargs("FINISH=%d", stop_clock))
begin
repeat (stop_clock) @(posedge clk);
$finish;
end
```

Enter:

```
x.exe -testplusarg FINISH=10000
```

ISim Hardware Co-Simulation Overview

Hardware co-simulation is integrated into ISim as a complementary flow to the software-based HDL simulation. This feature enables the simulation of a design or a submodule of the design to be off-loaded to hardware (Xilinx® FPGA on a regular board). It can accelerate the simulation of a complex design and verify that the design actually works in hardware.

- **Use Models -** Hardware co-simulation in ISim currently supports two use models: one for pure logic-based designs, and one for designs containing external inputs and outputs.
- **Usage -** Similar to software-based HDL simulation, you must first compile a design into a simulation executable before doing hardware co-simulation. The compilation is performed by invoking the ISim compiler fuse using the command line or through

Project Navigator. Implementation tools are run and a hardware co-simulation bitstream is produced at the end of the compilation. This bitstream is used for co-simulating the portion running in hardware with the portion running in ISim.

fuse

fuse Overview and Syntax

Note The following information is intended for advanced users.

The `fuse` command is the Hardware Description Language (HDL) compiler, elaborator, and linker used by the ISim. The command links design units already compiled with `vhpcomp` or `vlogcomp`, and creates a simulation executable. It also takes a mixed language project file and compiles the design units on-the-fly. Name(s) of the top-level design unit(s) must be specified on the `fuse` command line argument. The `fuse` command effects *static elaboration* on the design given the top design units and then compiles the design units to object code. The design unit object codes are then linked together to create a simulation executable.

- Use one or more top-level design unit names as {[<*library_name*>.]<*top_name*>}. For example, the design unit name of the testbench file. If the design uses a Verilog UNISIM or Verilog SIMPRIM library, one top name must be `glbl`. Including a library name is optional. The default library name `work` is assumed when a library name is not specified.
- Use the **`-prj`** option to call `vhpcomp` or `vlogcomp`, as appropriate, and to compile your HDL code.
- Use the **`-o`** <*`sim_exe`*> option to changes the simulation executable file name and location from the default `x.exe`.

Note To exclude certain lines in a `.prj` file in the `fuse` command, use the "--" option.

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

Note Do no remove the `isim/`<*`simulation_executable`*> `.sim` directory otherwise the design cannot be simulated.

Syntax

`fuse` (*`option`*)

where <*option*> is any option found in fuse Options.

Note This command is case sensitive.

fuse Options

The **`fuse`** command options are as follows.

`-d` <*`macro_definition`*> [= <*`value`*>]	This a Verilog-only option. Define the macros used in Verilog files, and any value they require. More than one **`-d`** can be specified. **Note** Ensure that there is no space between the **=**and the value; the space would be interpreted as part of the value.
`-f` <*`cmd_file`*>	You can save fuse command options in a text file for future use. This option reads and executes the saved options, specified in <*cmd_file*>.

-generic_top "<*parameter*>=<*value*>"	Overrides generic or parameter of a top level design unit with the specified value. For example, **-generic_top "P=10"** would apply the value of 10 on the top-level parameter before elaboration.
-h	Displays all command line options and their usage.
-hwcosim_instance arg	This Hardware Co-Simulation (HWCoSim) option specifies the hierarchical name of the instance to be run on the FPGA. For Example: `/testbench/UUT`
-hwcosim_clock arg	This HWCoSim option specifies the clock port name on the instance.
-hwcosim_board arg	This HWCoSim option specifies the name of the board.
-hwcosim_reuse_last_bitstream arg	This HWCoSim option skips the implementation phase and reuses the previously created bit file. Allowed values are: 0 and 1 (Default:0)
-i <*include_path*>	This option is for Verilog only. Specifies that if fuse calls vlogcomp, it should use the specified path for Verilog 'include directives. Each **-i** can be used for only one include path. More than one **-i** can be specified. Place quotes around paths with spaces.
-incremental	Compiles only the files that have changed since last compile.
-initfile <*sim_init_file*>	Specifies a user defined simulator init file to add to or to override the logical-to-physical mappings of libraries provided by the default `xilinxsim.ini` file.
-intstyle **ise** | **xflow** | **silent**	Use one of the specified styles for printing messages. Specify **ise** to print messages for the ISE® software Console or **xflow** to print messages for XFLOW. Specify **silent** to suppress all messages. By default all messages are printed.
-ise <*file*>	Lets you specify a Xilinx® ISE file.
-L|-lib <*search_lib*> [= <*lib_path*>]	Specifies other libraries and optionally the physical path name for those libraries. Multiple **-L** can be used, and are treated as resource libraries. The physical path provided through **-L** overrides mappings provided by the `xilinxsim.ini` file. *Search_lib* is the logical name of the specified library optionally followed by the <*lib_path*>, the path to the physical library. For example: **-L mylib=C:home/mylib** For Verilog designs, fuse searches for libraries in the order that the **-L** options are coded. For example:

	`fuse -L unisim -L abcsim -L xyzsim mytop` First, `fuse` searches for design units in UNISIM, and then *abcsim*, and *xyzsim*. If a design unit was defined in abcsim as well as in *xyzsim*, the one in *abcsim* would be used as that appears before *xyzsim*. If the order was changed to: **`fuse -L unisim -L xyzsim -L abcsim mytop`** And both *xyzsim* and *abcsim* defined the same design unit, `fuse` would select the design unit from *xyzsim*.
`-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 elaborator. If a design exceeds the depth, elaborator would error out. Can be used to increase the depth in case the elaborator falsely thinks 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. Possible values are **`on`**, **`off`**, or an integer greater than 1. Default is on, where the compiler automatically chooses a number based on number of cores in the system.
`-nodebug`	Generates output that has no information for debugging your HDL code during simulation. Output with no debug information runs simulation faster. The default is to generate HDL units for debugging.
`-nospecify`	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_lib>***`/`***<mod_name>***`/`***<platform>***`/x.exe`*** where: • *<work_lib>* is the work library. • *<mod_name>* is the first top module specified. • *<platform>* is Windows.

-override_timeprecision	Overrides the time precision (unit of accuracy) of all Verilog modules in the design with the time precision specified in the **-timescale** option.
-override_timeunit	Overrides the time unit (unit of measurement of delays) of all Verilog modules in the design with the time unit specified in the **-timescale** option.
-prj <*prj_file*>**.prj**	Specifies a project file to use for input. A project file contains a list of all the files associated with a design. It is the main source file used by the ISE software. <*prj_file*> is the project file and must have a **prj** extension.
-rangecheck	VHDL-only option. Specifies value range check to be performed on VHDL assignments. **Note** This option does not affect index range checking for arrays. ISim *always* checks an index into an array for being within the allowed range. By default **-rangecheck** is turned off.
-sourcelibdir	Specifies the source directory for library modules. For more information and examples, see Supporting Source Libraries.
-sourcelibext	Specifies the file extension for source files for modules. The **-sourcelibdir** option provides the location for these files. For more information and examples, see Supporting Source Libraries.
-sourcelibfile	Specifies the filename for library modules. For more information and examples, see Supporting Source Libraries.
-timeprecision_vhdl<*time_precision*>	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). The default is 1ps.
-timescale <*time_unit*/*time_precision*>	Specifies the default timescale for Verilog modules that do not have an effective timescale. The <*time_unit*> is the unit of measurement of delays. The <*time_precision*> is the unit of accuracy. Both <*time_unit*> and *time_precision* are entered as number (1|10|100|...) followed by unit (fs|ps|ns|us|ms|s). The default **-timescale** is 1ns/1ps.
-typdelay	Verilog- only option. Specifies that if fuse calls vlogcomp, it should use typical delays.

-v *<value>*	Specifies the verbosity level for printing messages. Allowed values are 0, 1, or 2. The default is 0. `fuse -v 1` prints useful debugging information, which can help to identify problems in ISim compilers. • Dumps the library mapping as seen by ISim compiler after reading all available library mapping files (`xilinxsim.ini`) • Gets verbose messages from design elaborator • Gets the dumps of current values of environment variables which affect the behavior of the ISim compiler • Gets the list of loaded shared objects by the ISim compiler • Dumps operating system information, including version number and processor • Dumps path to the GCC compiler being used to compile the generated code
-version	Prints the compiler version.

fuse Examples

Using Precompiled HDL

The following examples show how to invoke fuse using precompiled HDL.

