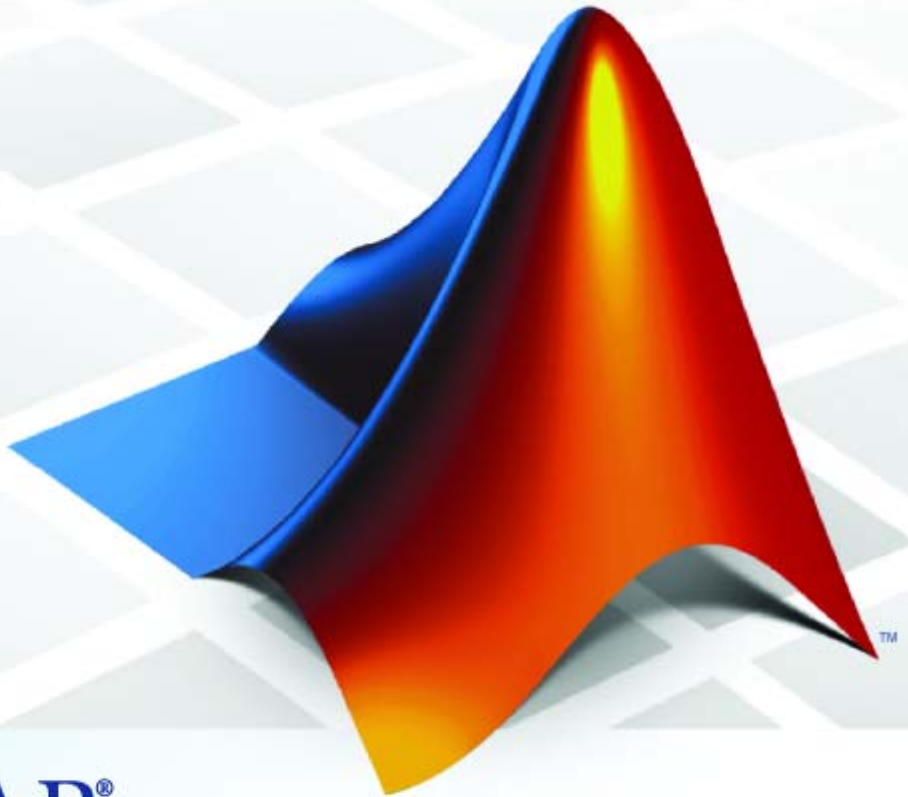


EDA Simulator Link™ 3 Reference



MATLAB®
& **SIMULINK®**

How to Contact The MathWorks



www.mathworks.com Web
comp.soft-sys.matlab Newsgroup
www.mathworks.com/contact_TS.html Technical Support



suggest@mathworks.com Product enhancement suggestions
bugs@mathworks.com Bug reports
doc@mathworks.com Documentation error reports
service@mathworks.com Order status, license renewals, passcodes
info@mathworks.com Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

EDA Simulator Link™ Reference

© COPYRIGHT 2003–2010 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

The MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

August 2003	Online only	New for Version 1 (Release 13SP1)
February 2004	Online only	Updated for Version 1.1 (Release 13SP1)
June 2004	Online only	Updated for Version 1.1.1 (Release 14)
October 2004	Online only	Updated for Version 1.2 (Release 14SP1)
December 2004	Online only	Updated for Version 1.3 (Release 14SP1+)
March 2005	Online only	Updated for Version 1.3.1 (Release 14SP2)
September 2005	Online only	Updated for Version 1.4 (Release 14SP3)
March 2006	Online only	Updated for Version 2.0 (Release 2006a)
September 2006	Online only	Updated for Version 2.1 (Release 2006b)
March 2007	Online only	Updated for Version 2.2 (Release 2007a)
September 2007	Online only	Updated for Version 2.3 (Release 2007b)
March 2008	Online only	Updated for Version 2.4 (Release 2008a)
October 2008	Online only	Updated for Version 2.5 (Release 2008b)
March 2009	Online only	Updated for Version 2.6 (Release 2009a)
September 2009	Online only	Updated for Version 3.0 (Release 2009b)
March 2010	Online only	Updated for Version 3.1 (Release 2010a)

Block Reference

1

HDL Cosimulation	1-2
FPGA Implementations	1-3
Virtual Platform Simulation	1-4

Blocks — Alphabetical List

2

Function Reference

3

HDL Cosimulation	3-2
FPGA Implementations	3-4
Virtual Platform Simulation	3-5

Functions — Alphabetical List

4

Index

Block Reference

HDL Cosimulation (p. 1-2)	Describes the EDA Simulator Link™ Simulink blocks available for use with HDL cosimulation
FPGA Implementations (p. 1-3)	Describes the EDA Simulator Link Simulink blocks available for use with generating FPGA implementations
Virtual Platform Simulation (p. 1-4)	Describes the EDA Simulator Link Simulink blocks available for use with generating Virtual Platform simulations

HDL Cosimulation

HDL Cosimulation

Cosimulate hardware component by communicating with HDL module instance executing in HDL simulator

To VCD File

Generate value change dump (VCD) file

FPGA Implementations

Currently, there are no EDA Simulator Link Simulink blocks available for use with generating FPGA implementations.

Virtual Platform Simulation

Currently, there are no EDA Simulator Link Simulink blocks available for use with generating Virtual Platform simulations.

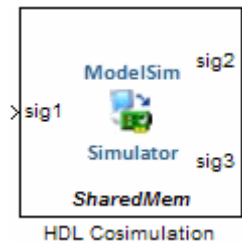
Blocks — Alphabetical List

HDL Cosimulation

Purpose Cosimulate hardware component by communicating with HDL module instance executing in HDL simulator

Library EDA Simulator Link

Description The HDL Cosimulation block cosimulates a hardware component by applying input signals to and reading output signals from an HDL model under simulation in the HDL simulator. You can use this block to model a source or sink device by configuring the block with input or output ports only.



The tabbed panes on the block's dialog box let you configure:

- Block input and output ports that correspond to signals (including internal signals) of an HDL module. You must specify a sample time for each output port; you can also specify a data type for each output port.
- Type of communication and communication settings used to exchange data between simulators.
- The timing relationship between units of simulation time in Simulink and the HDL simulator.
- Rising-edge or falling-edge clocks to apply to your model (Incisive and ModelSim users only; Discovery users see `launchDiscovery`). You can specify the period for each clock signal.
- Tcl commands to run before and after the simulation (Incisive and ModelSim users only; Discovery users see `launchDiscovery`).

The HDL Cosimulation Block Panes

The **Ports** pane provides fields for mapping signals of your HDL design to input and output ports in your block. The signals can be at any level of the HDL design hierarchy.

The **Timescales** pane lets you choose an optimal timing relationship between Simulink and the HDL simulator. You can configure either of the following timing relationships:

- *Relative* timing relationship (Simulink seconds correspond to an HDL simulator-defined tick interval)
- *Absolute* timing relationship (Simulink seconds correspond to an absolute unit of HDL simulator time)

The **Connection** pane specifies the communications mode used between Simulink and the HDL simulator. If you use TCP socket communication, this pane provides fields for specifying a socket port and for the host name of a remote computer running the HDL simulator. The **Connection** pane also provides the option for bypassing the cosimulation block during Simulink simulation.

The **Clocks** pane lets you create optional rising-edge and falling-edge clocks that apply stimuli to your cosimulation model.

The **Tcl** pane provides a way of specifying tools command language (Tcl) commands to be executed before and after the HDL simulator simulates the HDL component of your Simulink model. You can use the **Pre-simulation commands** field on this pane for simulation initialization and startup operations, but you cannot use it to change simulation state.

Note You must make sure that signals being used in cosimulation have read/write access. This rule applies to all signals on the **Ports**, **Clocks**, and **Tcl** panes.

Incisive and ModelSim users: Verify such access through the HDL simulator—see product documentation for details.

Discovery users: A tab file is included in the simulation via the required launchDiscovery property "AccFile".

Dialog Box

The Block Parameters dialog box consists of the following tabbed panes of configuration options:

- “Ports Pane” on page 2-4
- “Connection Pane” on page 2-11
- “Timescales Pane” on page 2-15
- “Clocks Pane” on page 2-19 (Incisive and ModelSim users only)
- “Tcl Pane” on page 2-22 (Incisive and ModelSim users only)

Ports Pane

Specify fields for mapping signals of your HDL design to input and output ports in your block. Simulink deposits an input port signal on an HDL simulator signal at the signal’s sample rate. Conversely, Simulink reads an output port signal from a specified HDL simulator signal at the specified sample rate.

In general, Simulink handles port sample periods as follows:

- If you connect an input port to a signal that has an explicit sample period, based on forward propagation, Simulink applies that rate to the port.
- If you connect an input port to a signal that does not have an explicit sample period, Simulink assigns a sample period that is equal to the least common multiple (LCM) of all identified input port sample periods for the model.
- After Simulink sets the input port sample periods, it applies user-specified output sample times to all output ports. You must specify an explicit sample time for each output port.

In addition to specifying output port sample times, you can force the fixed-point data types on output ports. For example, setting the **Data Type** property of an 8-bit output port to **Signed** and setting its **Fraction Length** property to 5 would force the data type to `sfixed8_En5`.

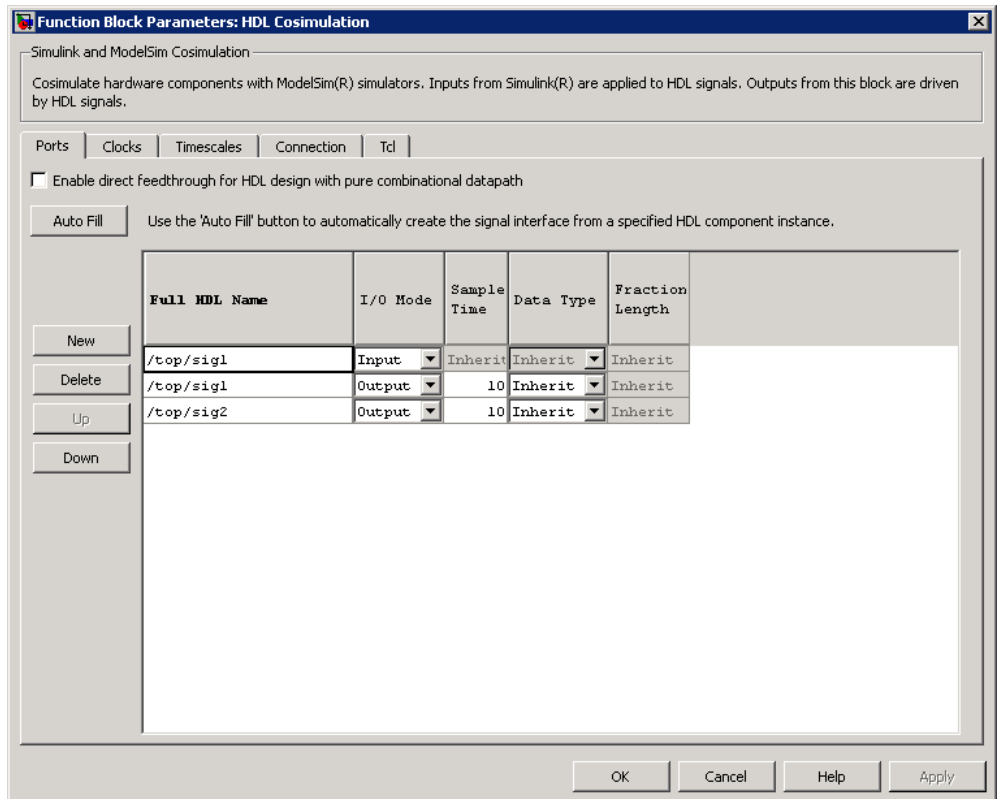
You can not force width; the width is always inherited from the HDL simulator.

Note The **Data Type** and **Fraction Length** properties apply only to the following signals:

- VHDL signals of any logic type, such as `STD_LOGIC` or `STD_LOGIC_VECTOR`
 - Verilog signals of `wire` or `reg` type
-

You can set input/output ports in the **Ports** pane also. To do so, specify port as both input and output (example shown for use with ModelSim).

HDL Cosimulation



If your model contains purely combinational paths, you can select **Enable direct feedthrough for HDL design with pure combinational datapath** to eliminate the one output-sample delay that occurs with using EDA Simulator Link blocks and Simulink. For more information on block simulation latency and using the direct feedthrough feature to eliminate it, see “Eliminating Block Simulation Latency”.

Discovery Users You may not enable direct feedthrough if your design contains mixed HDL (VHDL and Verilog). If you do, EDA Simulator Link will display an error in the HDL simulator.

The list at the center of the pane displays HDL signals corresponding to ports on the HDL Cosimulation block. Maintain this list with the buttons on the left of the pane:

- **Auto Fill** — Transmit a port information request to the HDL simulator. The port information request returns port names and information from an HDL model (or module) under simulation in the HDL simulator and automatically enters this information into the ports list. See “Obtaining Signal Information Automatically from the HDL Simulator” for a detailed description of this feature.
- **New** — Add a new signal to the list and select it for editing.
- **Delete** — Remove a signal from the list.
- **Up** — Move the selected signal up one position in the list.
- **Down** — Move the selected signal down one position in the list.

To commit edits to the Simulink model, you must also click **Apply** after selecting parameter values.

Note When you import VHDL signals from the HDL simulator, EDA Simulator Link returns the signal names in all capitals.

To edit a signal name, double-click on the name. Set the signal properties on the same line and in the appropriate columns. The properties of a signal are as follows.

Full HDL Name

Specifies the signal path name, using the HDL simulator path name syntax. For example (for use with Incisive), a path name for an input port might be `manchester.samp`. The signal can be at any level of the HDL design hierarchy. The HDL Cosimulation block port corresponding to the signal is labeled with the **Full HDL Name**.

For rules on specifying signal/port and module path specifications in Simulink, see “Specifying HDL Signal/Port and Module Paths for Cosimulation”.

Copying Signal Path Names (For Incisive and ModelSim Users)

You can copy signal path names directly from the HDL simulator **wave** window and paste them into the **Full HDL Name** field, using the standard copy and paste commands in the HDL simulator and Simulink. You must use the `Path.Name` view and not `Db::Path.Name` view. After pasting a signal path name into the **Full HDL Name** field, you must click the **Apply** button to complete the paste operation and update the signal list.

I/O Mode

Select either **Input**, **Output**, or both (“both” applies to Incisive and ModelSim users only).

Input designates signals of your HDL module that Simulink will drive. Simulink deposits values on the specified the HDL simulator signal at the signal’s sample rate.

Note When you define a block input port, make sure that only one source is set up to drive input to that signal. For example, you should avoid defining an input port that has multiple instances. If multiple sources drive input to a single signal, your simulation model may produce unexpected results.

Output designates signals of your HDL module that Simulink will read. For output signals, you must specify an explicit sample time. You can also specify any data type (except width). For details on specifying a data type, see Data Type and Fraction Length in a following section.

Because Simulink signals do not have the semantic of tri-states (there is no 'Z' value), you will gain no benefit by connecting to a bidirectional HDL signal directly. To interface with bidirectional signals, you can first interface to the input of the output driver, then the enable of the output driver and the output of the input driver. This approach leaves the actual tri-state buffer in HDL where resolution functions can handle interfacing with other tri-state buffers.

Sample Time

This property becomes available only when you specify an output signal. You must specify an explicit sample time.

Sample Time represents the time interval between consecutive samples applied to the output port. The default sample time is 1. The exact interpretation of the output port sample time depends on the settings of the **Timescales** pane of the HDL Cosimulation block. See also “Understanding the Representation of Simulation Time”.

Data Type

Fraction Length

These two related parameters apply only to output signals.

HDL Cosimulation

The **Data Type** property is enabled only for output signals. You can direct Simulink to determine the data type, or you can assign an explicit data type (with option fraction length). By explicitly assigning a data type, you can force fixed-point data types on output ports of an HDL Cosimulation block.

The **Fraction Length** property specifies the size, in bits, of the fractional part of the signal in fixed-point representation. **Fraction Length** becomes available if you do not set the **Data Type** property to **Inherit**.

The data type specification for an output port depends on the signal width and by the **Data Type** and **Fraction Length** properties of the signal.

Note The **Data Type** and **Fraction Length** properties apply only to the following signals:

- VHDL signals of any logic type, such as `STD_LOGIC` or `STD_LOGIC_VECTOR`
 - Verilog signals of wire or reg type
-

To assign a port data type, set the **Data Type** and **Fraction Length** properties as follows:

- Select **Inherit** from the **Data Type** list if you want Simulink to determine the data type.

This property defaults to **Inherit**. When you select **Inherit**, the **Fraction Length** edit field becomes unavailable.

Simulink always double checks that the word-length back propagated by Simulink matches the word length queried from the HDL simulator. If they do not match, Simulink generates an error message. For example, if you connect a Signal

Specification block to an output, Simulink will force the data type specified by Signal Specification block on the output port.

If Simulink cannot determine the data type of the signal connected to the output port, it will query the HDL simulator for the data type of the port. As an example, if the HDL simulator returns the VHDL data type `STD_LOGIC_VECTOR` for a signal of size N bits, the data type `ufixN` is forced on the output port. (The implicit fraction length is 0.)

- Select **Signed** from the **Data Type** list if you want to explicitly assign a signed fixed point data type. When you select **Signed**, the **Fraction Length** edit field becomes available. EDA Simulator Link assigns the port a fixed-point type `sfixN_EnF`, where N is the signal width and F is the **Fraction Length**.

For example, if you specify **Data Type** as **Signed** and a **Fraction Length** of 5 for a 16-bit signal, Simulink forces the data type to `sfix16_En5`. For the same signal with a **Data Type** set to **Signed** and **Fraction Length** of -5, Simulink forces the data type to `sfix16_E5`.

- Select **Unsigned** from the **Data Type** list if you want to explicitly assign an unsigned fixed point data type. When you select **Unsigned**, the **Fraction Length** edit field becomes available. EDA Simulator Link assigns the port a fixed-point type `ufixN_EnF`, where N is the signal width and F is the **Fraction Length**.

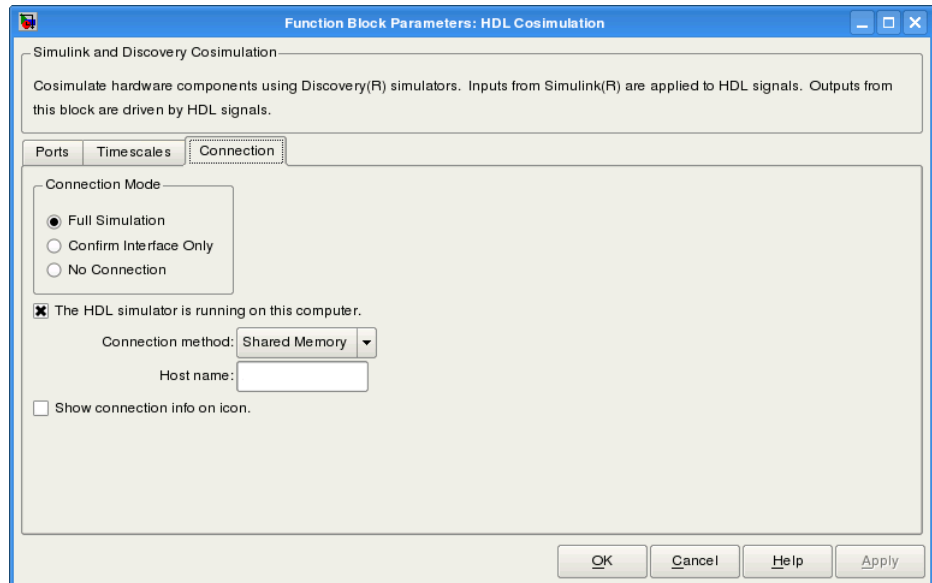
For example, if you specify **Data Type** as **Unsigned** and a **Fraction Length** of 5 for a 16-bit signal, Simulink forces the data type to `ufix16_En5`. For the same signal with a **Data Type** set to **Unsigned** and **Fraction Length** of -5, Simulink forces the data type to `ufix16_E5`.

Connection Pane

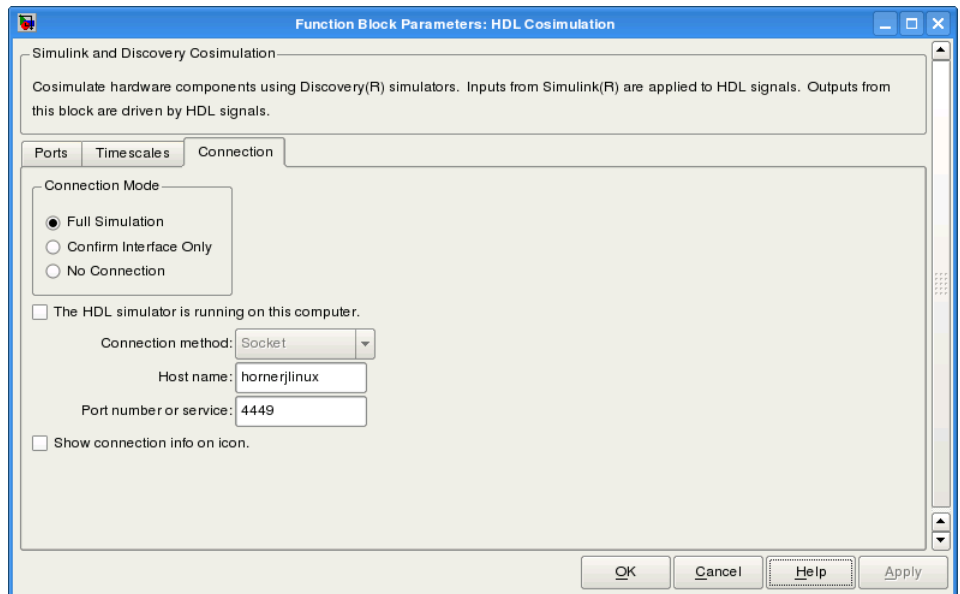
This figure shows the default configuration of the **Connection** pane (example shown is for use with Discovery). The block defaults to a

HDL Cosimulation

shared memory configuration for communication between Simulink and the HDL simulator, when they run on a single computer.



If you select TCP/IP socket mode communication, the pane displays additional properties, as shown in the following figure.



Connection Mode

If you want to bypass the HDL simulator when you run a Simulink simulation, use these options to specify what type of simulation connection you want. Select one of the following options:

- **Full Simulation:** Confirm interface and run HDL simulation (default).
- **Confirm Interface Only:** Connect to the HDL simulator and check for proper signal names, dimensions, and data types, but do not run HDL simulation.
- **No Connection:** Do not communicate with the HDL simulator. The HDL simulator does not need to be started.

With the second and third options, the EDA Simulator Link cosimulation interface does not communicate with the HDL simulator during Simulink simulation.

The HDL Simulator is running on this computer

Select this option if you want to run Simulink and the HDL simulator on the same computer. When both applications run on the same computer, you have the choice of using shared memory or TCP sockets for the communication channel between the two applications. If you do not select this option, only TCP/IP socket mode is available, and the **Connection method** list becomes unavailable.

Connection method

This list becomes available when you select **The HDL Simulator is running on this computer**. Select **Socket** if you want Simulink and the HDL simulator to communicate via a designated TCP/IP socket. Select **Shared memory** if you want Simulink and the HDL simulator to communicate via shared memory. For more information on these connection methods, see “Communications for HDL Cosimulation”.

Host name

If you run Simulink and the HDL simulator on different computers, this text field becomes available. The field specifies the host name of the computer that is running your HDL simulation in the HDL simulator.

Port number or service

Indicate a valid TCP socket port number or service for your computer system (if not using shared memory). For information on choosing TCP socket ports, see “Choosing TCP/IP Socket Ports”.

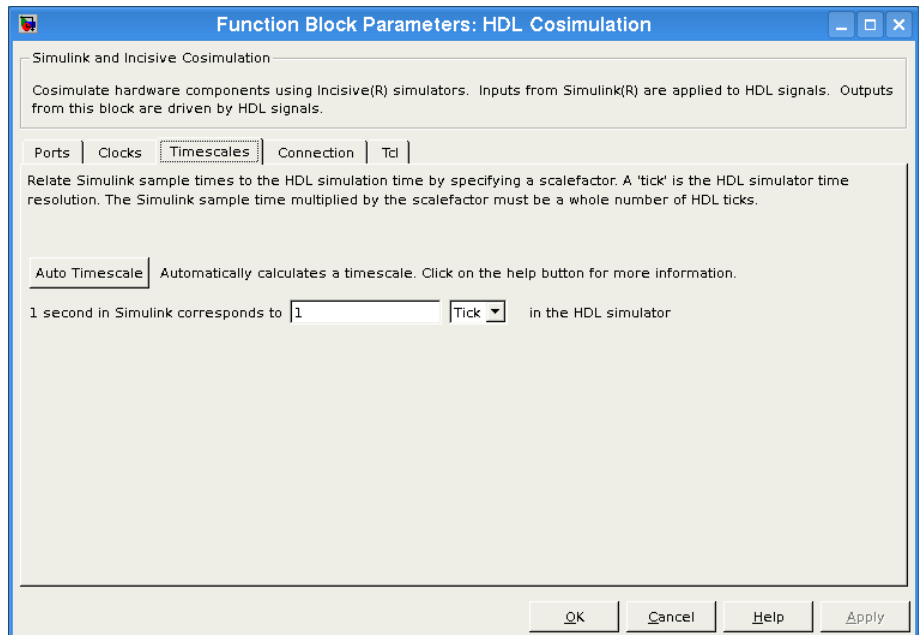
Show connection info on icon

When you select this option, Simulink indicates information about the selected communication method and (if applicable) communication options information on the HDL Cosimulation block icon. If you select shared memory, the icon displays the string **SharedMem**. If you select TCP socket communication, the icon displays the string **Socket** and displays the host name and port number in the format **hostname:port**.

In a model that has multiple HDL Cosimulation blocks, with each communicating to different instances of the HDL simulator in different modes, this information helps to distinguish between different cosimulation sessions.

Timescales Pane

The **Timescales** pane of the HDL Cosimulation block parameters dialog box lets you choose a timing relationship between Simulink and the HDL simulator, either manually or automatically. The following figure shows the default settings of the **Timescales** pane (example shown for use with Incisive).



The **Timescales** pane specifies a correspondence between one second of Simulink time and some quantity of HDL simulator time. This quantity of HDL simulator time can be expressed in one of the following ways:

HDL Cosimulation

- Using *relative timing mode*. EDA Simulator Link defaults to relative timing mode.
- Using *absolute timing mode*

For more information on calculating relative and absolute timing modes, see “Defining the Simulink and HDL Simulator Timing Relationship”.

For detailed information on the relationship between Simulink and the HDL simulator during cosimulation, and on the operation of relative and absolute timing modes, see “Understanding the Representation of Simulation Time”.

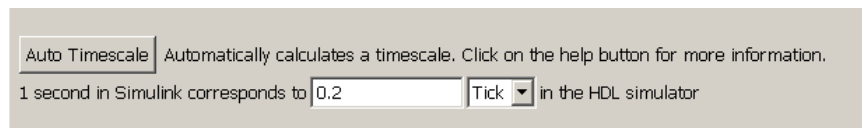
The following sections describe how to specify the timing relationship, either automatically or manually.

Automatically Specifying the Timing Relationship

You can have the EDA Simulator Link software calculate the timing relationship for you by performing the following steps:

- 1** Verify that the HDL simulator is running. EDA Simulator Link software can get the resolution limit of the HDL simulator only when that simulator is running.
- 2** Click on **Auto Timescale**.

The following graphic shows the result of clicking **Auto Timescale** in the **Timescales** pane of the HDL Cosimulation block in the Manchester Receiver demo (example shown for use with ModelSim).

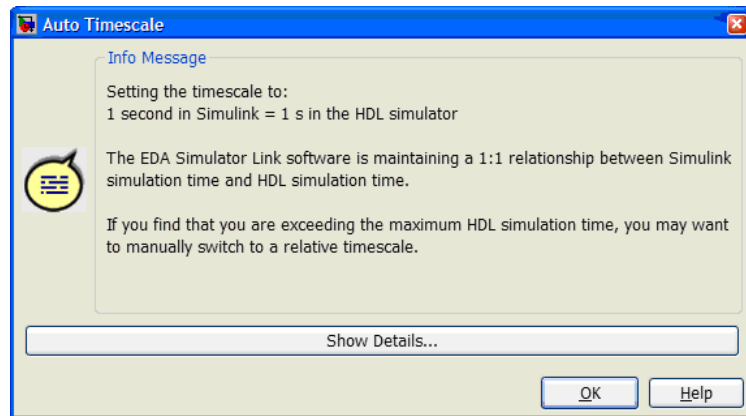


EDA Simulator Link software analyzes all the clock and port signal rates from the HDL Cosimulation block when it calculates the scale factor.

Note EDA Simulator Link cannot automatically calculate a sample timescale based on any signals driven via Tcl commands or in the HDL simulator. The link software cannot perform such calculations because it cannot know the rates of these signals.

The link software returns the sample rate in either seconds or ticks. If the results are in seconds, then the link software was able to resolve the timing differences in favor of fidelity (absolute time). If the results are in ticks, then the link software was best able to resolve the timing differences in favor of efficiency (relative time).

Each time you press Auto Timescale, the EDA Simulator Link software opens an informational GUI display that explains the results of Auto Timescale. If the link software cannot calculate a timescale for the given sample times, use the information in this dialog box to adjust your sample times.



Click **Show Details...** for information specific to your model's signals. Click **OK** to exit the informational dialog box.

3 Click **Apply** to commit your changes.

HDL Cosimulation

Note EDA Simulator Link does not support Auto Timescale calculated from frame-based signals.

For more on the timing relationship between the HDL simulator and Simulink, see “Understanding the Representation of Simulation Time”.

Manually Specifying a Relative Timing Relationship

To manually configure relative timing mode for a cosimulation, perform the following steps:

- 1 Select the **Timescales** tab of the HDL Cosimulation block parameters dialog box.
- 2 Verify that **Tick**, the default setting, is selected. If it is not, then select it from the list on the right.
- 3 Enter a scale factor in the text box on the left. The default scale factor is 1. For example, the next figure, shows the **Timescales** pane configured for a relative timing correspondence of 10 HDL simulator ticks to 1 Simulink second.



1 second in Simulink corresponds to Tick in the HDL simulator

- 4 Click **Apply** to commit your changes.

Manually Specifying an Absolute Timing Relationship

To manually configure absolute timing mode for a cosimulation, perform the following steps:

- 1 Select the **Timescales** tab of the HDL Cosimulation block parameters dialog box.
- 2 Select a unit of absolute time from the list on the right. The units available include **fs** (femtoseconds), **ps** (picoseconds), **ns** (nanoseconds), **us** (microseconds), **ms** (milliseconds), and **s** (seconds).

- 3 Enter a scale factor in the text box on the left. The default scale factor is 1. For example, in the next figure, the **Timescales** pane is configured for an absolute timing correspondence of 1 HDL simulator second to 1 Simulink second.



1 second in Simulink corresponds to s in the HDL simulator

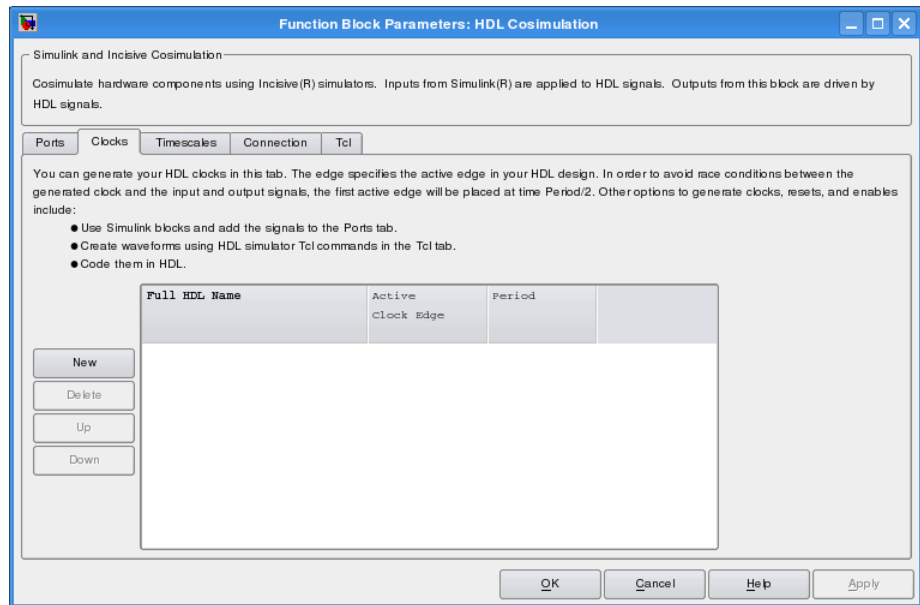
- 4 Click **Apply** to commit your changes.

Clocks Pane

Discovery Users The Clocks pane is not available on the HDL Cosimulation block for use with Synopsys Discovery. See `launchDiscovery` for instructions on adding clocks to your cosimulation model.

You can create optional rising-edge and falling-edge clocks that apply stimuli to your cosimulation model. To do so, use the Clocks pane of the HDL Cosimulation block (example shown for use with Incisive).