For VHDL using a top level configuration:

```
fuse work.yourtop
```

For Verilog or mixed language design using all of the top-level modules:

```
fuse work.top_level_module_name_1 work.top_level_module_name_2
work.glbl -L simprims_ver -L logicalLib1 -o mysim.exe
```

Using HDL Source

Example of fuse invoked using source VHDL -

This example produces an executable called `tb.exe` from VHDL source listed in the project file called `x.prj`. The contents of the project file are as follows:

```
VHDL work x1.vhd
VHDL work x2.vhd
VHDL work x3.vhd
VHDL work tb.vhd
```

```
fuse -prj x.prj work.tb_top -o tb.exe
```

To start simulation, execute the following:

```
tb.exe
```

Example of fuse invoked using source Verilog -

This example produces an executable called `tb.exe` from Verilog source listed in the project file called `myproj.prj`. The top level design unit is *tb* defined in file `tb.v`. The contents of `myproj.prj` are as follows:

```
Verilog work x1.v
Verilog work x2.v
Verilog work x3.v
Verilog work tb.v
```

Use the following command to run fuse:

```
fuse -prj myproj.prj work.tb work.glbl -o tb.exe
```

To start simulation, execute the following:

```
tb.exe
```

Note For Verilog, if the design instantiates any modules that have been compiled into any libraries other than the *work* library, those libraries must be passed to the `fuse` linker using the `-L` command line option so that `fuse` can locate those design units and link them into the simulation executable.

Example of fuse invoked using mixed VHDL/Verilog design -

This example produces an executable called `tb.exe` from Verilog and VHDL source code listed in the project file called `myproj.prj`. There are two top level design units: a VHDL top-level called `tb` and a Verilog top level unit called `glbl` defined in `tb.vhd` and `glbl.v` respectively. The contents of `myproj.prj` are as follows:

```
Verilog work x1.v
VHDL work x2.vhd
Verilog work x3.v
VHDL work x4.vhd
Verilog work glbl.v
VHDL work tb.vhd
```

Use the following command to run fuse:

```
fuse work.tb work.glbl -prj x.prj -o tb.exe
```

To start simulation, execute the following:

```
tb.exe
```

Note For mixed language designs, for the modules on language boundary and for the Verilog modules that have been compiled into any libraries other than the *work* library, those libraries must be passed to `fuse` using the `-L` command line option in the desired search order so that fuse can locate those design units and link them into the simulation executable. For more information, see Mixed Language Simulation Overview.

Using the Command File Option

The following example shows how to invoke fuse using the **`-f`** option:

```
fuse -f my_design.cmd
```

The following is an example of a command file for Verilog:

```
-nodebug
-intstyle xflow
-incremental
top_level_module_name_1
top_level_module_name_n
-L logicalLib1
-L logicalLib2
```

vlogcomp

vlogcomp Overview and Syntax

Note The following information is intended for advanced users.

ISim uses the Verilog compiler, `vlogcomp`, to parse Verilog source files and generate object code for all design units in those files. The object code generated by `vlogcomp` is used by fuse to create a simulation executable.
You must specify either a project file or one or more Verilog source files to compile. If neither the project file nor the Verilog file is specified, vlogcomp issues an error.

Syntax

`vlogcomp` (*`option`*)

where *option* is any option found in vlogcomp Options.

Note This command is case sensitive.

vlogcomp Options

The **vlogcomp** command options are as follows.

<verilog_file>	Specifies the Verilog file to be compiled.
-d *<macro_definition>* [=*<value>*]	Define the macros used in Verilog files, and any value they require. More than one **-d** can be specified
-f *<cmd_file>*	You can save `vlogcomp` command options in a text file for future use. This option reads and executes the saved options, specified in *<cmd_file>*.
-h	Displays all command line options and their usage.
-i *<include_path>*	Specifies path for Verilog 'include directives. More than one **-i** can be specified.
-incremental	Compiles only the files that have changed since last compile.
-initfile<*sim_init_file*>	Specifies the physical path to user defined simulator `init` file instead of default `xilinxsim.ini` file.
-intstyle ise |xflow |silent	Use one of the specified styles for printing messages. Specify **ise** to print messages for the ISE® Console or **xflow** to print messages for XFLOW. Specify **silent** to suppress all messages. By default all messages are printed.
-ise *<file>*	Lets you specify a Xilinx® ISE file.
-L|-lib *<search_lib>* [=*<lib_path>*]	Specifies other libraries and optionally the physical path name for those libraries. Multiple **-lib** options can be used, and are treated as resource libraries. The physical path provided through **-lib** overrides the mappings provided by the `xilinxsim.ini` file. The *<search_lib>* is the logical name of the specified library optionally followed by the *<lib_path>*, the path to the physical library. For example: **`-lib mylib=C:/home/mylib`**

-maxdelay	Specifies that **vlogcomp** should use worst case delays.
-mindelay	Specifies that **vlogcomp** should use fastest possible delays.
-nodebug	Generates output that has no information for debugging your HDL code during simulation. Output with no debug information runs simulation faster. Default is to generate HDL debug units.
-nospecify	Ignores Verilog path delays and timing checks.
-notimingchecks	Ignores timing check constructs in Verilog specify blocks.
-prj *<prj_file>* **.prj**	Specifies a project file to use for input. A project file contains a list of all the files associated with a design. It is the main source file used by the ISE software. *<prj_file>* is the path to the project file and the project file name with a **.prj** extension. You can include an absolute or relative path to the project file. For a relative path, include **../** as part of the path.
-sourcelibdir	Specifies the source directory for library modules. For more information and examples, see Supporting Source Libraries.
-sourcelibext	Specifies the file extension for source files for modules. The **-sourcelibdir** option provides the location for these files. For more information and examples, see Supporting Source Libraries.
-sourcelibfile	Specifies the filename for library modules. For more information and examples, see Supporting Source Libraries.
-typdelay	Specifies that **vlogcomp** should use typical delays.
-v *<value>*	Specifies the verbosity level for printing messages. Allowed values are 0, 1, or 2. The default is 0.
-version	Prints the compiler version.
-work *<work_lib>* [= *<lib_path>*]	Specifies the **work** library, and optionally, the physical path for the work library. The physical path provided through this option overrides mappings provided by the **xilinxsim.ini** file. The default work library is the logical library **work**. The *<work_lib>* is the logical name of the specified work library optionally followed by *<lib_path>*, the path to the physical library. For example: **mywork=C:/home/worklib**

vlogcomp Examples

The **vlogcomp** command can be used as follows:

- To invoke the Verilog compiler with all options saved in a file called run32.txt:**vlogcomp -f run32.txt**
- To invoke the Verilog compiler using a work library with the logical name mysimwork located in the /home/smithjj/mylib directory, and compile all the Verilog files in the project file dsp64.prj:

  vlogcomp -work mysimwork=/home/smithjj/mylib -prj dsp64.prj
- To invoke the Verilog compiler using the default work library specified in the xilinxsim.ini file and compile the Verilog files suba.v and subb.v:

  vlogcomp suba.v subb.v

vhpcomp

vhpcomp Overview and Syntax

Note The following information is intended for advanced users.

ISim uses the VHDL compiler, vhpcomp, to parses VHDL source files and generate object code for all design units in those files. The object code generated by vhpcomp is used by fuse to create a simulation executable.

Specify either a project file or one or more VHDL files to compile. If neither project file nor VHDL file are specified, vhpcomp issues an error.
Syntax -

vhpcomp (*option*)

where *option* is any option found in vhpcomp Options.

Note This command is case sensitive.

vhpcomp Options

The **vhpcomp** command options are:

<vhdl_file>	Specifies one or more VHDL source files to be compiled.
-f *<cmd_file>*	You can save **vhpcomp** command options in a text file for future use. This option reads and executes the saved options, specified in *cmd_file*.
-h	Displays all command line options and their usage.
-incremental	Compiles only the files that have changed since last compile.
-intstyle ise | xflow | silent	Use one of the specified styles for printing messages. Specify **ise** to print messages for the ISE® software Console **xflow** to print messages for XFLOW. Specify **silent** to suppress all messages. By default all messages are printed.
-ise *<file>*	Specify a Xilinx® ISE file.

-L|-lib <*search_lib*> [=<*lib_path*>]	Specifies other libraries and optionally the physical path name for those libraries. Multiple **-lib** options can be used, and are treated as resource libraries. The physical path provided through **-lib** overrides the mappings provided by the `xilinxsim.ini` file. *Search_lib* is the logical name of the specified library optionally followed by the *lib_path*, the path to the physical library. For example: **-lib mylib=C:/home/mylib**
-nodebug	Generates output that has no information for debugging your HDL code during simulation. Output with no debug information results in a faster simulation runtime. Default is to generate HDL debug units.
-prj <*prj_file*> **.prj**	Specifies a project file to use for input. A project file contains a list of all the files associated with a design. It is the main source file used by the ISE software. The <*prj_file*> is the path to the project file and the project file name with a **.prj** extension. You can include an absolute or relative path to the project file. For a relative path, include ../ as part of the path.