HDL Cosimulation



The scrolling list at the center of the pane displays HDL clocks that drive values to the HDL signals that you are modeling, using the deposit method.

Maintain the list of clock signals with the buttons on the left of the pane:

- **New** — Add a new clock signal to the list and select it for editing.
- **Delete** — Remove a clock signal from the list.
- **Up** — Move the selected clock signal up one position in the list.
- **Down** — Move the selected clock signal down one position in the list.

To commit edits to the Simulink model, you must also click **Apply**.

A clock signal has the following properties.

Full HDL Name

Specify each clock as a signal path name, using the HDL simulator path name syntax. For example: `/manchester/clock` or `manchester.clock`.

For information about and requirements for path specifications in Simulink, see “Specifying HDL Signal/Port and Module Paths for Cosimulation”.

Note You can copy signal path names directly from the HDL simulator **wave** window and paste them into the **Full HDL Name** field, using the standard copy and paste commands in the HDL simulator and Simulink. You must use the Path.Name view and not Db::Path.Name view. After pasting a signal path name into the **Full HDL Name** field, you must click the **Apply** button to complete the paste operation and update the signal list.

Edge

Select **Rising** or **Falling** to specify either a rising-edge clock or a falling-edge clock.

Period

You must either specify the clock period explicitly or accept the default period of 2.

If you specify an explicit clock period, you must enter a sample time equal to or greater than 2 resolution units (ticks).

If the clock period (whether explicitly specified or defaulted) is not an even integer, Simulink cannot create a 50% duty cycle. Instead, the EDA Simulator Link software creates the falling edge at

$$\text{clockperiod} / 2$$

(rounded down to the nearest integer).

HDL Cosimulation

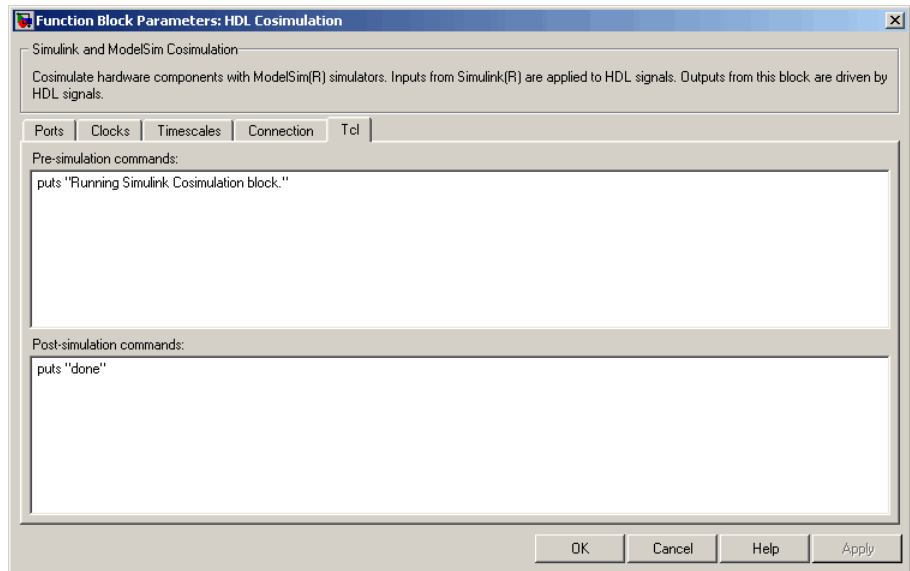
Note The **Clocks** pane does not support vectored signals. Signals must be logic types with 1 and 0 values.

For instructions on adding and editing clock signals, see “Creating Optional Clocks with the Clocks Pane of the HDL Cosimulation Block”.

Tcl Pane

Discovery Users The Tcl pane is not available on the HDL Cosimulation block for use with Synopsys Discovery. See `launchDiscovery` for instructions on issuing Tcl commands during a cosimulation session.

Specify tools command language (Tcl) commands to be executed before and after the HDL simulator simulates the HDL component of your Simulink model (example shown for use with ModelSim).



Pre-simulation commands

Contains Tcl commands to be executed before the HDL simulator simulates the HDL component of your Simulink model. You can specify one Tcl command per line in the text box or enter multiple commands per line by appending each command with a semicolon (;), the standard Tcl concatenation operator.

Use of this field can range from something as simple as a one-line echo command to confirm that a simulation is running to a complex script that performs an extensive simulation initialization and startup sequence.

Post-simulation commands

Contains Tcl commands to be executed after the HDL simulator simulates the HDL component of your Simulink model. You can specify one Tcl command per line in the text box or enter multiple commands per line by appending each command with a semicolon (;), the standard Tcl concatenation operator.

HDL Cosimulation

Creating a Tcl Script as an Alternative to Using the Tcl Pane

You can create a Tcl script that lists the Tcl commands you want to execute on the HDL simulator, either pre- or post-simulation.

Tcl Scripts for ModelSim Users

You can create a ModelSim DO file that lists Tcl commands and then specify that file with the ModelSim `do` command as follows:

```
do mycosimstartup.do
```

Or

```
do mycosimcleanup.do
```

You can include the `quit -f` command in an after-simulation Tcl command string or DO file to force ModelSim to shut down at the end of a cosimulation session. To ensure that all other after-simulation Tcl commands specified for the model will execute, specify all after simulation Tcl commands in a single cosimulation block and place `quit` at the end of the command string or DO file.

With the exception of `quit`, the command string or DO file that you specify cannot include commands that load a ModelSim project or modify simulator state. For example, they cannot include commands such as `start`, `stop`, or `restart`.

Tcl Scripts for Incisive Users

You can create an HDL simulator Tcl script that lists Tcl commands and then specify that file with the HDL simulator `source` command as follows:

```
source mycosimstartup.script_extension
```

Or

```
source mycosimcleanup.script_extension
```

You can include the `exit` command in an after-simulation Tcl script to force the HDL simulator to shut down at the end of a cosimulation session. To ensure that all other after-simulation Tcl commands specified for the model will execute, specify all after simulation Tcl commands in a single cosimulation block and place `exit` at the end of the command string or Tcl script.

With the exception of the `exit` command, the command string or Tcl script that you specify cannot include commands that load an HDL simulator project or modify simulator state. For example, neither can include commands such as `run`, `stop`, or `reset`.

The following example shows a Tcl script when the `-gui` argument was used with `hdlsimmatlab` or `hdlsimulink`:

```
after 1000 {ncsim -submit exit}
```

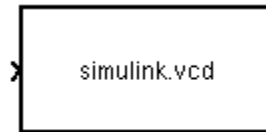
This next example is of a Tcl exit script to use when the `-tcl` argument was used with `hdlsimmatlab` or `hdlsimulink`:

```
after 1000 {exit}
```

To VCD File

Purpose Generate value change dump (VCD) file

Library EDA Simulator Link



Description To VCD File

The To VCD File block generates a VCD file that contains information about changes to signals connected to the block's input ports and names the file with the specified file name. You can use VCD files during design verification in the following ways:

- For comparing results of multiple simulation runs, using the same or different simulator environments
- As input to post-simulation analysis tools
- For porting areas of an existing design to a new design

Using the Block Parameters dialog box, you can specify the following parameters:

- The file name to be used for the generated file
- The number of block input ports that are to receive signal data
- The timescale to relate Simulink sample times with HDL simulator ticks

VCD files can grow very large for larger designs or smaller designs with longer simulation runs. However, the only limitation on the size of a VCD file generated by the To VCD File block is the maximum number of signals (and symbols) supported, which is 94^3 (830,584).

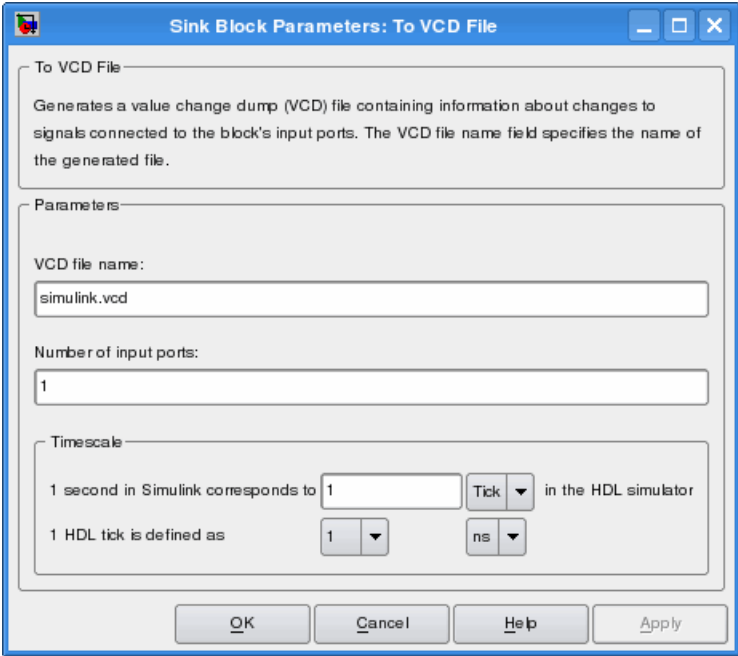
For a description of the VCD file format, see “VCD File Format” on page 2-29.

Note The To VCD File block is integrated into the Simulink Signal & Scope Manager. See the *Simulink User's Guide* for more information on using the Signal & Scope Manager.

Graphically Displaying VCD File Data

You can graphically display VCD file data or analyze the data with postprocessing tools. For example, the ModelSim `vcd2wlf` tool converts a VCD file to a WLF file that you can view in a ModelSim **wave** window. Other examples of postprocessing include the extraction of data pertaining to a particular section of a design hierarchy or data generated during a specific time interval.

Dialog Box



The dialog box, titled "Sink Block Parameters: To VCD File", contains the following sections and controls:

- To VCD File:** A text area containing the description: "Generates a value change dump (VCD) file containing information about changes to signals connected to the block's input ports. The VCD file name field specifies the name of the generated file."
- Parameters:**
 - VCD file name:** A text input field containing "simulink.vcd".
 - Number of input ports:** A text input field containing "1".
- Timescale:**
 - 1 second in Simulink corresponds to:** A text input field containing "1" followed by a "Tick" dropdown menu.
 - 1 HDL tick is defined as:** A text input field containing "1" followed by a dropdown menu set to "ns".

At the bottom of the dialog are four buttons: "OK", "Cancel", "Help", and "Apply".

VCD file name

The file name to be used for the generated VCD file. If you specify a file name only, Simulink places the file in your current MATLAB folder. Specify a complete path name to place the generated file in a different location. If you specify the same name for multiple To VCD File blocks, Simulink automatically adds a numeric postfix to identify each instance uniquely.

Note If you want the generated file to have a .vcd file type extension, you must specify it explicitly.

Do not give the same file name to different VCD blocks. Doing so results in invalid VCD files.

Number of input ports

The number of block input ports on which signal data is to be collected. The block can handle up to 94^3 (830,584) signals, each of which maps to a unique symbol in the VCD file.

In some cases, a single input port maps to multiple signals (and symbols). This multiple mapping occurs when the input port receives a multidimensional signal.

Because the VCD specification does not include multidimensional signals, Simulink flattens them to a 1D vector in the file.

Timescale

Choose an optimal timing relationship between Simulink and the HDL simulator.

The timescale options specify a correspondence between one second of Simulink time and some quantity of HDL simulator time. You can express this quantity of HDL simulator time in one of the following ways:

- In *relative* terms (i.e., as some number of HDL simulator ticks). In this case, the cosimulation operates in *relative timing mode*, which is the timing mode default.

To use relative mode, select **Tick** from the pop-up list at the label **in the HDL simulator**, and enter the desired number of ticks in the edit box at **1 second in Simulink corresponds to**. The default value is 1 Tick.

- In *absolute* units (such as milliseconds or nanoseconds). In this case, the cosimulation operates in *absolute timing mode*.

To use absolute mode, select the desired resolution unit from the pop-up list at the label **in the HDL simulator** (available units are fs, ps, ns, us, ms, s), and enter the desired number of resolution units in the edit box at **1 second in Simulink corresponds to**. Then, set the value of the HDL simulator tick by selecting 1, 10, or 100 from the pop-up list at **1 HDL Tick is defined as** and the resolution unit from the pop-up list at **defined as**.

VCD File Format

The format of generated VCD files adheres to IEEE Std 1364-2001. The following table describes the format.

Generated VCD File Format

File Content	Description
<pre>\$date 23-Sep-2003 14:38:11 \$end</pre>	Data and time the file was generated.
<pre>\$version EDA Simulator Link version 1.0 \$ end</pre>	Version of the VCD block that generated the file.

Generated VCD File Format (Continued)

File Content	Description
<pre>\$timescale 1 ns \$ end</pre>	<p>The time scale that was used during the simulation.</p>
<pre>\$scope module manchestermodel \$end</pre>	<p>The scope of the module being dumped.</p>
<pre>\$var wire 1 ! Original Data [0] \$end \$var wire 1 " Recovered Clock [0] \$end \$var wire 1 # Recovered Data [0] \$end \$var wire 1 \$ Data Validity [0] \$end</pre>	<p>Variable definitions. Each definition associates a signal with character identification code (symbol).</p> <p>The symbols are derived from printable characters in the ASCII character set from ! to ~.</p> <p>Variable definitions also include the variable type (wire) and size in bits.</p>
<pre>\$upscope \$end</pre>	<p>Marks a change to the next higher level in the HDL design hierarchy.</p>
<pre>\$enddefinitions \$end</pre>	<p>Marks the end of the header and definitions section.</p>
<pre>#0</pre>	<p>Simulation start time.</p>

Generated VCD File Format (Continued)

File Content	Description
<pre>\$dumpvars 0! 0" 0# 0\$ \$end</pre>	<p>Lists the values of all defined variables at time equals 0.</p>
<pre>#630 1!</pre>	<p>The starting point of logged value changes from checks of variable values made at each simulation time increment.</p> <p>This entry indicates that at 63 nanoseconds, the value of signal <code>Original Data</code> changed from 0 to 1.</p>
<pre>. . . #1160 1# 1\$</pre>	<p>At 116 nanoseconds the values of signals <code>Recovered Data</code> and <code>Data Validity</code> changed from 0 to 1.</p>
<pre>\$dumpoff x! x" x# x\$</pre>	<p>Marks the end of the file by dumping the values of all variables as the value x.</p>

To VCD File

Generated VCD File Format (Continued)

File Content	Description
\$end	

Function Reference

HDL Cosimulation (p. 3-2)	Describes the EDA Simulator Link MATLAB functions available for use with HDL cosimulation
FPGA Implementations (p. 3-4)	Describes the EDA Simulator Link MATLAB functions available for use with generating FPGA implementations
Virtual Platform Simulation (p. 3-5)	Describes the EDA Simulator Link MATLAB functions available for use with generating Virtual Platform simulations

HDL Cosimulation

<code>breakHdlSim</code>	Execute <code>stop</code> command in HDL simulator from MATLAB
<code>configuremodelsim</code>	Configure ModelSim for use with EDA Simulator Link
<code>dec2mvl</code>	Convert decimal integer to binary string
<code>hdldaemon</code>	Control MATLAB server that supports interactions with HDL simulator
<code>hdlsimmatlab</code>	Load instantiated HDL design for verification with Cadence Incisive and MATLAB
<code>hdlsimulink</code>	Load instantiated HDL design for cosimulation with Cadence Incisive and Simulink
<code>launchDiscovery</code>	Launch Synopsys Discovery tools for use with Simulink and MATLAB using EDA Simulator Link software
<code>matlabcp</code>	Associate MATLAB component function with instantiated HDL design
<code>matlabtb</code>	Schedule MATLAB test bench session for instantiated HDL module
<code>matlabtbeval</code>	Call specified MATLAB function once and immediately on behalf of instantiated HDL module
<code>mv12dec</code>	Convert multivalued logic to decimal
<code>nclaunch</code>	Start and configure Cadence Incisive simulators for use with EDA Simulator Link software
<code>nomatlabtb</code>	End active MATLAB test bench and MATLAB component sessions

<code>notifyMatlabServer</code>	Send HDL simulator event and process IDs to MATLAB server
<code>pingHdlSim</code>	Block cosimulation until HDL simulator is ready for simulation
<code>tclHdlSim</code>	Execute Tcl command in Incisive or ModelSim simulator
<code>vsim</code>	Start and configure ModelSim for use with EDA Simulator Link
<code>vsimmatlab</code>	Load instantiated HDL module for verification with ModelSim and MATLAB
<code>vsimulink</code>	Load instantiated HDL module for cosimulation with ModelSim and Simulink
<code>waitForHdlClient</code>	Wait until specified event ID is obtained or time-out occurs

FPGA Implementations

`fpgamodelsetup`

Set Simulink model parameters for
FPGA workflow

`makefpgaproject`

Generate Xilinx® ISE project and
FPGA hardware-in-the-loop

`setupxilinx tools`

Configure MATLAB environment for
use with Xilinx FPGA workflow

Virtual Platform Simulation

Currently, there are no EDA Simulator Link MATLAB functions available for use with generating Virtual Platform simulations.