-rangecheck	Enables runtime value range check (VHDL only). This option causes `vhpcomp` to generate output that checks that values assigned to VHDL signals are within their valid range. For example: • If a signal is declared as *positive*, fuse checks that the signal is not assigned a negative number. • If a signal is declared as `std_logic`, **vhpcomp** generates output to check that the signal is assigned only valid `std_logic` values (`U,X,0,1,Z,W,L,H,-`). **Note** This option does not affect the checking of index ranges. ISim *always* checks the ranges of indexes. By default **-rangecheck** is turned off.
-v <*value*>	Specifies the verbosity level for printing messages. Allowed values are 0, 1, or 2. The default is 0.
-work <*work_lib*> [=<*lib_path*>]	Specifies the work library, and optionally, the physical path for the work library. The physical path provided through this option overrides the mappings provided by the `xilinxsim.ini` file. The default work library is the logical library **work**. The <*work_lib*> is the logical name of the specified work library optionally followed by <*lib_path*>, the path to the physical library. For example: **mywork=C:/home/worklib**

vhpcomp Examples

The `vhpcomp` command can be used as follows:

- To invoke the VHDL compiler with all options saved in a file called **`run32.txt`**:

  `vhpcomp -f run32.txt`

- To invoke the VHDL compiler using a work library with the logical name `mysimwork` located in the `/home/smithjj/mylib` directory, and compile all the VHDL files in the project file `dsp64.prj`:

  `vhpcomp -work mysimwork=/home/smithjj/mylib -prj dsp64.prj`

- To invoke the VHDL compiler using the default work library specified in the `xilinxsim.ini` file and compile the VHDL files `suba.vhd` and `subb.vhd`:

  `vhpcomp suba.vhd subb.vhd`

Third-Party Command Equivalency

Third-Party Simulation Command Support Overview

ISim does not support third-party commands. If simulation commands in a DO (`*.do`) file are used, ISim cannot interpret the commands unless there is an exact ISim equivalent command, such as **`run`**).

You can map your DO file commands to ISim commands using the information below:

- Third-Party Compiler Commands
- Third-Party Tcl Commands

Third-Party Compiler Commands

Use the following table as guidance when mapping your DO file compilation commands to ISim commands.

Compiler Command Compatibility Table

DO File Command	ISim Command	Remarks
`vcom -work` *<libname>* **`-93`** *<file_name>*	**`vhpcomp`** *<file_name>*	Compiles a VHDL file.
`vlog -work` *<libname>* *<file_name>*	**`vlogcomp`** *<file_name>*	Compiles a Verilog file.
`vsim` *<lib_name>*`.`*<design_name>*	**`fuse -lib`** *<lib_name>* *<design_name>*	Builds a simulation executable.
`vsim` *<lib_name>*`.`*<design_name>* <mti.do>	*<executable_name>.exe* **`-tclbatch`** *<design_name>*`.tclbatch`	Runs simulation.
`vsim` [**`-sdfmin`** | **`-sdftyp`** | **`-sdfmax`**] [*<instance>*`=`]*<sdf_filename>*] [**`-sdfnoerror`**] [**`-sdfnowarn`**] [**`+sdf_verbose`**] SDF commands can only be invoked as command arguments to `vsim` commands.	**`sdfanno`** {**`-min`**|**`-typ`**|**`-max`**} *<file_name>* [**`-nowarn`**] [**`-noerror`**] [**`-root`***<path_name>*]	Standard Delay Format (SDF) annotation.

Third-Party Tcl Commands

Use the following table as guidance when mapping your DO file simulation commands to ISim commands.

Simulation Tcl Command Compatibility Table

DO File Command	ISim Command	Remarks
bd *id#* 1 or more can be specified.	**bp del** *<index>*[*<index>*...]	Removes breakpoint based on the index where the *<index>* is the index number assigned to the breakpoint you want to delete. Each breakpoint in your design is assigned a unique index number.
bd*<file_name> <line_number>*	**bp remove** *<file_name> <line_number>*	Removes break point at *<line_number>* in *<file_name>*.
bd*<file_name> <line_number>* | *<id#>*	**bp remove** *<file_name> <line_number>*	Removes breakpoint at *<line_number>*in *<file_name>*.
bp *<file_name> <line_number>*	**bp add** *<file_name> <line_number>*	Adds breakpoint at *<line_number>* in *<file_name>*.Ignored options: [**-id** *<id#>*], [**-inst** *<region>*], [**-cond** {*<condition_expression>*}]
bp -query *<file_name>*	**bp list**	Lists all breakpoints.
drivers *<item name>*	**show driver***<net_name>*	Shows all the drivers that are driving the specified *<net_name>*.
env	**scope**	Displays where you are in the design hierarchy.
env ..	**scope ..**	Changes to the parent of current scope.
env *<pathname>*	**scope** *path_name*	Changes to the scope specified by *<path_name>* Ignored options: **-nodataset**, **-dataset**.
examine*<signal_name>*	**show value** *<signal_name>*	Shows the value of a signal.
exit	**exit**	Exits the simulator.
force -deposit *<signal_name> <value>* [*<time>*]	**put**	Changes the value to new value, but the new value gets overwritten by assignments made in HDL.
force -freeze	**isim force add**	Overrides all assignments done from HDL, and makes the signal/wire truly stuck or frozen at a particular value.
help	**help**	Displays all Tcl commands and their usage.
help [*command* | *topic*]	**help** *<command>*	Displays help info on a command.

DO File Command	ISim Command	Remarks
if { [**exa sig_a**] **== "0011ZZ"**} {**echo"Signal value matches"**}	**test** <*signal_name*> <*value*>	Tests the signal value display if it is different.
noforce <*signal*>	**isim force remove** <*signal*>	Removes a value on a signal. This command forces a value on the signal, until isim force remove is issued. This command works only for VHDL signals and Verilog wire types. It does not work for Verilog regs.
quit	**quit**	Exits Tcl prompt. Ignored options: [**-f** | **-force**], [**-sim**].
radix	**isim get radix**	Returns the default radix as a string in Tcl result variable, and displays the default radix to stdout.
radix -<*radix_type*>	**isim get radix** <*radix_type*>	Sets the global radix for the current simulation.
restart	**restart**	Stops simulation and sets simulation time back to 0.
run <*length*> <*unit*>	**run** <*length*> <*unit*>	Runs simulation for <*length*> <*unit*> time.
run -all|-continue	**run** {**all** | **continue**}	Runs simulation until there are no more events.
run for time steps and time units **Note** Also set with the *RunLength* and *UserTimeUnit* variables in the `modelsim.ini` file.	**run**	Runs simulation for 100 ns.
show	**show scope**	Shows current scope in the design hierarchy.
show	**show signal**	Shows all signals and port signals within the current block.
show -all	**show child -r**	Recursively shows all children blocks of current block.
vcd add -r	**vcd dumpvars -m** <*module instance*> **-l** <*int*>	Dumps specific instance with level. Ignored options: [**-in**], [**-out**], [**-inout**], [**-internal**], [**-ports**].
vcd file <*file_name*>	**vcd dumpfile** <*file_name*>	Writes output to <*file_name*>. Ignored options: **-dumpports**, **-map**.

DO File Command	ISim Command	Remarks
`vcd flush`	`vcd dumpflush`	Flushes vcd data to file. No support for *<file_name>*. The file specified by `vcd dumpfile` are flushed.
`vcd limit` *<size>*	`vcd dumplimit` *<no_of_bytes>*	Limits `var` dump size.
`vcd on` | `off` [*<file_name>*]	`vcd` {`dumpon` | `dumpoff`}	Turns `vcd` tracing on or off.

HDL Language Support

VHDL Language Support (a to m)

ISim supports the following VHDL constructs, with exceptions noted. The constructs are listed in alphabetical order. See VHDL Language Support (n to z) for the second half of the list.

Supported VHDL Construct	Exceptions
abstract_literal	Floating point expressed as based literals are unsupported.
access_type_definition	
actual_designator	
actual_parameter_part	
actual_part	
adding_operator	
aggregate	Mixing choice directions in an aggregate not supported.
alias_declaration	Alias to non-objects are in general not supported, in particular following are not supported: • Alias of an alias is not supported. • Alias declaration without subtype_indication is not supported. • Signature on alias declarations is not supported. • Operator symbol as alias_designator is not supported. • Alias of an operator symbol is not supported. • Character literals as alias designators are not supported.
alias_designator	• Operator_symbol as alias_designator is not supported. • Character_literal as alias_designator is not supported.
allocator	
architecture_body	

Supported VHDL Construct	Exceptions
architecture_declarative_part	
architecture_statement_part	
array_type_definition	
assertion	
assertion_statement	
association_element	Now, globally locally static range is acceptable for taking slice of an actual in an association element. Note that non locally static index/slice in formal is an error and we detect and report so. However, a formal name with an index/slice/selection is not supported if the prefix of the formal name is also indexed/sliced/selected.
association_list	
attribute_declaration	
attribute_designator	
attribute_name	Signature after prefix is not supported. The following predefined attributes are supported: • A'ACTIVE, A'ASCENDING([N]), A'HIGH([N]), A'LENGTH([N]), A'LEFT([N]), A'LOW([N]), A'RANGE([N]), A'REVERCE_RANGE([N]), A'RIGHT([N]) • S'DELAYED[(T)], S'EVENT, S'LAST_ACTIVE, S'LAST_EVENT, S'LAST_VALUE • T'ASCENDING, T'BASE, T'HIGH, T'IMAGE(X), T'LEFT, T'LEFTOF(X), T'LOW, T'POS(X), T'PRED(X), T'RIGHT, T'RIGHTOF(X), T'SUCC(X), T'VAL(X)
attribute_specification	
base	
base_specifier	
base_unit_declaration	
based_integer	
based_literal	
basic_character	
basic_graphic_character	
basic_identifier	
binding_indication	Binding_indication without use of entity_aspect is not supported.
bit_string_literal	Empty bit_string_literal ("") is not supported.
bit_value	
block_configuration	
block_declarative_item	
block_declarative_part	

Supported VHDL Construct	Exceptions
block_header	
block_specification	
block_statement	Guard_expression is not supported; for example, guarded blocks, guarded signals, guarded targets, and guarded assignments are not supported.
case_statement	
case_statement_alternative	
character_literal	
choice	Aggregate used as choice in case statement is unsupported.
choices	
component_configuration	
component_declaration	
component_instantiation_statement	
component_specification	
composite_type_definition	
concurrent_assertion_statement	Postponed is not supported.
concurrent_procedure_call_statement	Postponed is not supported.
concurrent_signal_assignment_statement	Postponed is not supported.
concurrent_statement	Concurrent procedure call containing wait statement is not supported.
condition	
condition_clause	
conditional_signal_assignment	Keyword **guarded** as part of options is not supported as there is no supported for guarded signal assignment.
conditional_waveform	
configuration_declaration	Non locally static for generate index used in configuration is unsupported.
configuration_declarative_item	
configuration_declarative_part	
configuration_item	
configuration_specification	
constant_declaration	
constrained_array_definition	
constraint	
context_clause	
context_item	
decimal_literal	
declaration	

Supported VHDL Construct	Exceptions
delay_mechanism	
design_file	
design_unit	
designator	
direction	
discrete_range	
element_association	
element_declaration	
element_subtype_definition	
entity_aspect	
entity_class	Literals, unit, file and group as entity class are not supported
entity_class_entry	Optional <> intended for use with group templates is not supported.
entity_class_entry_list	
entity_declaration	
entity_declarative_item	
entity_declarative_part	
entity_designator	
entity_header	
entity_name_list	
entity_specification	
entity_tag	
enumeration_literal	
enumeration_type_definition	
exit_statement	
exponent	
expression	
extended_digit	
extended_identifier	
factor	
file_declaration	
file_logical_name	Although file_logical_name is allowed to be any wild expression evaluating to a string value, only string literal and identifier is acceptable as file name.
file_open_information	
file_type_definition	
floating_type_definition	
formal_designator	

Supported VHDL Construct	Exceptions
formal_parameter_list	
formal_part	
full_type_declaration	
function_call	In named parameter association in a function_call slicing, indexing or selection of formals is unsupported.
generate_statement	
generate_scheme	
generic_clause	
generic_list	
generic_map_aspect	
graphic_character	
identifier	
identifier_list	
if_statement	
incomplete_type_declaration	
index_constraint	
index_specification	
index_subtype_definition	
indexed_name	
instantiated_unit	Direct configuration instantiation is unsupported.
instantiation_list	
integer	
integer_type_definition	
interface_constant_declaration	
interface_declaration	
interface_element	
interace_file_declaration	
interface_list	
interface_signal_declaration	
interafce_variable_declaration	
iteration_scheme	
label	
letter	
letter_or_digit	
library_clause	
library_unit	
literal	

Supported VHDL Construct	Exceptions
logical_name	
logical_name_list	
logical_operator	
loop_statement	
miscellaneous_operator	
mode	Linkage and Buffer ports are not supported completely.
multiplying_operator	

VHDL Language Support (n to z)

ISim supports the following VHDL constructs, with exceptions noted. The constructs are listed in alphabetical order. See VHDL Language Support (a to m) for the first half of the list.

Supported VHDL Construct	Exceptions
name	
next_statment	
numeirc_literal	
object_declaration	
operator_symbol	
options	Guarded is not supported.
package_body	
package_body_declarative_item	
package_body_declarative_part	
package_declaration	
package_declaration_item	
package_declarative_part	
parameter_specification	
physical_literal	
physical_type_definition	
port_clause	
port_list	
port_map_aspect	
prefix	
primary	At places where primary is used, allocator is expanded there.
primary_unit	
procedure_call	In named parameter association in a procedure_call slicing, indexing or selection of formals is unsupported.
procedure_call_statement	

Supported VHDL Construct	Exceptions
process_declarative_item	
process_declarative_part	
process_statement	Postponed processes are not supported.
process_statement_part	
qualified_expression	
range	
range_constraint	
record_type_definition	
relation	
relational_operator	
report_statement	
return_statement	
scalar_type_definition	
secondary_unit	
secondary_unit_declaration	
selected_name	
selected_signal_assignment	The "guarded" keyword as part of options is not supported as there is no support for guarded signal assignment.
selected_waveform	
sensitivity_clause	
sensitivity_list	
sequence_of_statements	
sequential_statement	
shift_expression	
shift_operator	
sign	
signal_assignment_statement	
signal_declaration	Signal_kind is not supported. Signal_kind is used for declaring guarded signals, which are not supported.
signal_list	
signature	
simple_expression	
simple_name	
slice_name	
string_literal	
subprogram_body	
subprogram_declaration	
subprogram_declarative_item	

Supported VHDL Construct	Exceptions
subprogram_declarative_part	
subprogram_kind	
subprogram_specification	
subprogram_statement_part	
subtype_declaration	
subtype_indication	Resolved subtype of composites (arrays and records) is not supported.
suffix	
target	
term	
timeout_clause	
type_conversion	
type_declaration	
type_definition	
type_mark	
unconstrained_array_defintion	
use_clause	
variable_assignment_statement	
variable_declaration	
wait_statement	
waveform	**Unaffected** is not supported.
waveform_element	Null waveform element is unsupported as it only has relevance in the context of guarded signals.

Verilog Language Coverage

Behavioral Statement Constructs

The following Verilog Behavioral Statements constructs are supported by the ISim as defined below.

Continuous Assignment Statements

Verilog Construct	ISim Support
continuous_assign	supported
list_of_net_assignments	supported
net_assignment	supported

Procedural Blocks and Assignments

Verilog Construct	ISim Support
initial_construct	supported
always_construct	supported
blocking_assignment	supported
nonblocking_assignment	supported
procedural_continuous_assignment	supported
function_blocking_assignment	supported
function_statement_or_null	supported

Parallel and Sequential Blocks and Assignments

Verilog Construct	ISim Support	Comment
function_seq_block	supported	
variable_assignment	supported	
par_block	partially supported	*Fork* and *join* constructs are not supported within tasks or functions
seq_block	supported	

Statements

Verilog Construct	ISim Support
statement	supported
statement_or_null	supported
function_statement	supported

Timing Control Statements

Verilog Construct	ISim Support
delay_control	supported
delay_or_event_control	supported
disable_statement	supported
event_control	supported
event_trigger	supported
event_expression	supported
procedural_timing_control_statement	supported
wait_statement	supported

Conditional Statements

Verilog Construct	ISim Support
conditional_statement	supported
if_else_if_statement	supported
function_conditional_statement	supported
function_if_else_if_statement	supported

Case Statements

Verilog Construct	ISim Support
case_statement	supported
case_item	supported
function_case_statement	supported
function_case_item	supported

Looping Statements

Verilog Construct	ISim Support
function_loop_statement	supported
loop_statement	supported

Task Enable Statements

Verilog Construct	ISim Support
system_task_enable	supported
task_enable	supported

Compiler Directive Constructs

The following Verilog Compiler Directives constructs are supported by the ISim as defined below.