Functions — Alphabetical List

breakHdlSim

Purpose Execute stop command in HDL simulator from MATLAB

Syntax

```
breakHdlSim()  
breakHdlSim('portNumber')  
breakHdlSim('portNumber','hostName')
```

Description `breakHdlSim()` executes a stop command on the HDL simulator on the local host. Use this function to unblock the HDL simulator after the HDL simulator has loaded the simulation but before Simulink starts the simulation. If, after starting the simulation, you decide to add more signals to the waveform window, use this function to unblock the HDL simulator first. When you use `breakHdlSim`, make sure that you specify the proper connection information to the HDL simulator.

`breakHdlSim('portNumber')` executes a stop command on the HDL simulator on port *portNumber*.

`breakHdlSim('portNumber','hostName')` executes a stop command on the HDL simulator on host *hostName*.

Examples Stop the HDL simulator that is currently running on the local host.

```
>> breakHdlSim()
```

Stop the HDL simulator that is currently running on port 1234.

```
>> breakHdlSim('1234')
```

Stop the HDL simulator that is currently running on port 1234 and host "mylinux".

```
>> breakHdlSim('1234','mylinux')
```

See Also `pingHdlSim`

Purpose

Configure ModelSim for use with EDA Simulator Link

Syntax

```
configuremodelsim  
configuremodelsim('PropertyName', 'PropertyValue'...)
```

Description

Note `configuremodelsim` has been replaced by the guided setup script (`syscheckmq`) for configuring your simulator setup. Although `configuremodelsim` is supported for backward compatibility, you should consider using the setup script instead. See “Diagnosing and Customizing Your Setup for Use with the HDL Simulator and EDA Simulator Link Software”.

`configuremodelsim` configures ModelSim for use with the MATLAB and Simulink features of EDA Simulator Link. There are two uses for this function:

- To configure ModelSim so that it may access EDA Simulator Link when invoked from outside of MATLAB
- To add Tcl commands to the Tcl startup script that runs every time you invoke ModelSim

When you use `configuremodelsim` without any arguments, the function prompts you to either allow it to find the installed ModelSim executable or have you provide the path to the ModelSim installation you want to use. If you had not configured the software previously (no Tcl DO file exists), `configuremodelsim` creates a new `ModelSimTclFunctionsForMATLAB.tcl` script in the `tcl` folder under the ModelSim installation. If a previous configuration exists, `configuremodelsim` prompts you to decide if you want to replace the existing configuration.

`configuremodelsim('PropertyName', 'PropertyValue'...)` starts an interactive or programmatic script (depending on which property name/value pairs you select) that allows you to customize the ModelSim

configuration. See `configuremodelsim` Property Name/Property Value pairs.

After you call this function, ModelSim is ready to use EDA Simulator Link when you invoke ModelSim from outside of MATLAB. You can use the EDA Simulator Link functions for use in the HDL simulator to perform the following actions:

- Load instances of VHDL entities or Verilog modules for simulations that use MATLAB or Simulink for verification or cosimulation
- Begin MATLAB test bench or component sessions for loaded instances
- End MATLAB test bench or component sessions

If you have specified Tcl commands to add to the Tcl startup DO file, those commands are now added to the `ModelSimTclFunctionsForMATLAB.tcl` script.

Note that `configuremodelsim` can only configure one platform. The process hard-codes the path to the HDL server/client libraries, which are specific to the OS (and GCC).

Usage Considerations

- `configuremodelsim` is intended to be used for setting up ModelSim and MATLAB when you plan to start ModelSim from outside of MATLAB.

If you intend to invoke `vsim` from the MATLAB prompt, then you do not need to use `configuremodelsim`. (MATLAB will find `vsim` if it already appears in the system path, and, if it does not, you can set the `vsimdir` property value of `vsim` in MATLAB to provide the path information.)

In addition, if you are starting ModelSim from outside of MATLAB, you should define your environment with the path to the ModelSim executable before running `configuremodelsim`.

Note The property name/property value options for `vsim` may have been set previously with a call to `configuremodelsim`. To check on current settings, search for and browse through the contents of the file `\tcl\ModelSimTclFunctionsForMATLAB.tcl` in your ModelSim installation path. The `vsim` function overrides any options previously defined by the `configuremodelsim` function.

To start ModelSim from MATLAB with a default configuration previously defined by `configuremodelsim`, issue the command `!vsim` at the MATLAB command prompt.

- The `vsimdir` property value of `configuremodelsim` only instructs `configuremodelsim` where to put the Tcl DO file. It does not set up MATLAB workspace for MATLAB invocation of ModelSim (instead, you can perform this setup with the `vsimdir` property value of `vsim`).
- If you are using `configuremodelsim` to add Tcl commands to the Tcl startup DO file, to change the location of the Tcl startup DO file, or to remove the Tcl startup DO file, you can run `configuremodelsim` as many times as you wish. You need to run `configuremodelsim` only once to set the location of the Tcl DO file.

**Property
Name/Property
Value
Pairs**

`'action', 'install'`

Instructs `configuremodelsim` to create a new `ModelSimTclFunctionsForMATLAB.tcl` script.

This script is programmatic if you use `'vsimdir'` to specify the ModelSim installation you want to use; otherwise, `configuremodelsim` prompts you for the desired folder.

If a previous configuration exists, `configuremodelsim` prompts you to decide if you want to replace the existing configuration. If you respond yes, the old Tcl DO file is overwritten with a new one.

configuremodelsim

'action', 'uninstall'

Removes the EDA Simulator Link configuration from the ModelSim startup DO file. The property replaces the contents of ModelSimTclFunctionsForMATLAB.tcl with this single line of text: “# MATLAB and Simulink option was deconfigured.”

This script is programmatic if you use 'vsimdir' to specify the ModelSim installation you want to use; otherwise, configuremodelsim prompts you for the desired folder.

'tclstart', 'tcl_commands'

Adds one or more Tcl commands to the Tcl DO file that executes during ModelSim startup. Specify a command string or a cell array of command strings that configuremodelsim will append to ModelSimTclFunctionsForMATLAB.tcl.

This script is programmatic only; if you do not also use 'vsimdir' with this property, configuremodelsim uses the first vsim it encounters on the system path and modifies the Tcl DO file (ModelSimTclFunctionsForMATLAB.tcl) in the \tcl folder under this ModelSim installation.

'vsimdir', 'pathname'

Specifies where to put the Tcl script containing EDA Simulator Link Tcl commands. This script is programmatic only; if you do not specify a folder with this property, configuremodelsim uses the first vsim it encounters on the system path and installs the Tcl DO file (ModelSimTclFunctionsForMATLAB.tcl) in the \tcl folder under this ModelSim installation.

Examples

The following function call starts the interactive installation script that installs EDA Simulator Link commands for use with ModelSim:

```
configuremodelsim
```

Because the property name vsimdir was not supplied, configuremodelsim prompts you for the folder:

```
Identify the ModelSim installation to be configured for MATLAB and Simulink

Do you want configuremodelsim to locate installed ModelSim executables [y]/n? n

Please enter the path to your ModelSim executable file (modelsim.exe or vsim.exe):
D:\Applications\Modeltech_6.0e\win32

Modelsim successfully configured to be used with MATLAB and Simulink
```

When `configuremodelsim` is run on an existing configuration, the dialog looks similar to the following sample:

```
Identify the ModelSim installation to be configured for MATLAB and Simulink

Do you want configuremodelsim to locate installed ModelSim executables [y]/n? n

Please enter the path to your ModelSim executable file (modelsim.exe or vsim.exe):
D:\Applications\Modeltech_6.0e\win32

Previous MATLAB startup file found in this installation of ModelSim:
D:\Applications\Modeltech_6.0e\win32\..\tcl\ModelSimTclFunctionsForMATLAB.tcl
Do you want to replace this file [y]/n? y
Modelsim successfully configured to be used with MATLAB and Simulink
```

If you answer no to the prompt for replacing the file, you receive this message instead:

```
Modelsim configuration not updated for MATLAB and Simulink
```

This next example shows adding a Tcl command to the ModelSim configuration, for a customized Tcl DO file:

```
configuremodelsim('tclstart','echo Starting ModelSim and EDA Simulator Link')

vsimoptions =

echo Starting ModelSim and EDA Simulator Link
```

configuremodelsim

Modelsim successfully configured to be used with MATLAB and Simulink

If you now inspect ModelSimTclFunctionsForMATLAB.tcl you will find this last Tcl command appended to the file.

The following example shows removing the EDA Simulator Link configuration from ModelSim:

```
configuremodelsim ('action', 'uninstall')

Identify the Modelsim installation to be deconfigured for MATLAB and Simulink

Do you want configuremodelsim to locate installed Modelsim executables [y]/n? n

Please enter the path to your Modelsim executable file (modelsim.exe or vsim.exe):
D:\Applications\Modeltech_6.0e\win32

Previous MATLAB startup file found in this installation of Modelsim:
D:\Applications\Modeltech_6.0e\win32...\tcl\ModelSimTclFunctionsForMATLAB.tcl
Do you want to replace this file (required for deconfiguration) [y]/n? y
Modelsim successfully deconfigured
```

If you now inspect ModelSimTclFunctionsForMATLAB.tcl you will find that the contents of the file have been removed.

Purpose Convert decimal integer to binary string

Syntax `dec2mvl(d)`
`dec2mvl(d,n)`

Description `dec2mvl(d)` returns the binary representation of `d` as a multivalued logic string. `d` must be an integer smaller than 2^{52} .
`dec2mvl(d,n)` produces a binary representation with at least `n` bits.

Examples The following function call returns the string '10111':

```
dec2mvl(23)
```

The following function call returns the string '01001':

```
dec2mvl(-23)
```

The following function call returns the string '11101001':

```
dec2mvl(-23,8)
```

See Also `mv12dec`

fpgamodelsetup

Purpose Set Simulink model parameters for FPGA workflow

Syntax `fpgamodelsetup (Model)`

Description `fpgamodelsetup (Model)` changes the parameters of the Simulink model specified by the *model* argument to values that are commonly used for HDL code generation and the Xilinx FPGA workflow. It also causes the Simulink® HDL Coder™ pane and the EDA Simulator Link pane to be visible in the Configuration Parameters dialog box.

The `fpgamodelsetup` command uses the Simulink `set_param` function to set up models for HDL code generation and FPGA workflow quickly and consistently. The model parameters settings provided by `fpgamodelsetup` are intended as useful defaults, but they may not be appropriate for all your applications.

Issue this function after you open or create a model but before you want to generate the FPGA project. You must set the GUI up first with this command or the EDA Link configuration panel will not be available to you.

Inputs *Model*

Name of the model whose generated code is to be used in creating the FPGA project, FPGA HIL, or in generating Tcl script

Examples

```
>>model = 'sfir_fixed';  
>>open_system(model);  
>>fpgamodelsetup(model);
```

See Also `makefpgaproject`

Purpose Control MATLAB server that supports interactions with HDL simulator

Syntax

```
hdldaemon
s=hdldaemon
hdldaemon('ParameterName',ParameterValue)
s=hdldaemon('ParameterName',ParameterValue)
hdldaemon('Option')
```

Description hdldaemon starts the HDL Link MATLAB server using shared memory interprocess communication. Only one hdldaemon per MATLAB session can be running at any given time.

s=hdldaemon starts the MATLAB server using shared memory and returns the server status connection in structure s.

hdldaemon('ParameterName',ParameterValue) starts the MATLAB server using shared memory and accepts optional inputs as one or more comma-separated parameter-value pairs. *ParameterName* is the name of the parameter inside single quotes. *ParameterValue* is the value corresponding to *ParameterName*. To start the server in socket mode, use the 'socket' parameter.

Note If server is already running, issuing hdldaemon with these arguments will shut down the current server and start the server up again using shared memory (unless socket is specified). The exception is issuing hdldaemon

s=hdldaemon('ParameterName',ParameterValue) works the same as hdldaemon('ParameterName',ParameterValue) and returns the server status connection in structure s.

hdldaemon('Option') accepts a single optional input. Only one option may be specified in a single call. You must establish the server connection before calling hdldaemon with one of these options.

Inputs

Option

Select one of the following options:

- 'kill'
Shuts down the MATLAB server without shutting down MATLAB.
- 'stop'
Shuts down the MATLAB server without shutting down MATLAB. There is no difference between using 'kill' and 'stop'.
- 'status'
Displays status of the MATLAB server. You can also use `s=hdldaemon('status')`, which displays MATLAB server status and returns status in structure `s`.

Parameter/Value Pairs

'time'

Specifies how the MATLAB server sends and returns time values.

- 'int64'
Specifies that the MATLAB server send and return time values in the MATLAB function callbacks as 64-bit integers representing the number of simulation steps.
See the `matlabcp/matlabtb` `tnow` parameter reference (“Defining EDA Simulator Link MATLAB Functions and Function Parameters”).
- 'sec'
Specifies that the MATLAB server sends and returns time values in the MATLAB function callbacks as `double` values that EDA Simulator Link scales to seconds based on the current HDL simulation resolution.

If server is already running, issuing `hdldaemon` with the `time` parameter alone will shut down the current server and start the server up again using shared memory.

Default: `'sec'`

`'quiet'`

Suppresses printing diagnostic messages. Errors still appear. Use this option to suppress the MATLAB server shutdown message when using `hdldaemon` to get an unused socket number.

- `'true'`

Suppress printing diagnostic messages.

- `'false'`

Do not suppress printing diagnostic messages.

If server is already running, issuing `hdldaemon` with the `quiet` parameter alone will shut down the current server and start the server up again using shared memory.

Default: `'false'`

`'socket'`

Defines the TCP/IP port used for communication. The socket value can be:

- 0, indicating the host automatically chooses a valid TCP/IP port
- An explicit port number ($1024 < \text{port} < 49151$)
- A service name (that is, alias) from `/etc/services` file

If you specify the operating system option (0), use `hdldaemon('status')` to acquire the assigned socket port number.

See “Specifying TCP/IP Values” for more information about TCP/IP ports.

`'tclcmd'`

Transmits a Tcl command to all connected clients (ModelSim and Incisive users only).

You may specify any valid Tcl command string. The Tcl command string you specify cannot include commands that load an HDL simulator project or modify simulator state. For example, the string cannot include commands such as `start`, `stop`, or `restart` (for ModelSim) or `run`, `stop`, or `reset` (for Incisive).

Note You can issue this command only after the software establishes a server connection.

Caution

Do not call `hdldaemon(tclcmd, 'tclcmd')` from inside a `matlabtb` or `matlabcp` function. Doing so results in a race condition, and the simulator hangs.

Outputs

`s`

`s` is a structure with these fields:

- `comm`
Shared memory or sockets
- `connections`
Number of open connections
- `ipc_id`

File system name (for shared memory communication channel)
or TCP/IP port number (for socket)

Examples

Start the MATLAB server using shared memory communication and use an integer representation of time:

```
hdldaemon('time', 'int64')
```

Start MATLAB server and specify socket communication on port 4449:

```
hdldaemon('socket', 4449)
```

Start MATLAB server with socket communication and use a 64-bit representation of time:

```
hdldaemon('socket', 4449, 'time', 'int64')
```

Check hdldaemon server status:

```
hdldaemon('status')
```

Returns, for example,

```
HDLDaemon socket server is running on port 4449 with 1 connections
```

Or

```
HDLDaemon shared memory server is running with 0 connections
```

Or

```
HDLDaemon is NOT running
```

hdldaemon

Check connection information for communication mode, number of existing connections, and the interprocess communication identifier (`ipc_id`) the MATLAB server is using for a link:

```
x=hlddaemon('status')
```

For a socket connection, returns:

```
x =  
      comm: 'sockets'  
 connections: 0  
      ipc_id: '4449'
```

For shared memory, returns:

```
x =  
      comm: 'shared memory'  
 connections: 0  
      ipc_id: [1x45 char]
```

You can examine `ipc_id` by entering it at the MATLAB command prompt:

```
>>x.ipc_id
```

Shut down server without shutting down MATLAB:

```
hdldaemon('kill')
```

Issue simple Tel commands:

```
hdldaemon('tclcmd','puts "This is a test"')
```

Issue complex Tel commands:

See the demo for Implementing the Filter Component of an Oscillator in MATLAB for an extensive example of a compound Tcl command.

See Also

`launchDiscovery` | `nclaunch` | `vsim`

How To

- “Starting the HDL Simulator from MATLAB”
- “Run MATLAB Test Bench Simulation”
- “Stop Test Bench Simulation”
- “Run MATLAB Component Function Simulation”

hdlsimmatlab

Purpose Load instantiated HDL design for verification with Cadence Incisive and MATLAB

Syntax `hdlsimmatlab <instance> [<ncsim_args>]`

Description The `hdlsimmatlab` command loads the specified instance of an HDL design for verification and sets up the Cadence Incisive simulator so it can establish a communication link with MATLAB. The Cadence Incisive simulator opens a simulation workspace as it loads the HDL design.

This command may be run from the HDL simulator prompt or from a Tcl script shell (`tclsh`).

This command is issued in the HDL simulator.

Arguments

`<instance>`
Specifies the instance of an HDL design to load for verification.

`<ncsim_args>`
Specifies one or more `ncsim` command arguments. For details, see the description of `ncsim` in the Cadence Incisive simulator documentation.

Examples The following command loads the module instance `parse` from library `work` for verification and sets up the Cadence Incisive simulator so it can establish a communication link with MATLAB:

```
tclshell> hdlsimmatlab work.parse
```

Purpose Load instantiated HDL design for cosimulation with Cadence Incisive and Simulink

Syntax `hdlsimulink [<ncsim_args>] <instance>
[-socket <tcp_spec>]`

Description The `hdlsimulink` command loads the specified instance of an HDL design for cosimulation and sets up the Cadence Incisive simulator so it can establish a communication link with Simulink. The Cadence Incisive simulator opens a simulation workspace into which it loads the HDL design.

This command is issued in the HDL simulator.

Argument `<ncsim_args>`
Specifies one or more `ncsim` command arguments. At a minimum, either `-gui` or `-tcl` is required. If you specify `-gui`, the Simulink GUI launches when the HDL design is loaded. If you specify `-tcl`, a Tcl script shell launches instead. If you do not specify either of these arguments, the HDL simulator runs the simulation without Simulink. Other valid `ncsim` arguments may be specified in addition to `-gui` or `-tcl`. For more information on `-gui`, `-tcl`, or other `ncsim` arguments, see the description of `ncsim` in the Cadence Incisive simulator documentation.

`<instance>`
Specifies the instance of an HDL design to load for cosimulation.

`-socket <tcp_spec>`
Specifies TCP/IP socket communication for the link between the Cadence Incisive simulator and MATLAB. This setting overrides the setting specified with the MATLAB `nclaunch` function. The `<tcp_spec>` can consist of a TCP/IP socket port number or service name (alias). For example, you might specify port number 4449 or the service name `matlab-service`.

For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports”.

If you run the HDL simulator and MATLAB on the same computer, you have the option of using shared memory for communication. Shared memory is the default mode of communication and takes effect if you do not specify `-socket <tcp-spec>` on the command line.

Note The communication mode that you specify with the `hdlsimulink` command must match what you specify for the communication mode when you configure EDA Simulator Link blocks in your Simulink model. For more information on modes of communication, see “Communications for HDL Cosimulation”. For more information on establishing the Simulink end of the communication link, see “Configuring the Communication Link in the HDL Cosimulation Block”.

Examples

The following command loads the module instance `parse` from library `work` for cosimulation, sets up the Cadence Incisive simulator so it can establish a communication link with Simulink, and opens a Tcl script shell:

```
tclshell> hdlsimulink -gui work.parse
```

Purpose Launch Synopsys Discovery tools for use with Simulink and MATLAB using EDA Simulator Link software

Syntax

```
pv = launchDiscovery('PropertyName', 'PropertyValue'...)  
pv = launchDiscovery(PropertyValueStruct)
```

Description

pv = launchDiscovery('PropertyName', 'PropertyValue'...) generates HDL compile scripts and HDL simulator launch scripts and executes them. These scripts set up an appropriate GCC environment and load the correct EDA Simulator Link library into the ModelSim[®] simulator. The function returns a structure of properties and their values.

For custom scripting requirements, you can use launchDiscovery to generate template "sh" scripts that you can modify and invoke from MATLAB using a "system" command.

You must use a property name/property value pair with launchDiscovery('PropertyName', 'PropertyValue'...).

pv = launchDiscovery(PropertyValueStruct) both passes and returns a structure of properties and their values.

In batch run modes, the function returns only after the HDL simulator starts and the HDL simulation begins. In interactive run modes, the function returns without waiting for the user to start the HDL simulation.

Property Name/ Property Value Pairs

Required Properties

'LinkType' , 'apname'

Specifies either Simulink or MATLAB. A Simulink link session includes using the HDL Cosimulation block in a Simulink model for cosimulation with the HDL simulator. A MATLAB link session includes using matlabbt, matlabcp, and matlabbtbeval to employ MATLAB functions as callbacks for HDL simulator events.

'VerilogFiles', 'pathname'

Specifies the full or relative (to "RunDir") path to Verilog files. Specify as single string in double quotes or as cell array of filenames.

Only one of "VerilogFiles" and "VhdlFiles" is required; specify both for mixed language designs.

'VhdlFiles', 'pathname'

Specifies the full or relative (to "RunDir") path to VHDL files. Specify as single string in double quotes or as cell array of filenames.

Only one of "VerilogFiles" and "VhdlFiles" is required; specify both for mixed language designs.

'TopLevel', 'modulename'

Specifies the name of the top-level HDL module.

'AccFile', 'filename'

Specifies the name of the signal access file that gives cosimulated signals read/write/force access to the EDA Simulator Link application. See the Synopsys Discovery documentation (search for "PLI table") on how to create this file.

Common Optional Properties

'PreSimTcl', 'command'

Specifies Tcl commands to execute before starting the HDL simulation. Use this property for simple waveform generation statements for signals such as clocks, resets, and enables.

'PingTimeout', 'seconds'

For Simulink link sessions only. Specifies the number of seconds to wait for the HDL simulator to launch before reporting back an error. To avoid waiting for the simulator to start, use the value of '0'. This property defaults to '60' for 'Batch' and 'Batch with Xterm' run modes, and '0' for 'CLI' and 'GUI' run modes.

'RunDir', 'dirname'
Specifies the folder from which to execute the compilation and launch scripts. This property defaults to an automatically created temporary folder.

'RunMode', 'modetype'
Specifies how to start the HDL simulator. This property accepts the following valid values:

- 'Batch': Start the HDL simulator in the background with no window.
- 'Batch with Xterm': Start the HDL simulator in the background but show session in an Xterm.
- 'CLI': Start the HDL simulator in an interactive shell.
- 'GUI': Start the HDL simulator in the Synopsys DVE GUI.

This value defaults to 'GUI'.

'RunTime', 'runtime'
Amount of time to run the simulation for when running in 'Batch' or 'Batch with Xterm' run modes. You can specify a raw number (which uses the time resolution unit value) or a number with a time unit (one of {'s','ms','us','ns','ps','fs'}). The default amount of run time is: "

For MATLAB link sessions only.

'VlogAnFlags', 'flagnames'
Specifies 'vlogan' flags.

'VhdlAnFlags', 'flagnames'
Specifies 'vhdlan' flags.

'UumCompFlags', 'flagnames'
Sets UUM-compatible compilation flags to 'vcs'.

'UumRunFlags', 'flagnames'
Sets UUM-compatible runtime flags to 'simv'.

Advanced Optional Properties

'CosimBlockList', 'blocklist'

For Simulink links only. Specifies a cell array of HDL Cosimulation block instances that are bound to the HDL simulator about to be built and launched. This value defaults to all cosimulation blocks in the current model. Correct syntax is:

```
'CosimBlockList', { 'block1', block2', ... }
```

'HostComm', 'commtype'

For Simulink link sessions only. Specifies the communications mechanism between Simulink and a local HDL simulator. This property accepts the following valid values:

- 'AutoGenSocketPort': Find an available TCP port on the current host and program the CosimBlockList with that port. This is the default value for HostComm.
- 'SharedPipe': Program the CosimBlockList to use a shared pipe connection.
- 'GetFromCosimBlock': Use whichever communication parameters appear in any existing cosimulation block masks.
- '<portnumber>': Program the CosimBlockList with a numeric socket port value, '<portnumber>', specified as a string.
- '<servicename>': Program the CosimBlockList with an OS TCP/IP service name, '<servicename>', specified as a string.

Note launchDiscovery currently does not directly support remote host execution; see “Examples” on page 4-26 section for help in setting up remote connections.

'HostName', 'name'

Specifies a remote host name for cross-machine co-simulations with Simulink.

- 'SkipCompilation', true|false
When true, instructs the HDL simulator *not* to execute the compilation script. This value defaults to false.
- 'SkipLaunch', true|false
When true, instructs the HDL simulator *not* to execute the launch script. This value defaults to false.
- 'SkipScriptGeneration', true|false
When true, instructs HDL simulator *not* to write the compilation and launch scripts. This value defaults to false.
- 'UserEnv', 'arrayname'
Specifies a cell array of VAR=value environment variables for use by the compilation and launch scripts. Correct syntax is:

```
'UserEnv', { 'VAR1=val1', 'VAR2=val2', ... }
```

VG_GNU_PACKAGE Properties

The default GCC compiler used is the default VG_GNU_PACKAGE from a standard installation in the VCS tree. If you want to compile using a different version of GCC, you must specify the following properties.

- 'UseDefaultVgGnuPackage', true|false
Specifies using the default VG_GNU_PACKAGE in the VCS installation tree. See Synopsys documentation for the installation instructions. When you set the UseDefaultVgGnuPackage property to True, the function ignores VgGnuPackage and VgGnuGccVersion. To guarantee inter-operability of the link application with the ModelSim software, keep this property set to True. This value defaults to True.
- 'VgGnuPackage', 'dirpath'
Specifies the full directory path to a nondefault installation (an installation outside of the VCS tree) of VG_GNU_PACKAGE. This value defaults to 'none'.

`'VgGnuGccVersion', 'version'`

Specifies the version of GCC to use. Currently, only gcc-4.2.2 is supported.

The default value is `'gcc-4.2.2'`.

Typical use cases for these properties include:

- Use default GCC version in the default `VG_GNU_PACKAGE` installation location. You specify nothing. Synopsys and the `VG_GNU_PACKAGE` distribution determine these defaults.
- Use default GCC version in a nondefault `VG_GNU_PACKAGE` installation location. For this property, you must specify:
 - `'UseDefaultVgGnuPackage', false`
 - `'VgGnuPackage', '/path/to/vg/gnu/installation'`
- Use nondefault GCC version in the default `VG_GNU_PACKAGE` installation location. For this property, you must specify:
 - `'UseDefaultVgGnuPackage', false`
 - `'VgGnuGccVersion', 'gcc-4.4.2'` (for example)
- Use nondefault GCC version in a nondefault `VG_GNU_PACKAGE` installation location. For this property, you must specify:
 - `'UseDefaultVgGnuPackage', false`
 - `'VgGnuPackage', '/path/to/vg/gnu/installation'`
 - `'VgGnuGccVersion', 'gcc-4.4.2'` (for example)

Examples

This example compiles and launches a single-file HDL design for cosimulation with Simulink. The code allows the use of Verilog-2000 syntax in the HDL source. This code launches the Synopsys DVE software.

```
>> launchDiscovery( ...  
    'LinkType',      'Simulink', ...
```

```
        'VerilogFiles', 'myinverter.v', ...
        'VlogAnFlags', '+v2k', ...
        'TopLevel', 'myinverter', ...
        'AccFile', 'myinverter.acc' ...
    );
```

This next example compiles and launches an HDL design in batch mode. In batch mode, the HDL simulator exits after the simulation completes, thus the example relaunches the simulation by calling `launchDiscovery` again with the previously returned property/value structure.

To run cosimulation after HDL simulator has exited:

```
>> pv = launchDiscovery( ...
    'LinkType', 'Simulink', ...
    'VhdlFiles', "mymultiplier.vhd mymultiplier_tb.vhd", ...
    'TopLevel', 'mymultiplier_tb', ...
    'AccFile', 'mymultiplier.acc', ...
    'RunMode', 'Batch', ...
);
```

To rerun cosimulation:

```
>> pv.SkipScriptGeneration = true;
>> pv.SkipCompilation = true;
>> pv = launchDiscovery(pv);           % relaunch the simulator
```

This next example generates scripts for customizing the environment of a specific project (`USER_ENV` includes some custom environment). Some common reasons to customize the resultant script include:

- You want to run the scripts on a different platform.
- You want to run the scripts on the same platform but on a remote machine.
- The build and run for the HDL simulator is part of a larger environment involving Perl scripts, makefiles, or LSF.

- You want to run in 32-bit mode on a 64-bit machine

```
>> srcDir = '/path/to/src';
>> launchDiscovery( ...
    'LinkType',          'MATLAB', ...
    'VhdlFiles',        {[srcDir '/top.vhd'], [srcDir '/dut.vhd']}, ...
    'TopLevel',         'top', ...
    'AccFile',          'top.acc', ...
    'RunDir',           '/testruns/myrun', ...
    'UserEnv',          {'LM_LICENSE_FILE=/path/to/license.dat'}, ...
    'SkipCompilation',  true, ...
    'SkipLaunch',       true ...
);
```

On remote machine, for example, you might use:

```
sh> cd /testruns/myrun
sh> (edit scripts as needed)
sh> . tmwESLDS.compile.sh
sh> . tmwESLDS.launch.sh
```

After you finalize the scripts, you can execute them from MATLAB:

```
>> system('rsh linux100 cd /testruns/myrun ; sh tmwESLDS.compile.sh ;
sh tmwESLDS.launch.sh');
```

This example shows the generated compilation script:

```
# AUTO-GENERATED SH SCRIPT FOR Simulink COSIMULATION
#--- EDA Link Environment ---

LAUNCHER_NAME=tmwESLDS
NUM_BITS=64
LINK_LIB_DIR=/matlab/toolbox/edalink/extensions/discovery/linux64
LINK_SL_FILE=lib1fdhd1s_vlog_gcc336.so
LINK_ML_FILE=lib1fdhd1c_vlog_gcc336.so
BITS_FLAG=-full64
export VG_GNU_PACKAGE=${VCS_HOME}/gnu/linux
COMPILE_SETUP_CMDS=". ${VG_GNU_PACKAGE}/source_me_${NUM_BITS}.sh"
export LD_LIBRARY_PATH=${VG_GNU_PACKAGE}/gcc-${NUM_BITS}/slib64:${LINK_LIB_DIR}:${LD_LIBRARY_PATH}
LOAD_SL_LIB="-load ${LINK_SL_FILE}:simlinkserver"
LOAD_ML_LIB="-load ${LINK_ML_FILE}:matlabclient"
VHPI_SL_LIB="-vhpi ${LINK_SL_FILE}:simlinkserver"
VHPI_ML_LIB="-vhpi ${LINK_ML_FILE}:matlabclient"
export SL_LIB_SOCKET=37592
VLOG_FILES=/matlab/toolbox/edalink/extensions/discovery/discoverydemos/Filter/lowpass_filter.v
VHDL_FILES=
TOP_LEVEL=lowpass_filter
ACC_FILE=/matlab/toolbox/edalink/extensions/discovery/discoverydemos/Filter/lowpass_filter.pli_acc.tab
VHDLAN_FLAGS=
VLOGAN_FLAGS="+v2k"
UUMCOMP_FLAGS=
UUMRUN_FLAGS=
COMP_DEBUG_FLAGS=-debug_all
LAUNCH_DEBUG_FLAGS="-gui -i tmwESLDS.presim.tcl"

#--- User Environment ---

eval ${COMPILE_SETUP_CMDS}
vlogan ${BITS_FLAG} ${VLOGAN_FLAGS} ${VLOG_FILES}
vcs ${COMP_DEBUG_FLAGS} ${UUMCOMP_FLAGS} +vpi +applylearn+${ACC_FILE} ${BITS_FLAG} ${TOP_LEVEL} ${LOAD_ML_LIB}
```

This example shows the generated launch script:

```
# AUTO-GENERATED SH SCRIPT FOR Simulink COSIMULATION
#--- EDA Link Environment ---

LAUNCHER_NAME=tmwESLDS
NUM_BITS=64
LINK_LIB_DIR=/matlab/toolbox/edalink/extensions/discovery/linux64
LINK_SL_FILE=liblfdhd1s_vlog_gcc336.so
LINK_ML_FILE=liblfdhd1c_vlog_gcc336.so
BITS_FLAG=-full64
export VG_GNU_PACKAGE=${VCS_HOME}/gnu/linux
COMPILE_SETUP_CMDS=". ${VG_GNU_PACKAGE}/source_me_${NUM_BITS}.sh"
LAUNCH_SETUP_CMDS=". ${VG_GNU_PACKAGE}/source_me_${NUM_BITS}.sh"
export LD_LIBRARY_PATH=${VG_GNU_PACKAGE}/gcc-${NUM_BITS}/slib64:${LINK_LIB_DIR}:${LD_LIBRARY_PATH}
LOAD_SL_LIB="-load ${LINK_SL_FILE}:simlinkserver"
LOAD_ML_LIB="-load ${LINK_ML_FILE}:matlabclient"
VHPI_SL_LIB="-vhpi ${LINK_SL_FILE}:simlinkserver"
VHPI_ML_LIB="-vhpi ${LINK_ML_FILE}:matlabclient"
export SL_LIB_SOCKET=37592
VLOG_FILES=/matlab/toolbox/edalink/extensions/discovery/discoverydemos/Filter/lowpass_filter.v
VHDL_FILES=
TOP_LEVEL=lowpass_filter
ACC_FILE=/matlab/toolbox/edalink/extensions/discovery/discoverydemos/Filter/lowpass_filter.pli_acc.tab
VHDLAN_FLAGS=
VLOGAN_FLAGS="+v2k"
UUMCOMP_FLAGS=
UUMRUN_FLAGS=
COMP_DEBUG_FLAGS=-debug_all
LAUNCH_DEBUG_FLAGS="-gui -i tmwESLDS.presim.tcl"

#--- User Environment ---

eval ${LAUNCH_SETUP_CMDS}
simv ${LAUNCH_DEBUG_FLAGS} ${UUMRUN_FLAGS}
```


makefpgaproject

Purpose Generate Xilinx ISE project and FPGA hardware-in-the-loop

Syntax `makefpgaproject(model/subsystem)`
`makefpgaproject(model/subsystem, 'ParameterName',
ParameterValue)`

Description `makefpgaproject(model/subsystem)` generates a Xilinx ISE project workflow according to Simulink model parameter settings. *model* specifies the name of the Simulink model, and *subsystem* specifies the name of a subsystem at the top level of the Simulink model.