Compiler Directive Constructs

Verilog Construct	ISim Support	Comment
‘celldefine	unsupported	
‘endcelldefine	unsupported	
‘default_nettype	supported	
‘define	supported	
‘undef	supported	ISim now supports parameterized ‘define macros.
‘ifdef	supported	
‘ifndef	supported	
‘elsif	supported	
‘else	supported	
‘endif	supported	

Verilog Construct	ISim Support	Comment
‘include	supported	
‘resetall	supported	
‘line	supported	
‘timescale	supported	
‘unconnected_drive	unsupported	
‘nounconnected_driv	unsupported	

Declaration Constructs

The following Verilog Declaration constructs are supported by the ISim as defined below.

Module Parameter Declarations

Verilog Construct	ISim Support
local_parameter_declaration	supported
parameter_declaration	supported
specparam_declaration	supported

Type Declarations

Verilog Construct	ISim Support
event_declaration	supported
genvar_declaration	supported
integer_declaration	supported
net_declaration	supported
real_declaration	supported
reg_declaration	supported
time_declaration	supported

Net Variable Types

Verilog Construct	ISim Support
net_type	supported
output_variable_type	supported
real_type	supported
variable_type	supported

Strengths

Verilog Construct	ISim Support
drive_strength	supported
strength0	supported
strength1	supported
charge_strength	unsupported

Delays

Verilog Construct	ISim Support
delay2	supported
delay3	supported
delay_value	supported

Declaration Lists

Verilog Construct	ISim Support
list_of_event_identifiers	supported
list_of_genvar_identifiers	supported
list_of_net_decl_assignments	supported
list_of_net_identifiers	supported
list_of_param_assignments	supported
list_of_port_identifiers	supported
list_of_real_identifiers	supported
list_of_specparam_assignments	supported
list_of_variable_identifiers	supported
list_of_variable_port_identifiers	supported

Declaration Assignments

Verilog Construct	ISim Support
net_decl_assignment	supported
param_assignment	supported
specparam_assignment	supported
pulse_control_specparam	supported
error_limit_value	supported
reject_limit_value	supported
limit_value	supported

Declaration Ranges

Verilog Construct	ISim Support
dimension	supported
range	supported

Function Declarations

Verilog Construct	ISim Support
function_declaration	supported
function_item_declaration	supported
function_port_list	supported
range_or_type	supported

Task Declarations

Verilog Construct	ISim Support
task_declaration	supported
task_item_declaration	supported
task_port_list	supported
task_port_item	supported
tf_input_declaration	supported
tf_output_declaration	supported
tf_inout_declaration	supported
task_port_type	supported

Block Item Declarations

Verilog Construct	ISim Support
block_item_declaration	supported
block_reg_declaration	supported
list_of_block_variable_identifiers	supported
block_variable_type	supported

Expression Constructs

The following Verilog Expression constructs are supported by the ISim as defined below.

Concatenations

Verilog Construct	ISim Support
concatenation	supported
constant_concatenation	supported
constant_multiple_concatenation	supported
module_path_concatenation	supported
module_path_multiple_concatenation	supported
multiple_concatenation	supported
net_concatenation	supported
net_concatenation_value	supported
variable_concatenation	supported
variable_concatenation_value	supported

Function Calls

Verilog Construct	ISim Support
constant_function_call	supported
function_call	supported
system_function_call	supported

Expressions

Verilog Construct	ISim Support
base_expression	supported
conditional_expression	supported
constant_base_expression	supported
constant_expression	supported
constant_mintypmax_expression	supported
constant_range_expression	supported
dimension_constant_expression	supported
expression1	supported
expression2	supported
expression3	supported
expression	supported
lsb_constant_expression	supported
mintypmax_expression	supported
module_path_conditional_expression	supported
module_path_expression	supported
module_path_mintypmax_expression	supported
msb_constant_expression	supported
range_expression	supported
width_constant_expression	supported

Primaries

Verilog Construct	ISim Support
constant_primary	supported
module_path_primary	supported
primary	supported

Expresson Left-Side Values

Verilog Construct	ISim Support
net_lvalue	supported
variable_lvalue	supported

Operators

Verilog Construct	ISim Support
unary_operator	supported
binary_operator	supported
unary_module_path_operator	supported
binary_module_path_operator	supported

Numbers

Verilog Construct	ISim Support
number	supported
real_number	supported
exp	supported
decimal_number	supported
binary_number	supported
octal_number	supported
hex_number	supported
sign	supported
size	supported
non_zero_unsigned_number	supported
unsigned_number	supported
binary_value	supported
octal_value	supported
hex_value	supported
decimal_base	supported
binary_base	supported
octal_base	supported
hex_base	supported
non_zero_decimal_digit	supported
decimal_digit	supported
binary_digit	supported
octal_digit	supported
hex_digit	supported
x_digit	supported
z_digit	supported

Strings

Verilog Construct	ISim Support
string	supported

General Constructs

The following Verilog general and miscellaneous constructs are support by the ISim as defined below.

Attributes

Verilog Construct	ISim Support
attribute_instance	unsupported
attr_spec	unsupported
attr_name	unsupported

Comments

Verilog Construct	ISim Support
comment	supported
one_line_comment	supported
block_comment	supported
comment_text	supported

Identifiers

Verilog Construct	ISim Support
arrayed_identifier	supported
block_identifier	supported
cell_identifier	unsupported
config_identifier	unsupported
escaped_arrayed_identifier	supported
escaped_hierarchical_identifier	supported
escaped_identifier	supported
event_identifier	supported
function_identifier	supported
gate_instance_identifier	supported
generate_block_identifier	supported
genvar_identifier	supported
genvar_function_identifier	unsupported
hierarchical_block_identifier	supported
hierarchical_event_identifier	supported
hierarchical_function_identifier	supported
hierarchical_identifier	supported
hierarchical_net_identifier	supported
hierarchical_variable_identifier	supported
hierarchical_task_identifier	supported
identifier	supported
inout_port_identifier	supported
input_port_identifier	supported
instance_identifier	supported

Verilog Construct	ISim Support
library_identifier	unsupported
memory_identifier	supported
module_identifier	supported
module_instance_identifier	supported
net_identifier	supported
output_port_identifier	supported
parameter_identifier	supported
port_identifier	supported
real_identifier	supported
simple_arrayed_identifier	supported
simple_hierarchical_identifier	supported
simple_identifier	supported
specparam_identifier	supported
system_function_identifier	supported
system_task_identifier	supported
task_identifier	supported
terminal_identifier	supported
text_macro_identifier	supported
topmodule_identifier	supported
udp_identifier	supported
udp_instance_identifier	supported
variable_identifier	supported

Identifier Branches

Verilog Construct	ISim Support
simple_hierarchical_branch	supported
escaped_hierarchical_branch	supported

White Space

Verilog Construct	ISim Support
white_space	supported

Primitive and Module Instance Constructs

The following Verilog primitive instance and module instance constructs are supported by the ISim as defined below.

Primitive Instantiations

Verilog Construct	ISim Support
gate_instantiation	supported
cmos_switch_instance	unsupported
enable_gate_instance	supported
mos_switch_instance	unsupported
n_input_gate_instance	supported
n_output_gate_instance	supported
pass_switch_instance	unsupported
pass_enable_switch_instance	unsupported
pull_gate_instance	supported
name_of_gate_instance	supported

Primitive Strengths

Verilog Construct	ISim Support
pulldown_strength	supported
pullup_strength	supported

Primitive Terminals

Verilog Construct	ISim Support
enable_terminal	supported
inout_terminal	supported
input_terminal	supported
ncontrol_terminal	supported
output_terminal	supported
pcontrol_terminal	supported

Primitive Gate and Switch Types

Verilog Construct	ISim Support
cmos_switchtype	unsupported
enable_gatetype	supported
mos_switchtype	unsupported
n_input_gatetype	supported
n_output_gatetype	supported
pass_en_switchtype	unsupported
pass_switchtype	unsupported

Module Instantiations

Verilog Construct	ISim Support	Comment
module_instantiation	supported	
parameter_value_assignment	supported	
list_of_parameter_assignments	supported	
ordered_parameter_assignment	supported	
named_parameter_assignment	supported	
module_instance	supported	
name_of_instance	partially supported	Module instance arrays are not supported
list_of_port_connections	supported	
ordered_port_connection	supported	
named_port_connection	supported	

Generated Instantiation

Verilog Construct	ISim Support	Comment
generated_instantiation	supported	
generate_item_or_null	partially supported	The module_or_generate_item alternative is not supported. Production from 1364-2001 Verilog standard: `generate_item_or_null ::= generate_conditional_statement	generate_case_statement	generate_loop_statement	generate_block	module_or_generate_item` Production supported by ISim: `generate_item_or_null ::= generate_conditional_statement	generate_case_statement	generate_loop_statement	generate_block`
generate_item	supported	
generate_conditional_statement	supported	
generate_case_statement	supported	
generate_case_item	supported	
generate_loop_statement	supported	
genvar_assignment	partially supported	All generate blocks must be named. Production from 1364-2001 Verilog standard: `generate_block ::= begin [ : generate_block_identifier ] { generate_item } end` Production supported by ISim:

Verilog Construct	ISim Support	Comment
		generate_block ::= begin : generate_block_identifier { generate_item } end

Source Text Constructs

The following Verilog source text constructs are supported by the ISim as defined below.

Library Source Text

Verilog Construct	ISim Support	Comment
library_text	unsupported	
library_descriptions	unsupported	
library_declaration	unsupported	
file_path_spec	unsupported	
include_statement	unsupported	This refers to include statements within library map files (See IEEE 1364-2001, section 13.2). This does *not* refer to the ‘include compiler directive.

Configuration Source Text

Verilog Construct	ISim Support
config_declaration	unsupported
design_statement	unsupported
config_rule_statement	unsupported
default_clause	unsupported
inst_clause	unsupported
inst_name	unsupported
cell_clause	unsupported
liblist_clause	unsupported
use_clause	unsupported

Module and Primitive Source Text

ISim Support	ISim Support
source_text	supported
description	supported
module_declaration	supported
module_keyword	supported

Module Parameters and Ports

Verilog Construct	ISim Support
module_parameter_port_list	supported
list_of_ports	supported
list_of_port_declarations	supported
port	supported
port_expression	supported
port_reference	supported
port_declaration	supported

Module Items

Verilog Construct	ISim Support
module_item	supported
module_or_generate_item	supported
module_or_generate_item_declaration	supported
non_port_module_item	supported
parameter_override	supported

Specify Function Constructs

The following Verilog Specify function constructs are supported by the ISim as defined below.

Specify Block Declarations

Verilog Construct	ISim Support
specify_block	supported
specify_item	supported
pulsestyle_declaration	supported
showcancelled_declaration	supported

Specify Path Declarations

Verilog Construct	ISim Support
path_declaration	supported
simple_path_declaration	supported
parallel_path_declaration	supported
full_path_description	supported
list_of_path_inputs	supported
list_of_path_outputs	supported

Specify Block Terminals

Verilog Construct	ISim Support
specify_input_terminal_descriptor	supported
specify_output_terminal_descriptor	supported
input_identifier	supported
output_identifier	supported

Specify Path Delays

Verilog Construct	ISim Support
path_delay_value	supported
list_of_path_delay_expressions	supported
t_path_delay_expression	supported
trise_path_delay_expression	supported
tfall_path_delay_expression	supported
tz_path_delay_expression	supported
t01_path_delay_expression	supported
t10_path_delay_expression	supported
t0z_path_delay_expression	supported
tz1_path_delay_expression	supported
t1z_path_delay_expression	supported
tz0_path_delay_expression	supported
t0x_path_delay_expression	supported
tx1_path_delay_expression	supported
t1x_path_delay_expression	supported
tx0_path_delay_expression	supported
txz_path_delay_expression	supported
tzx_path_delay_expression	supported
path_delay_expression	supported
edge_sensitive_path_declaration	supported
parallel_edge_sensitive_path_declaration	supported
full_edge_sensitive_path_declaration	supported
data_source_expression	supported
edge_identifier	supported
state_dependent_path_declaration	supported
polarity_operator	supported

System Timing Check Commands

Verilog Construct	ISim Support
system_timing_check	supported
$hold_timing_check	supported
$setuphold_timing_check	supported
$recovery_timing_check	supported
$removal_timing_check	supported
$recrem_timing_check	supported
$skew_timing_check	unsupported
$timeskew_timing_check	unsupported
$fullskew_timing_check	unsupported
$period_timing_check	supported
$width_timing_check	supported
$nochange_timing_check	unsupported

System Timing Check Command Arguments

Verilog Construct	ISim Support
checktime_condition	unsupported
controlled_reference_event	supported
data_event	supported
delayed_data	unsupported
delayed_reference	unsupported
end_edge_offset	unsupported
event_based_flag	unsupported
notify_reg	supported
reference_event	supported
remain_active_flag	unsupported
stamptime_condition	unsupported
start_edge_offset	unsupported
threshold	supported
timing_check_limit	supported

System Timing Check Event Definitions

Verilog Construct	ISim Support
timing_check_event	supported
controlled_timing_check_event	supported
timing_check_event_control	supported
specify_terminal_descriptor	supported
edge_control_specifier	supported
edge_descriptor	supported
zero_or_one	supported
z_or_x	supported
timing_check_condition	supported
scalar_timing_check_condition	supported
scalar_constant	supported

System Task and Function Constructs

The following Verilog System Tasks and Functions constructs are supported by the ISim as defined below.

Display System Tasks

Verilog Construct	ISim Support
$display	supported
$displayb	supported
$displayh	supported
$displayo	supported
$monitor	supported
$monitorb	supported
$monitorh	supported
$monitoro	supported
$monitoroff	supported
$monitoron	supported
$strobe	supported
$strobeb	supported
$strobeh	supported
$strobeo	supported
$write	supported
$writeb	supported
$writeh	supported
$writeo	supported

File I/O Tasks

Verilog Construct	ISim Support
$fclose	supported
$fdisplay	supported
$fdisplayb	supported
$fdisplayh	supported
$fdisplayo	supported
$ferror	supported
$fflush	supported
$fgetc	supported
$fgets	supported
$fmonitor	supported
$fmonitorb	supported
$fmonitorh	supported
$fmonitoro	supported
$fopen	supported
$fread	supported
$fscanf	supported
$fseek	supported
$fstrobe	supported
$fstrobeb	supported
$fstrobeh	supported
$fstrobeo	supported
$ftell	supported
$fwrite	supported
$fwriteb	supported
$fwriteh	supported
$fwriteo	supported
$readmemb	supported
$readmemh	supported
$rewind	supported
$sdf_annotate	supported
$sformat	supported
$sscanf	supported
$swrite	supported
$swriteb	supported
$swriteh	supported
$swriteo	supported
$ungetc	supported

Timescale Tasks

Verilog Construct	ISim Support
$printtimescale	supported
$timeformat	supported

Simulation Control Tasks

Verilog Construct	ISim Support
$finish	supported
$stop	supported

PLA Modeling Tasks

Verilog Construct	ISim Support
$async$and$array	unsupported
$async$nand$array	unsupported
$async$nor$array	unsupported
$async$or$array	unsupported
$sync$and$array	unsupported
$sync$nand$array	unsupported
$sync$nor$array	unsupported
$sync$or$array	unsupported
$async$and$plane	unsupported
$async$nand$plane	unsupported
$async$nor$plane	unsupported
$async$or$plane	unsupported
$sync$and$plane	unsupported
$sync$nand$plane	unsupported
$sync$nor$plane	unsupported
$sync$or$plane	unsupported

Stochastic Analysis Tasks

Verilog Construct	ISim Support
$q_add	supported
$q_exam	supported
$q_full	supported
$q_initialize	supported
$q_remove	supported

Simulation Time Functions

Verilog Construct	ISim Support
$realtime	supported
$stime	supported
$time	supported

Conversion Functions

Verilog Construct	ISim Support
$bitstoreal	supported
$realtobits	supported
$itor	supported
$rtoi	supported
$signed	supported
$unsigned	supported

Probabilistic Distribution Functions

Verilog Construct	ISim Support
$dist_chi_square	supported
$dist_erlang	supported
$dist_exponential	supported
$dist_normal	supported
$dist_poisson	supported
$dist_t	supported
$dist_uniform	supported
$random	supported

Command Line Input

Verilog Construct	ISim Support
$test$plusargs	supported
$value$plusargs	supported

Value Change Dump (VCD) Files

Verilog Construct	ISim Support
$dumpall	supported
$dumpfile	supported
$dumpflush	supported
$dumplimit	supported
$dumpoff	supported
$dumpon	supported
$dumpports	unsupported
$dumpportsall	unsupported
$dumpportsflush	unsupported
$dumpportslimit	unsupported
$dumpportsoff	unsupported
$dumpportson	unsupported
$dumpvars	supported

UDP Declaration and Instantiation Constructs

The following Verilog UDP instantiation constructs are supported by the ISim as defined below.