`makefpgaproject(model/subsystem, 'ParameterName', ParameterValue)` accepts one or more comma-separated parameter name/value pairs so that you may specify optional build settings such as whether or not to continue build on warnings and HDL Coder parameters. Specify *ParameterName* inside single quotes.

Inputs `model/subsystem`

Name and path of the top-level subsystem whose generated code is to be used in creating and updating the FPGA project, FPGA HIL, and in generating Tcl script

Parameter Name/Value Pairs

ContinueOnWarning

When ContinueOnWarning is set to 'on', `makefpgaproject` continues to run when a warning is encountered, without pausing for user action. For example, instead of asking if you want to overwrite an existing ISE project, `makefpgaproject` overwrites the project without asking and displays a warning message.

- 'on'

Continue build when it encounters a warning without prompting user.

- 'off'

Prompt input from user when build encounters a warning.

Default: 'off'

HDLCoderParam

Specify HDL code generation options in a cell array of property-value pairs.

- (*HDLCoderParam*, {*HDLParamName1*, *HDLParamValue1*, *HDLParamName2*, *HDLParamValue2*, ...})

See Simulink HDL Coder documentation for a list of valid parameters and values for the `makehdl` command; you can use those same property-value pairs with `makefpga project`.

For example:

```
{'TargetLanguage', 'VHDL', ...  
 'TargetDirectory', 'myhdlsrc'}
```

You may have to turn on the option "Always generate HDL" in the **EDA Link** configuration parameters pane for the HDL code generation options to take effect.

Examples

Create FPGA project workflow with all defaults from the specified top level subsystem.

```
> makefpga project('model/subsystem')
```

Create FPGA project workflow from specified top level subsystem and do not prompt for action when warnings are encountered.

```
>makefpga project('model/subsystem', 'ContinueOnWarning', 'on')
```

makefpgaproject

Create FPGA project workflow from specified top level subsystem and use the specified HDL Coder property value pairs when generating HDL code.

```
>makefpgaproject('model/subsystem', 'HDLCoderParam', ...  
                {'TargetLanguage', 'VHDL', 'TargetDirectory', 'myhdlsrc'})
```

See Also

fpgamodelsetup

Purpose Associate MATLAB component function with instantiated HDL design

Syntax

```
matlabcp <instance>  
[<time-specs>]  
[-socket <tcp-spec>]  
[-rising <port>[,<port>...]]  
[-falling <port> [,<port>,...]]  
[-sensitivity <port>[,<port>,...]]  
[-mfunc <name>]  
[-use_instance_obj]  
[-argument]
```

Description The `matlabcp` command has the following characteristics:

- Starts the HDL simulator client component of the EDA Simulator Link software.
- Associates a specified instance of an HDL design created in the HDL simulator with a MATLAB function.
- Creates a process that schedules invocations of the specified MATLAB function.
- Cancels any pending events scheduled by a previous `matlabcp` command that specified the same instance. For example, if you issue the command `matlabcp` for instance `foo`, all previously scheduled events initiated by `matlabcp` on `foo` are canceled.

This command is issued in the HDL simulator.

MATLAB component functions simulate the behavior of modules in the HDL model. A stub module (providing port definitions only) in the HDL model passes its input signals to the MATLAB component function. The MATLAB component processes this data and returns the results to the outputs of the stub module. A MATLAB component typically provides some functionality (such as a filter) that is not yet implemented in the HDL code. See “Replacing an HDL Component with a MATLAB® Component Function”.

Notes The communication mode that you specify for `matlabcp` must match the communication mode you specified for `hdldaemon` when you established the server connection.

For socket communications, specify the port number you selected for `hdldaemon` when you issue a link request with the `matlabcp` command in the HDL simulator.

Arguments

<instance>

Specifies an instance of an HDL design that is associated with a MATLAB function. By default, `matlabcp` associates the instance to a MATLAB function that has the same name as the instance. For example, if the instance is `myfirfilter`, `matlabcp` associates the instance with the MATLAB function `myfirfilter` (note that hierarchy names are ignored; for example, if your instance name is `top.myfirfilter`, `matlabcp` would associate only `myfirfilter` with the MATLAB function). Alternatively, you can specify a different MATLAB function with `-mfunc`.

Note Do not specify an instance of an HDL module that has already been associated with a MATLAB function (via `matlabcp` or `matlabtb`). If you do, the new association overwrites the existing one.

<time-specs>

Specifies a combination of time specifications consisting of any or all of the following:

<timen>,...

Specifies one or more discrete time values at which the HDL simulator calls the specified MATLAB function. Each time value is relative to the current simulation time. Even if you do not specify a time, the HDL simulator calls the MATLAB function once at the start of the simulation. Separate multiple time values by a space. For example:

```
matlabtb vlogtestbench_top 10 ns, 10 ms, 10 sec
```

The MATLAB function executes when time equals 0 and then 10 nanoseconds, 10 milliseconds, and 10 seconds from time zero.

Note For time-based parameters, you can specify any standard time units (ns, us, and so on). If you do not specify units, the command treats the time value as a value of HDL simulation ticks.

-repeat <time>

Specifies that the HDL simulator calls the MATLAB function repeatedly based on the specified <timen>,... pattern. The time values are relative to the value of tnow at the time the HDL simulator first calls the MATLAB function.

-cancel <time>

Specifies a time at which the specified MATLAB function stops executing. The time value is relative to the value of tnow at the time the HDL simulator first calls the MATLAB function. If you do not specify a cancel time, the application calls the MATLAB function until you finish the simulation, quit the session, or issue a nomatlabtb call.

Note The -cancel option works only with the <time-specs> arguments. It does not affect any of the other scheduling arguments for matlabcp.

Note Place time specifications after the `matlabcp` instance and before any additional command arguments; otherwise the time specifications are ignored.

All time specifications for the `matlabcp` functions appear as a number and, optionally, a time unit:

- fs (femtoseconds)
- ps (picoseconds)
- ns (nanoseconds)
- us (microseconds)
- ms (milliseconds)
- sec (seconds)
- no units (tick)

`-socket <tcp_spec>`

Specifies TCP/IP socket communication for the link between the HDL simulator and MATLAB. When you provide TCP/IP information for `matlabcp`, you can choose a TCP/IP port number or TCP/IP port alias or service name for the `<tcp_spec>` parameter. If you are setting up communication between computers, you must also specify the name or Internet address of the remote host that is running the MATLAB server (`hdldaemon`). See “Specifying TCP/IP Values” for some valid `tcp_spec` examples.

For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports”.

If you run the HDL simulator and MATLAB on the same computer, you have the option of using shared memory for communication. Shared memory is the default mode of communication and takes effect if you do not specify `-socket <tcp_spec>` on the command line.

Note The communication mode that you specify with the `matlabcp` command must match what you specify for the communication mode when you issue the `hdldaemon` command in MATLAB.

For more information on modes of communication, see “Communications for HDL Cosimulation”. For more information on establishing the MATLAB end of the communication link, see “Starting the HDL Simulator from MATLAB”.

`-rising <signal>[, <signal>...]`

Indicates that the application calls the specified MATLAB function on the rising edge (transition from '0' to '1') of any of the specified signals. Specify `-rising` with the path names of one or more signals defined as a logic type (STD_LOGIC, BIT, X01, and so on).

For determining signal transition in:

- VHDL: Rising edge is {0 or L} to {1 or H}.
- Verilog: Rising edge is the transition from 0 to x, z, or 1, and from x or z to 1.

Note When specifying signals with the `-rising`, `-falling`, and `-sensitivity` options, specify them in full path name format. If you do not specify a full path name, the command applies the HDL simulator rules to resolve signal specifications.

`-falling <signal>[, <signal>...]`

Indicates that the application calls the specified MATLAB function whenever any of the specified signals experiences a falling edge—changes from '1' to '0'. Specify `-falling` with

the path names of one or more signals defined as a logic type (STD_LOGIC, BIT, X01, and so on).

For determining signal transition in:

- VHDL: Falling edge is {1 or H} to {0 or L}.
- Verilog: Falling edge is the transition from 1 to x, z, or 0, and from x or z to 0.

Note When specifying signals with the `-rising`, `-falling`, and `-sensitivity` options, specify them in full path name format. If you do not specify a full path name, the command applies the HDL simulator rules to resolve signal specifications.

`-sensitivity <signal>[, <signal>...]`

Indicates that the application calls the specified MATLAB function whenever any of the specified signals changes state. Specify `-sensitivity` with the path names of one or more signals. Signals of any type can appear in the sensitivity list and can be positioned at any level in the HDL model hierarchy.

Note When specifying signals with the `-rising`, `-falling`, and `-sensitivity` options, specify them in full path name format. If you do not specify a full path name, the command applies the HDL simulator rules to resolve signal specifications.

`-mfunc <name>`

The name of the MATLAB function that is associated with the HDL module instance you specify for `instance`. By default, the EDA Simulator Link software invokes a MATLAB function that has the same name as the specified HDL instance. Thus, if the names are the same, you can omit the `-mfunc` option. If the names are not the same, use this argument when you call `matlabcp`. If

you omit this argument and `matlabcp` does not find a MATLAB function with the same name, the command generates an error message.

`-use_instance_obj`

Instructs the function specified with the argument `-mfunc` to use an HDL instance object passed by EDA Simulator Link to the function. You include the `-use_instance_obj` argument with `matlabcp` in the following format:

```
matlabcp modelname -mfunc funcname -use_instance_obj
```

When you call `matlabcp` with the `use_instance_obj` argument, the function has the following signature:

```
function MyFunctionName(hdl_instance_obj)
```

The HDL instance object (`hdl_instance_obj`) has the fields shown in the following table.

Field	Read/Write Access	Description
<code>tnext</code>	Write only	Used to schedule a callback during the set time value. This field is equivalent to old <code>tnext</code> . For example: <pre>hdl_instance_obj.tnext = hdl_instance_obj.tnow + 5e-9</pre> will schedule a callback at time equals 5 nanoseconds from <code>tnow</code> .
<code>userdata</code>	Read/Write	Stores state variables of the current <code>matlabcp</code> instance. You can retrieve the variables the next time the callback of this instance is scheduled.

Field	Read/Write Access	Description
simstatus	Read only	<p>Stores the status of the HDL simulator. The EDA Simulator Link software sets this field to 'Init' during the first callback for this particular instance and to 'Running' thereafter. simstatus is a read-only property.</p> <pre>>> hdl_instance_obj.simstatus ans= Init</pre>
instance	Read only	<p>Stores the full path of the Verilog/VHDL instance associated with the callback. instance is a read-only property. The value of this field equals that of the module instance specified with the function call. For example:</p> <p>In the HDL simulator:</p> <pre>hdlsim> matlabcp osc_top -mfunc oscfilter use_instance_obj</pre> <p>In MATLAB:</p> <pre>>> hdl_instance_obj.instance ans= osc_top</pre>

Field	Read/Write Access	Description
argument	Read only	<p>Stores the argument set by the <code>-argument</code> option of <code>matlabcp</code>. For example:</p> <pre>matlabtb osc_top -mfunc oscfilter -use_instance_obj -argument foo</pre> <p>The link software supports the <code>-argument</code> option only when it is used with <code>-use_instance_obj</code>, otherwise the argument is ignored. <code>argument</code> is a read-only property.</p> <pre>>>hdl_instance_obj.argument ans= foo</pre>
portinfo	Read only	<p>Stores information about the VHDL and Verilog ports associated with this instance. <code>portinfo</code> is a read-only property, which has a field structure that describes the ports defined for the associated HDL module. For each port, the <code>portinfo</code> structure passes information such as the port's type, direction, and size. For more information on port data, see "Gaining Access to and Applying Port Information".</p> <pre>hdl_instance_obj.portinfo.field1.field2.field3</pre> <hr/> <p>Note When you use <code>use_instance_obj</code>, you access <code>tscale</code> through the HDL instance object. If you do not use <code>use_instance_obj</code>, you can still access <code>tscale</code> through <code>portinfo</code>.</p> <hr/>

Field	Read/Write Access	Description
tscale	Read only	<p>Stores the resolution limit (tick) in seconds of the HDL simulator. <code>tscale</code> is a read-only property.</p> <pre>>> hdl_instance_obj.tscale ans= 1.0000e-009</pre> <hr/> <p>Note When you use <code>use_instance_obj</code>, you access <code>tscale</code> through the HDL instance object. If you do not use <code>use_instance_obj</code>, you can still access <code>tscale</code> through <code>portinfo</code>.</p>
tnow	Read only	<p>Stores the current time. <code>tnow</code> is a read-only property.</p> <pre>hdl_instance_obj.tnext = hdl_instance_obj.tnow + fastestrate;</pre>
portvalues	Read/Write	<p>Stores the current values of and sets new values for the output and input ports for a <code>matlabcp</code> instance. For example:</p> <pre>>> hdl_instance_obj.portvalues ans = Read Only Input ports: clk_enable: [] clk: [] reset: [] Read/Write Output ports: sine_out: [22x1 char]</pre>

Field	Read/Write Access	Description
linkmode	Read only	Stores the status of the callback. The EDA Simulator Link software sets this field to 'testbench' if the callback is associated with <code>matlabtb</code> and 'component' if the callback is associated with <code>matlabcp</code> . <code>linkmode</code> is a read-only property. <pre>>> hdl_instance_obj.linkmode ans= component</pre>

-argument

Used to pass user-defined arguments from the `matlabcp` invocation on the HDL side to the MATLAB function callbacks. Supported with `-use_instance_obj` only. See the field listing under the `-use_instance_obj` property.

Examples

The following examples demonstrate some ways you might use the `matlabcp` function.

Using `matlabcp` with the `-mfunc` option to Associate an HDL Component with a MATLAB Function of a Different Name

This example explicitly associates the Verilog module `vlogtestbench_top.u_matlab_component` with the MATLAB function `vlogmatlabcp` using the `-mfunc` option. The `'-socket'` option specifies using socket communication on port 4449.

```
matlabcp vlogtestbench_top.u_matlab_component -mfunc vlogmatlabcp -socket 4449
```

Using `matlabcp` with Explicit Times and the `-cancel` Option

This example implicitly associates the Verilog module, `vtestbench_top`, with the MATLAB function `vlogtestbench_top`, and includes explicit times with the `-cancel` option.

```
matlabcp vlogtestbench_top 1e6 fs 3 2e3 ps -repeat 3 ns -cancel 7ns
```

Using matlabcp with Rising and Falling Edges

This example implicitly associates the Verilog module, `vlogtestbench_top`, with the MATLAB function `vlogtestbench_top`, and also uses rising and falling edges.

```
hdlsim> matlabcp vlogtestbench_top 1 2 3 4 5 6 7 -rising outclk3
        -falling u_matlab_component/inoutclk
```

Using matlabcp and the HDL Instance Object

In this example, the HDL simulator makes repeated calls to `matlabcp` to bind multiple HDL instances to the same MATLAB function. Each call contains `-argument` as a constructor parameter to differentiate behavior.

```
> matlabcp u1_filter1x -mfunc osc_filter -use_instance_obj -argument oversample=1
> matlabcp u1_filter8x -mfunc osc_filter -use_instance_obj -argument oversample=8
> matlabcp u2_filter8x -mfunc osc_filter -use_instance_obj -argument oversample=8
```

The MATLAB function callback, `osc_filter.m`, sets up user instance-based state using `obj.userdata`, queries port and simulation context using other `obj` fields, and uses the passed in `obj.argument` to differentiate behavior.

```
function osc_filter(obj)
    if (strcmp(obj.simstatus,'Init'))
        ud = struct('Nbits', 22, 'Norder', 31, 'clockperiod', 80e-9, 'phase', 1);
        eval(obj.argument);
        if (~exist('oversample','var'))
            error('HdlLinkDemo:UseInstanceObj:BadCtorArg', ...
                'Bad constructor arg to osc_filter callback. Expecting
                ''oversample=value''.');
        end
        ud.oversample      = oversample;
        ud.oversampleperiod = ud.clockperiod/ud.oversample;
        ud.InDelayLine     = zeros(1,ud.Norder+1);
```

```
centerfreq = 70/256;
passband   = [centerfreq-0.01, centerfreq+0.01];
b          = fir1((ud.Norder+1)*ud.oversample-1, passband./ud.oversample);
ud.Hresp   = ud.oversample .* b;

obj.userdata = ud;
end

...
```

matlabtb

Purpose Schedule MATLAB test bench session for instantiated HDL module

Syntax

```
matlabtb <instance>
[<time-specs>]
[-socket <tcp-spec>]
[-rising <port>[,<port>...]]
[-falling <port> [,<port>,...]]
[-sensitivity <port>[,<port>,...]]
[-mfunc <name>]
[-use_instance_obj]
[-argument]
```

Description The matlabtb command has the following characteristics:

- Starts the HDL simulator client component of the EDA Simulator Link software.
- Associates a specified instance of an HDL design created in the HDL simulator with a MATLAB function.
- Creates a process that schedules invocations of the specified MATLAB function.
- Cancels any pending events scheduled by a previous matlabtb command that specified the same instance. For example, if you issue the command matlabtb for instance foo, all previously scheduled events initiated by matlabtb on foo are canceled.

This command is issued in the HDL simulator.

MATLAB test bench functions mimic stimuli passed to entities in the HDL model. You force stimulus from MATLAB or HDL scheduled with matlabtb.

Notes The communication mode that you specify for `matlabtb` must match the communication mode you specified for `hdldaemon` when you established the server connection.

For socket communications, specify the port number you selected for `hdldaemon` when you issue a link request with the `matlabtb` command in the HDL simulator.

Arguments

<instance>

Specifies the instance of an HDL module that the EDA Simulator Link software associates with a MATLAB test bench function. By default, `matlabtb` associates the instance with a MATLAB function that has the same name as the instance. For example, if the instance is `myfirfilter`, `matlabtb` associates the instance with the MATLAB function `myfirfilter` (note that hierarchy names are ignored; for example, if your instance name is `top.myfirfilter`, `matlabtb` would associate only `myfirfilter` with the MATLAB function). Alternatively, you can specify a different MATLAB function with `-mfunc`.

Note Do not specify an instance of an HDL module that has already been associated with a MATLAB function (via `matlabcp` or `matlabtb`). If you do, the new association overwrites the existing one.

<time-specs>

Specifies a combination of time specifications consisting of any or all of the following:

<timen>,...

Specifies one or more discrete time values at which the HDL simulator calls the specified MATLAB function. Each time value is relative to the current simulation time. Even if you do not specify a time, the HDL simulator calls the MATLAB function once at the start of the simulation. Separate multiple time values by a space. For example:

```
matlabtb vlogtestbench_top 10 ns, 10 ms, 10 sec
```

The MATLAB function executes when time equals 0 and then 10 nanoseconds, 10 milliseconds, and 10 seconds from time zero.

Note For time-based parameters, you can specify any standard time units (ns, us, and so on). If you do not specify units, the command treats the time value as a value of HDL simulation ticks.

-repeat <time>

Specifies that the HDL simulator calls the MATLAB function repeatedly based on the specified <timen>, ... pattern. The time values are relative to the value of tnow at the time the HDL simulator first calls the MATLAB function. For example:

```
matlabtb vlogtestbench_top 5 ns -repeat 10 ns
```

The MATLAB function executes at time equals 0 ns, 5 ns, 15 ns, 25 ns, and so on.

-cancel <time>

Specifies a time at which the specified MATLAB function stops executing. The time value is relative to the value of tnow at the time the HDL simulator first calls the MATLAB function. If you do not specify a cancel time, the application calls the MATLAB function until you finish the simulation, quit the session, or issue a nomatlabtb call.

Note The `-cancel` option works only with the `<time-specs>` arguments. It does not affect any of the other scheduling arguments for `matlabtb`.

Note Place time specifications after the `matlabtb` instance and before any additional command arguments; otherwise the time specifications are ignored.

All time specifications for the `matlabtb` functions appear as a number and, optionally, a time unit:

- fs (femtoseconds)
- ps (picoseconds)
- ns (nanoseconds)
- us (microseconds)
- ms (milliseconds)
- sec (seconds)
- no units (tick)

`-socket <tcp_spec>`

Specifies TCP/IP socket communication for the link between the HDL simulator and MATLAB. When you provide TCP/IP information for `matlabtb`, you can choose a TCP/IP port number or TCP/IP port alias or service name for the `<tcp_spec>` parameter. If you are setting up communication between computers, you must also specify the name or Internet address of the remote host that is running the MATLAB server (`hdldaemon`). See “Specifying TCP/IP Values” for some valid `tcp_spec` examples.

For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports”.

If you run the HDL simulator and MATLAB on the same computer, you have the option of using shared memory for communication. Shared memory is the default mode of communication and takes effect if you do not specify `-socket <tcp_spec>` on the command line.

Note The communication mode that you specify with the `matlabtb` command must match what you specify for the communication mode when you issue the `hdldaemon` command in MATLAB. For more information on modes of communication, see “Communications for HDL Cosimulation”. For more information on establishing the MATLAB end of the communication link, see “Starting the HDL Simulator from MATLAB”.

`-rising <signal>[, <signal>...]`

Indicates that the application calls the specified MATLAB function on the rising edge (transition from '0' to '1') of any of the specified signals. Specify `-rising` with the path names of one or more signals defined as a logic type (STD_LOGIC, BIT, X01, and so on).

For determining signal transition in:

- VHDL: Rising edge is {0 or L} to {1 or H}.
- Verilog: Rising edge is the transition from 0 to x, z, or 1, and from x or z to 1.

Note When specifying signals with the `-rising`, `-falling`, and `-sensitivity` options, specify them in full path name format. If you do not specify a full path name, the command applies the HDL simulator rules to resolve signal specifications.

`-falling <signal>[, <signal>...]`

Indicates that the application calls the specified MATLAB function whenever any of the specified signals experiences a falling edge—changes from '1' to '0'. Specify `-falling` with the path names of one or more signals defined as a logic type (STD_LOGIC, BIT, X01, and so on).

For determining signal transition in:

- VHDL: Falling edge is {1 or H} to {0 or L}.
- Verilog: Falling edge is the transition from 1 to x, z, or 0, and from x or z to 0.

Note When specifying signals with the `-rising`, `-falling`, and `-sensitivity` options, specify them in full path name format. If you do not specify a full path name, the command applies the HDL simulator rules to resolve signal specifications.

`-sensitivity <signal>[, <signal>...]`

Indicates that the application calls the specified MATLAB function whenever any of the specified signals changes state. Specify `-sensitivity` with the path names of one or more signals. Signals of any type can appear in the sensitivity list and can be positioned at any level of the HDL design.

If you specify the option with no signals, the interface is sensitive to value changes for all signals.

Note Use of this option for INOUT ports can result in double calls.

For example:

```
-sensitivity /randnumgen/dout
```

The MATLAB function executes if the value of dout changes.

Note When specifying signals with the `-rising`, `-falling`, and `-sensitivity` options, specify them in full path name format. If you do not specify a full path name, the command applies the HDL simulator rules to resolve signal specifications.

`-mfunc <name>`

The name of the associated MATLAB function. If you omit this argument, `matlabtb` associates the HDL module instance to a MATLAB function that has the same name as the HDL instance. If you omit this argument and `matlabtb` does not find a MATLAB function with the same name, the command generates an error message.

`-use_instance_obj`

Instructs the function specified with the argument `-mfunc` to use an HDL instance object passed by EDA Simulator Link to the function. The `-use_instance_obj` argument is called with `matlabtb` in the following format:

```
matlabtb modelname -mfunc funcname -use_instance_obj
```

When you call `matlabcp` with the `use_instance_obj` argument, the function has the following signature:

```
function MyFunctionName(hdl_instance_obj)
```

The HDL instance object (`hdl_instance_obj`) has the fields shown in the following table.

Field	Read/Write Access	Description
<code>tnext</code>	Write only	Used to schedule a callback during the set time value. This field is equivalent to old <code>tnext</code> . For example: <pre>hdl_instance_obj.tnext = hdl_instance_obj.tnow + 5e-9</pre> will schedule a callback at time equals 5 nanoseconds from <code>tnow</code> .
<code>userdata</code>	Read/Write	Stores state variables of the current <code>matlabtb</code> instance. You can retrieve the variables the next time the callback of this instance is scheduled.
<code>simstatus</code>	Read only	Stores the status of the HDL simulator. The EDA Simulator Link software sets this parameter to 'Init' during the first callback for this particular instance and to 'Running' thereafter. <code>simstatus</code> is a read-only property. <pre>>> hdl_instance_obj.simstatus</pre> ans= Init

Field	Read/Write Access	Description
instance	Read only	<p>Stores the full path of the Verilog/VHDL instance associated with the callback. <code>instance</code> is a read-only property. The value of this field equals that of the module instance specified with the function call. For example:</p> <p>In the HDL simulator:</p> <pre>hdlsim> matlabtb osc_top -mfunc oscfilter use_instance_obj</pre> <p>In MATLAB:</p> <pre>>> hdl_instance_obj.instance ans= osc_top</pre>
argument	Read only	<p>Stores the argument set by the <code>-argument</code> option of <code>matlabtb</code>. For example:</p> <pre>matlabtb osc_top -mfunc oscfilter -use_instance_obj -argument foo</pre> <p>The link software supports the <code>-argument</code> option only when it is used with <code>-use_instance_obj</code>, otherwise the argument is ignored. <code>argument</code> is a read-only property.</p> <pre>>> hdl_instance_obj.argument ans= foo</pre>

Field	Read/Write Access	Description
portinfo	Read only	<p>Stores information about the VHDL and Verilog ports associated with this instance. portinfo is a read-only property, which has a field structure that describes the ports defined for the associated HDL module. For each port, the portinfo structure passes information such as the port's type, direction, and size. For more information on port data, see "Gaining Access to and Applying Port Information".</p> <pre data-bbox="709 644 1240 666">hdl_instance_obj.portinfo.field1.field2.field3</pre> <hr/> <p>Note When you use use_instance_obj, you access tscale through the HDL instance object. If you do not use use_instance_obj, you can still access tscale through portinfo.</p> <hr/>
tscale	Read only	<p>Stores the resolution limit (tick) in seconds of the HDL simulator. tscale is a read-only property.</p> <pre data-bbox="709 1008 1092 1124">>> hdl_instance_obj.tscale ans= 1.0000e-009</pre>

Field	Read/Write Access	Description
		<hr/> <p>Note When you use <code>use_instance_obj</code>, you access <code>tscale</code> through the HDL instance object. If you do not use <code>use_instance_obj</code>, you can still access <code>tscale</code> through <code>portinfo</code>.</p> <hr/>
<code>tnow</code>	Read only	<p>Stores the current time. <code>tnow</code> is a read-only property.</p> <pre>hdl_instance_obj.tnext = hld_instance_obj.tnow + fastestrate;</pre>
<code>portvalues</code>	Read/Write	<p>Stores the current values of and sets new values for the output and input ports for a <code>matlabtb</code> instance. For example:</p> <pre>>> hdl_instance_obj.portvalues ans = Read/Write Input ports: clk_enable: [] clk: [] reset: [] Read Only Output ports: sine_out: [22x1 char]</pre>

Field	Read/Write Access	Description
		<p>For example, you can set the reset port to 1 by calling <code>hdl_instance_obj.portvalues.reset = '1'</code>.</p>
linkmode	Read only	<p>Stores the status of the callback. The EDA Simulator Link software sets this parameter to 'testbench' if the callback is associated with <code>matlabtb</code> and 'component' if the callback is associated with <code>matlabcp</code>. <code>linkmode</code> is a read-only property.</p> <pre data-bbox="709 805 1121 927">>> hdl_instance_obj.linkmode ans= testbench</pre>

-argument

Used to pass user-defined arguments from the `matlabtb` instantiation on the HDL side to the MATLAB function callbacks. Supported with `-use_instance_obj` only. See the field listing for argument under the `-use_instance_obj` property.

Examples

The following examples demonstrate some ways you might use the `matlabtb` function.

Using matlabtb with the -socket Argument and Time Parameters

The following command starts the HDL simulator client component of EDA Simulator Link, associates an instance of the entity, `myfirfilter`, with the MATLAB function `myfirfilter`, and begins a local TCP/IP socket-based test bench session using TCP/IP port 4449. Based on the specified test bench stimuli, `myfirfilter.m` executes 5 nanoseconds from the current time, and then repeatedly every 10 nanoseconds:

```
hdlsim> matlabtb myfirfilter 5 ns -repeat 10 ns -socket 4449
```

Applying Rising Edge Clocks and State Changes with matlabtb

The following command starts the HDL simulator client component of EDA Simulator Link, and begins a remote TCP/IP socket-based session using remote MATLAB host `compb` and TCP/IP port 4449. Based on the specified test bench stimuli, `myfirfilter.m` executes 10 nanoseconds from the current time, each time the signal `/top/fclk` experiences a rising edge, and each time the signal `/top/din` changes state.

```
hdlsim> matlabtb /top/myfirfilter 10 ns -rising /top/fclk -sensitivity /top/din  
-socket 4449@computer123
```

Specifying a MATLAB Function Name and Sensitizing Signals with matlabtb

The following command starts the HDL simulator client component of the EDA Simulator Link software. The `'-mfunc'` option specifies the MATLAB function to connect to and the `'-socket'` option specifies the port number for socket connection mode. `'-sensitivity'` indicates that the test bench session is sensitized to the signal `sine_out`.

```
hdlsim> matlabtb osc_top -sensitivity /osc_top/sine_out  
-socket 4448 -mfunc hosctb
```

Purpose Call specified MATLAB function once and immediately on behalf of instantiated HDL module

Syntax `matlabtbeval <instance> [-socket <tcp_spec>]
[-mfunc <name>]`

Description The `matlabtbeval` command has the following characteristics:

- Starts the HDL simulator client component of the EDA Simulator Link software.
- Associates a specified instance of an HDL design created in the HDL simulator with a MATLAB function.
- Executes the specified MATLAB function once and immediately on behalf of the specified module instance.

This command is issued in the HDL simulator.

Note The `matlabtbeval` command executes the MATLAB function immediately, while `matlabtb` provides several options for scheduling MATLAB function execution.

Notes The communication mode that you specify for `matlabtbeval` must match the communication mode you specified for `hdldaemon` when you established the server connection.

For socket communications, specify the port number you selected for `hdldaemon` when you issue a link request with the `matlabtbeval` command in the HDL simulator.

Arguments `<instance>`
Specifies the instance of an HDL module that is associated with a MATLAB function. By default, `matlabtbeval` associates the

HDL module instance with a MATLAB function that has the same name as the HDL module instance. For example, if the HDL module instance is `myfirfilter`, `matlabtbeval` associates the HDL module instance with the MATLAB function `myfirfilter`. Alternatively, you can specify a different MATLAB function with the `-mfunc` property.

`-socket <tcp_spec>`

Specifies TCP/IP socket communication for the link between the HDL simulator and MATLAB. For TCP/IP socket communication on a single computer, the `<tcp_spec>` can consist of just a TCP/IP port number or service name (alias). If you are setting up communication between computers, you must also specify the name or Internet address of the remote host. See “Specifying TCP/IP Values” for some valid `tcp_spec` examples.

For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports”.

If you run the HDL simulator and MATLAB on the same computer, you have the option of using shared memory for communication. Shared memory is the default mode of communication and takes effect if you do not specify `-socket <tcp-spec>` on the command line.

Note The communication mode that you specify with the `matlabtbeval` command must match what you specify for the communication mode when you call the `hdldaemon` command to start the MATLAB server. For more information on communication modes, see “Communications for HDL Cosimulation”. For more information on establishing the MATLAB end of the communication link, see “Starting the HDL Simulator from MATLAB”.