UDP Declaration

Verilog Construct	ISim Support
udp_declaration	supported

UDP Ports

Verilog Construct	ISim Support
udp_port_list	supported
udp_declaration_port_list	supported
udp_port_declaration	supported
udp_output_declaration	supported
udp_input_declaration	supported
udp_reg_declaration	supported

UDP Body

Verilog Construct	ISim Support
udp_body	supported
combinational_body	supported
combinational_entry	supported
sequential_body	supported
udp_initial_statement	supported
init_val	supported
sequential_entry	supported
seq_input_list	supported
level_input_list	supported
edge_input_list	supported
edge_indicator	supported
current_state	supported
next_state	supported
output_symbol	supported
level_symbol	supported
edge_symbol	supported

UDP Instantiation

Verilog Construct	ISim Support	Comment
udp_instantiation	supported	
udp_instance	supported	
name_of_udp_instance	partially supported	UDP instance arrays are not supported

XILINX®

Appendix B

Migrating from ModelSim XE to ISim

Migration Overview

Retargeting from a ModelSim XE simulation environment to an 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 a tutorial are also available.

- ISim Tutorial - http://www.xilinx.com/support/documentation/dt_ise.htm
- ISim Video Demo - http://www.xilinx.com/products/design_resources/design_tool/resources/index.htm
- ISim Product Page - http://www.xilinx.com/tools/isim.htm
- ModelSim XE Library Download - http://www.xilinx.com/support/download/index.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 12.4 software 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, and comes in 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 Project Navigator, Embedded Design Kit, and System Generator.

ISim is available with all major Xilinx ISE® Design Suite releases, and comes in two versions:

- **ISim Lite -** a limited version of the ISE Simulator. In this version, when your design plus testbench 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.

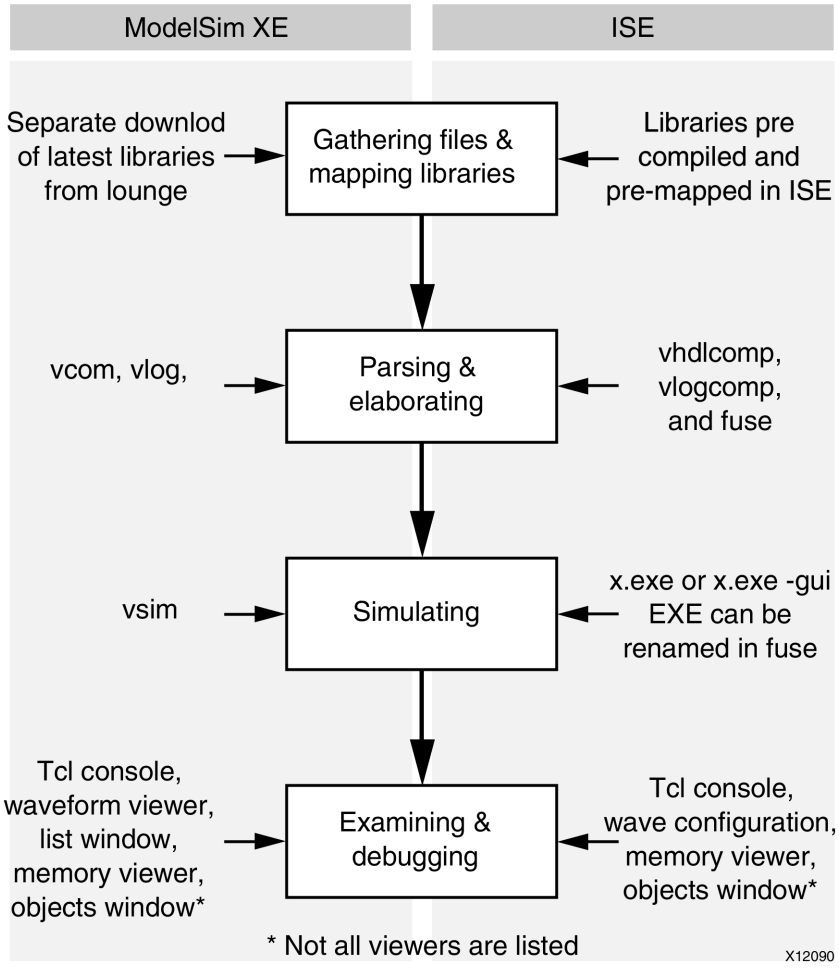
Feature comparison

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
SystemVerilog for Design	No	No	Roadmap	Roadmap
SystemVerilog for Verification	No	No	Roadmap	Roadmap
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	Roadmap	Roadmap
VHDL FLI/VHPI	No	No	Roadmap	Roadmap
Code Coverage	No	No	Roadmap	Roadmap
SecureIP/HardIP Support	No	No	Yes	Yes
EDK Support	No	No	Yes	Yes
System Generator Support	No	No	Yes	Yes
CoreGen Support	Yes	Yes	Yes	Yes
MIG Support	No	No	Yes	Yes
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.

The following figure shows the different steps in simulation and the process for each step.



Step 1: Gathering Files and Mapping Libraries

ModelSim XE Flow

The ModelSim XE libraries can be downloaded from http://www.xilinx.com/support/download/index.htm.

Each time a new release of 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 Simulation Executable Command Overview.

ISim Flow

ModelSim XE uses the following commands for compilation and elaboration.

`vhpcomp` ***`options`*** (VHDL Compiler) runs the VHDL compiler and compiles VHDL files to a specified directory.

`vlogcomp` ***`options`*** (Verilog Compiler) runs the Verilog compiler and compiles Verilog files to a specified directory.

`fuse` ***`options`*** (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.

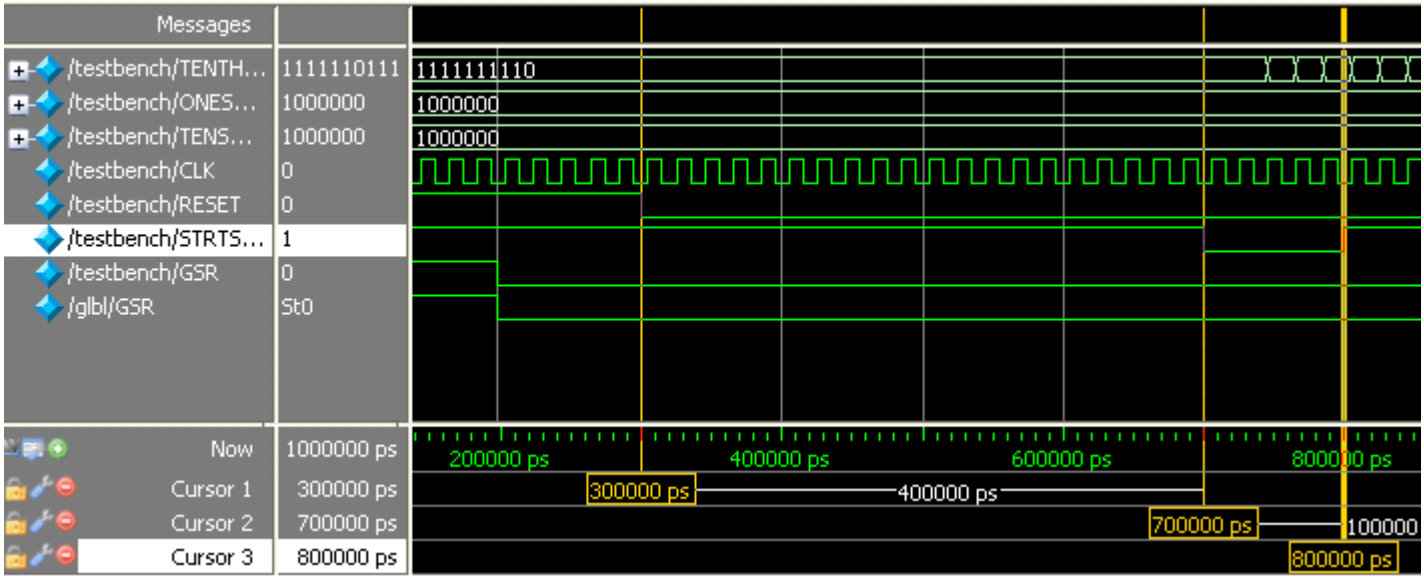
Note ISim does not have Tcl support for all wave configuration operations.

Measuring with Markers and Cursors

Measuring with markers and cursors is different between ModelSim XE and ISim.

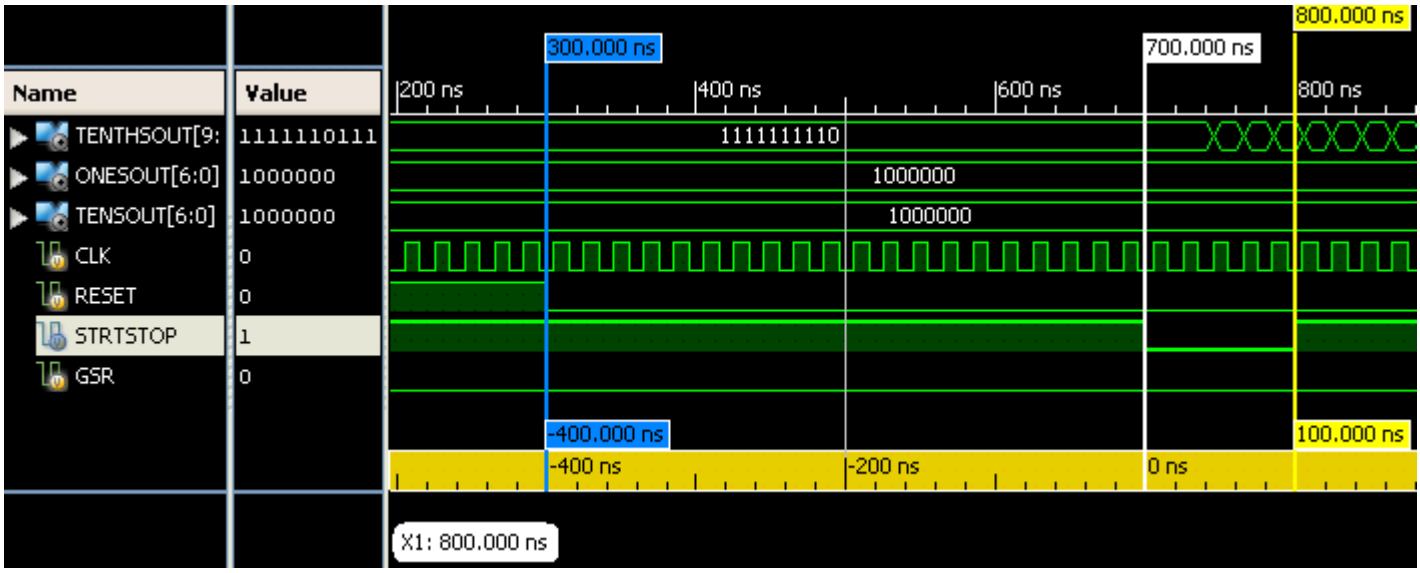
ModelSim XE provides cursors to measure between two points of interest. You can add cursors as needed, and each new cursor gets added under the existing cursors. The waveform viewer automatically shows the distance between the cursors.

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.

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. This image 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

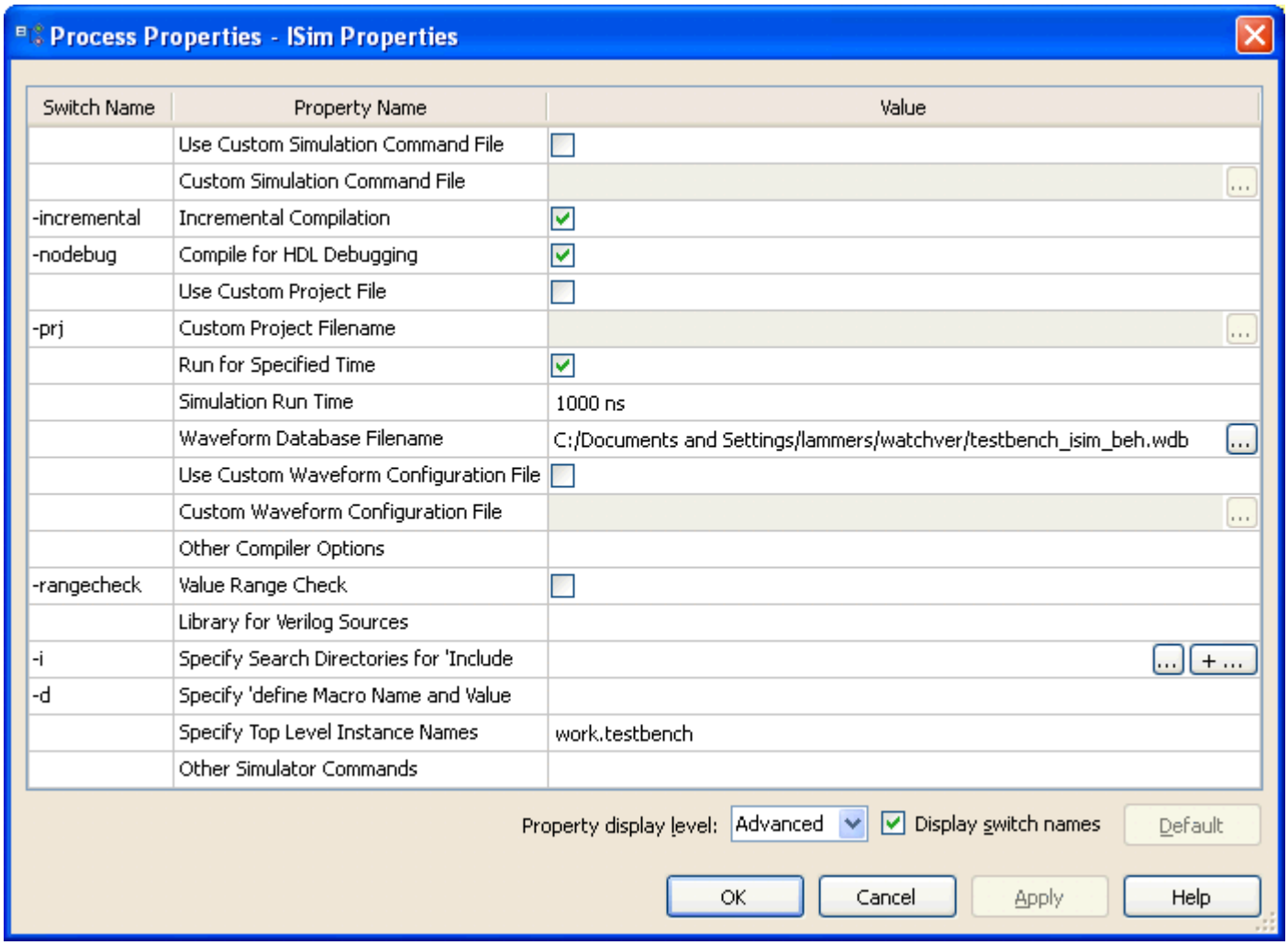
ModelSim XE has a true text editor built into the standalone GUI which lets you make changes to HDL code, recompile, and re-simulate.

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® Design Suite Project Navigator text editor, then re-launch the simulation in ISim.

Project Navigator Integration

Project Navigator notifies you when ModelSim XE is not a valid simulator choice, and that you should select one of the other integrated simulators.

Project Navigator Project Settings Dialog Box with ISim Selected



Simulation Properties in Project Navigator

The simulation properties are similar between ModelSim XE and ISim; the following section lists the differences.

Library Compilation

ModelSim XE Property Name	ISim Property Name	Comments
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

ModelSim XE Property Name	ISim Property Name	Comments
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	Wavform Database Filename	Lets you specify a different database for the simulation.

Custom Compiler Commands

ModelSim XE Property Name	ISim Property Name	Comments
Other VSIM Command Line Options	Other Compiler Options Other Simulator Commands	ISim splits the VSIM commands into fuse commands and executable commands.
Other VLOG Command Line Options	Other Compiler Options	Passes the options to the `fuse` command in ISim.
Other VCOM Command Line Options		

Runtime settings

ModelSim XE Property Name	ISim Property Name	Comments
Simulation Run Time	Simulation Run Time	
Simulation Resolution	N/A	ISim defaults to 1ps.

Language Settings

ModelSim XE Property Name	ISim Property Name	Comments
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.
Other VLOG Command Line Options	Specify Search Directories for ‘include Incremental Compilation	
Other VLOG Command Line Options	Specify ‘define Macro Name and Value Incremental Compilation	
N/A	Compile for HDL Debugging	

Miscellaneous Settings

ModelSim XE Property Name	ISim Property Name	Comments
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 Names	

XILINX®

Appendix C

Additional Resources

- **Glossary of Terms -** http://www.xilinx.com/support/documentation/sw_manuals/glossary.pdf
- **Xilinx Support -**http://www.xilinx.com/support
- *ML506 Evaluation Platform User Guide (UG347)*
- *ISE Design Suite: Installation and Licensing Guide (UG798)*.
- *ISE Hardware Co-Simulation Tutorial: Accelerating Floating Point FFT Simulation (UG817)*
- *ISE Hardware Co-Simulation Tutorial: Processing Live Ethernet Traffic through Virtex-5 Embedded Ethernet MAC, (UG819)*
- *XPower Estimator (UG440)*
- *Synthesis and Simulation Design Guide (UG626)*
- *ChipScope Pro Software and Cores User Guide (UG029)*