`-mfunc <name>`

The name of the associated MATLAB function. If you omit this argument, `matlabtbeval` associates the HDL module instance with a MATLAB function that has the same name as the HDL module instance.. If you omit this argument and `matlabtbeval` does not find a MATLAB function with the same name, the command displays an error message.

Examples

This example starts the HDL simulator client component of the link software, associates an instance of the module `myfirfilter` with the function `myfirfilter.m`, and uses a local TCP/IP socket-based communication link to TCP/IP port 4449 to execute the function `myfirfilter.m`:

```
> matlabtbeval myfirfilter -socket 4449:
```

mvl2dec

Purpose Convert multivalued logic to decimal

Syntax `mvl2dec('mv_logic_string')`
`mvl2dec('mv_logic_string', signed)`

Description `mvl2dec('mv_logic_string')` converts a multivalued logic string to a positive decimal. If *mv_logic_string* contains any character other than '0' or '1', NaN is returned. *mv_logic_string* must be a vector.

`mvl2dec('mv_logic_string', signed)` converts a multivalued logic string to a positive or a negative decimal. If *signed* is true, this function assumes the first character *mv_logic_string*(1) to be a signed bit of a 2s complement number. If *signed* is missing or false, the multivalued logic string becomes a positive decimal.

Examples The following function call returns the decimal value 23:

```
mvl2dec('010111')
```

The following function call returns NaN:

```
mvl2dec('xxxxxx')
```

The following function call returns the decimal value -9:

```
mvl2dec('10111', true)
```

See Also `dec2mvl`

Purpose	Start and configure Cadence Incisive simulators for use with EDA Simulator Link software
Syntax	<code>nclaunch('PropertyName', 'PropertyValue'...)</code>
Description	<p><code>nclaunch('PropertyName', 'PropertyValue'...)</code> starts the Cadence Incisive simulator for use with the MATLAB and Simulink features of the EDA Simulator Link software. The first folder in the Cadence Incisive simulator matches your MATLAB current folder if you do not specify an explicit <code>rundir</code> parameter.</p> <p>After you call this function, you can use EDA Simulator Link functions for the HDL simulator (for example, <code>hdlsimmatlab</code>, <code>hdlsimulink</code>) to do interactive debug setup.</p> <p>The property name/property value pair settings allow you to customize the Tcl commands used to start the Cadence Incisive simulator, the <code>ncsim</code> executable to be used, the path and name of the Tcl script that stores the start commands, and for Simulink applications, details about the mode of communication to be used by the applications. You must use a property name/property value pair with <code>nclaunch</code>.</p>
Property Name/ Property Value Pairs	<p><code>'hdlsimdir', 'pathname'</code> Specifies the path name to the Cadence Incisive simulator executable to be started. By default, the function uses the first version of the simulator that it finds on the system path (defined by the path variable) . Use this option to start different versions of the Cadence Incisive simulator or if the version of the simulator you want to run does not reside on the system path.</p> <p><code>'hdlsimexe', 'simexename'</code> Specifies the name of a Cadence Incisive simulator executable. By default, this function uses <code>'ncsim'</code>. You can specify a custom-built simulator executable with <code>'simexename.'</code></p> <p><code>'libdir', 'folder'</code> Specifies the folder containing MATLAB shared libraries. This property creates an entry in the startup Tcl file that points to the</p>

folder with the shared libraries needed for the Cadence Incisive simulator to communicate with MATLAB when the Cadence Incisive simulator runs on a machine that does not have MATLAB.

'libfile', 'library_file_name'

Specifies a particular library file. This value defaults to the version of the library file that was built using the same compiler that MATLAB itself uses. If the HDL simulator links other libraries, including SystemC libraries, that were built using a compiler supplied with the HDL simulator, you can specify an alternate library file with this property. See “Using the EDA Simulator Link Libraries for HDL Cosimulation” for versions of the library built using other compilers.

'rundir', 'dirname'

Specifies where to run the HDL simulator. By default, the function uses the current working folder.

The following conditions apply to this name/value pair:

- If the value of `dirname` is “TEMPDIR”, the function creates a temporary folder in which it runs the HDL simulator.
- If you specify `dirname` and the folder does *not* exist, you will get an error.

'runmode', 'mode'

Specifies how to start the HDL simulator. This property accepts the following valid values:

- 'Batch': Start the HDL simulator in the background with no window.
- 'Batch with Xterm': Run HDL simulator in a non-interactive Xterm window.
- 'CLI': Start the HDL simulator in an interactive terminal window.
- 'GUI': Start the HDL simulator with the SimVision graphical user interface.

This value defaults to 'GUI'.

'socketsimulink', 'tcp_spec'

Specifies TCP/IP socket communication for links between the Cadence Incisive simulator and Simulink. See “Specifying TCP/IP Values” for valid TCP/IP examples. For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports”.

If the Cadence Incisive simulator and Simulink run on the same computing system, you have the option of using shared memory for communication. Shared memory is the default mode of communication and takes effect if you do not specify `-socket <tcp-spec>` on the command line.

'starthdlsim', ['yes' | 'no']

Determines whether the Cadence Incisive simulator is launched. This parameter defaults to `yes`, which launches the Cadence Incisive simulator and creates a startup Tcl file. If you set `starthdlsim` to `no`, the function does not launch the Cadence Incisive simulator, but it still creates a startup Tcl file.

This startup Tcl file contains pointers to MATLAB and Simulink shared libraries. To run the Cadence Incisive simulator manually, see “Starting the HDL Simulator from MATLAB”.

'startupfile', 'pathname'

Each invocation of `nclaunch` creates a Tcl script that, when executed, compiles and launches the HDL simulator. By default, this function generates a filename of `compile_and_launch.tcl` in the folder specified by `rundir`. With this property, you can specify the name and location of the generated Tcl script. If the file name already exists, that file’s contents are overwritten. You can edit and use the generated file in a regular shell outside of MATLAB. For example:

```
sh> tclsh compile_and_launch.tcl
```

```
'tclstart', 'tcl_commands'
```

Specifies one or more Tcl commands to execute before the Cadence Incisive simulator launches. Specify a command string or a cell array of command strings. You must specify at least one command; otherwise, no action occurs.

Note You must type `exec` in front of non-Tcl system shell commands. For example:

```
exec -ncverilog -c +access+rw +linedebug top.v
hdlsimulink -gui work.top
```

Examples

The following function call sequence compiles the design and starts Simulink with a GUI from the "proj" folder with the model loaded. Simulink is instructed to communicate with the EDA Simulator Link interface on socket port 4449. All of these commands are specified in a single string as the property value to `tclstart`.

```
nclaunch(...
'tclstart',...
{'exec ncverilog -c +access+rw +linedebug top.v',...
'hdlsimulink -gui work.top'},...
'socketsimulink','4449',...
'rundir', '/proj');
```

In this next example, `tclcmd` is used to build the sequence of Tcl commands that are executed in a Tcl shell after calling `nclaunch` from MATLAB, as follows:

- `tclcmd{1}` compiles `vlogtestbench_top`.
- `tclcmd{2}` elaborates the model.
- `tclcmd{3}` calls `hdlsimmatlab` in gui mode and loads the elaborated `vlogtestbench_top` in the simulator.

The function executes the arguments being passed with `-input` (`matlabtb` and `run`) in the `ncsim` Tcl shell. In this example, `matlabcp` associates the function `vlogmatlabc` to the module instance `u_matlab_component`. It assumes that the `hdldaemon` in MATLAB is listening on port 32864. The `run` function will run 50 resolution units (ticks).

```
tclcmd{1} = 'exec ncvlog vlogtestbench_top.v'  
tclcmd{2} = 'exec ncelab -access +wc vlogtestbench_top'  
tclcmd{3} = ['hdlsimmatlab -gui vlogtestbench_top ' ...  
            '-input "@matlabcp vlogtestbench_top.u_matlab_component...  
                -mfunc vlogmatlabc -socket 32864}" '...  
            '-input "@run 50}'']  
nclaunch('hdlsimdir', 'local.IUS.glnx.tools.bin', 'tclstart',tclcmd);
```

The following example shows using the property `startupfile` to designate a Tcl script that the function then uses to start the HDL simulator from the Tcl shell.

In MATLAB:

```
nclaunch ('tclstart', `xxx', `startupfile', `mytclscript',...  
        `starthdlsim', `no')
```

In Tcl shell:

```
shell> tclsh mytclscript
```

nomatlabtb

Purpose End active MATLAB test bench and MATLAB component sessions

Syntax `nomatlabtb`

Description The `nomatlabtb` command ends all active MATLAB test bench and MATLAB component sessions that were previously initiated by `matlabtb` or `matlabcp` commands.

This command is issued in the HDL simulator.

Note This command should be called before shutting down `hdldaemon` or `hdldaemon` will block shutdown until the call occurs.

Examples The following command ends all MATLAB test bench and MATLAB component sessions:

```
> nomatlabtb
```

See Also `matlabtb`, `matlabcp`

Purpose	Send HDL simulator event and process IDs to MATLAB server
Syntax	<code>notifyMatlabServer <i>EventID</i> -socket <i>tcp-spec</i></code>
Description	<p><code>notifyMatlabServer <i>EventID</i> -socket <i>tcp-spec</i></code> sends the HDL simulator event ID and process identification (PID) to the MATLAB server (<code>hdldaemon</code>) using the specified connection methods (socket or shared memory). For MATLAB to receive this message, <code>hdldaemon</code> must be running with the same communication mode as specified with the <code>notifyMatlabServer</code> command. The event ID and the PID queue in <code>hdldaemon</code>. <code>notifyMatlabServer</code> is often used in conjunction with <code>waitForHdlClient</code> to make sure the HDL simulator is ready to begin or continue processing.</p> <p>This command issues in the HDL simulator.</p>
Inputs	<p><code>EventID</code></p> <p>Specifies the event ID to be sent to <code>hdldaemon</code>. The ID requires a positive number less than the maximum value of 32-bit signed integer. This parameter contains the event ID expected by the command <code>waitForHdlClient</code> in MATLAB.</p> <p>Default: 1</p> <p><code>socket <i>tcp_spec</i></code></p> <p>Specifies that TCP/IP socket communication be used for the link between the HDL simulator and MATLAB. For TCP/IP socket communication on a single computer, <code>tcp_spec</code> requires either a TCP/IP port number or service name (alias). To set up communication between computers, you must also specify the name or Internet address of the remote host that is running the MATLAB server (<code>hdldaemon</code>).</p> <p>When you omit the <code>socket</code> option, MATLAB and the HDL simulator use shared memory communication.</p>

notifyMatlabServer

Examples

In MATLAB, use the function `waitForHdlClient` to verify whether the HDL simulator event ID has been received. In the following example, the function returns the HDL Simulator PID if EventID = 5 is received within 100 seconds. If a time-out occurs, the function returns -1.

```
>> hdldaemon('socket',5002);  
...  
>> hdlpid = waitForHdlClient(100,5);
```

In the HDL simulator, issue the `notifyMatlabServer` command to send event ID 5 to `hdldaemon` running on the same machine using TCP/IP socket port 5002.

```
>> notifyMatlabServer 5 -socket 5002
```

See Also

`waitForHdlClient`

Purpose Block cosimulation until HDL simulator is ready for simulation

Syntax

```
pingHdlSim(timeout)
pingHdlSim(timeout, 'portnumber')
pingHdlSim(timeout, 'portnumber', 'hostname')
```

Description

`pingHdlSim(timeout)` blocks cosimulation by not returning until the HDL server loads or until the specified time-out occurs. `pingHdlSim` returns the process ID of the HDL simulator or -1 if a time-out occurs. You must enter a time-out value. You may find this function useful if you are trying to automate a cosimulation and need to know that the HDL server has loaded before your script continues the simulation.

`pingHdlSim(timeout, 'portnumber')` tries to connect to the local host on port *portnumber* and times out after *timeout* seconds you specify.

`pingHdlSim(timeout, 'portnumber', 'hostname')` tries to connect to the host *hostname* on port *portname*. It times out after *timeout* seconds you specify.

Examples

The following function call blocks further cosimulation until the HDL server loads or until 30 seconds have passed:

```
pingHdlSim(30)
```

If the server loads within 30 seconds, `pingHdlSim` returns the process ID. If it does not, `pingHdlSim` returns -1.

The following function call blocks further cosimulation on port 5678 until the HDL server loads or until 20 seconds have passed:

```
pingHdlSim(20, '5678')
```

The following function call blocks further cosimulation on port 5678 on host name `msuser` until the HDL server loads or until 20 seconds pass:

```
pingHdlSim(20, '5678', 'msuser')
```

setupxilinx tools

Purpose Configure MATLAB environment for use with Xilinx FPGA workflow

Syntax `setupxilinx tools`

Description `setupxilinx tools` performs the following tasks to ensure your environment is set up correctly to use the Xilinx FPGA workflow:

- Checks that the XILINX system environment variable is defined and points to a valid Xilinx ISE installation.
- Checks if your platform and Xilinx ISE version is supported for the FPGA workflow.
- Adds any paths to your MATLAB search path necessary for using the Xilinx FPGA workflow.

Examples Enter the following command at the MATLAB command prompt:

```
>setupxilinx tools
```

See Also `fpgamodelsetup` | `makefpgaproject`

Purpose	Execute Tcl command in Incisive or ModelSim simulator
Syntax	<pre>tclHdlSim(tclCmd) tclHdlSim(tclCmd, 'portNumber') tclHdlSim(tclCmd, 'portnumber', 'hostname')</pre>
Description	<p>tclHdlSim(tclCmd) executes a Tcl command on the Incisive or ModelSim simulator using a shared connection during a Simulink cosimulation session.</p> <p>tclHdlSim(tclCmd, 'portNumber') executes a Tcl command on the Incisive or ModelSim simulator by connecting to the local host on port <i>portNumber</i>.</p> <p>tclHdlSim(tclCmd, 'portnumber', 'hostname') executes a Tcl command on the Incisive or ModelSim simulator by connecting to the host <i>hostname</i> on port <i>portname</i>.</p> <p>The Incisive or ModelSim simulator must be connected to MATLAB and Simulink using the EDA Simulator Link software for this function to work (see either vsimulink or hdlSimulink).</p> <p>To execute a Tcl command on the Incisive or ModelSim simulator during a MATLAB cosimulation session, use hdlDaemon('tclcmd', 'command').</p>
Examples	<p>The following function call displays a message in the HDL simulator command window using port 5678 on host name msuser:</p> <pre>tclHdlSim('puts "Done"', '5678', 'msuser')</pre>
See Also	hdlDaemon, launchDiscovery, nclaunch, vsim

vsim

Purpose Start and configure ModelSim for use with EDA Simulator Link

Syntax `vsim('PropertyName', 'PropertyValue'...)`

Description `vsim('PropertyName', 'PropertyValue'...)` starts and configures the ModelSim simulator (`vsim`) for use with the MATLAB and Simulink features of EDA Simulator Link. The first folder in ModelSim matches your MATLAB current folder.

After you call this function, you can use EDA Simulator Link functions for the HDL simulator (for example, `vsimmatlab`, `vsimulink`) to perform the following actions:

- Load instances of VHDL entities or Verilog modules for simulations that use MATLAB for verification
- Load instances of VHDL entities or Verilog modules for simulations that use Simulink for cosimulation

The property name/property value pair settings allow you to customize the Tcl commands used to start ModelSim, the `vsim` executable to be used, the path and name of the DO file that stores the start commands, and for Simulink applications, details about the mode of communication to be used by the applications.

Tip Use `pingHdlSim` to add a pause between the call to `vsim` and the call to actually run the simulation when you are attempting to automate the cosimulation.

Property Name/Property Value Pairs `'libdir', 'folder'`
Specifies the folder containing MATLAB library files. This property creates an entry in the startup Tcl file that points to the folder with the libraries needed for ModelSim to communicate with MATLAB when ModelSim runs on a machine that does not have MATLAB.

'libfile', 'library_file_name'

Specifies a particular library file. This value defaults to the version of the library file that was built using the same compiler that MATLAB itself uses. If the HDL simulator links other libraries, including SystemC libraries, that were built using a compiler supplied with the HDL simulator, you can specify an alternate library file with this property. See “Using the EDA Simulator Link Libraries for HDL Cosimulation” for versions of the library built using other compilers.

'pingTimeout', 'seconds'

Time to wait, in seconds, for the HDL simulator to start. Specify 0 (the default) to immediately return without waiting.

'rundir', 'dirname'

Specifies where to run ModelSim. By default, the function uses the current working folder.

The following conditions apply to this name/value pair:

- If the value of `dirname` is “TEMPDIR”, the function creates a temporary folder in which it runs ModelSim.
- If you specify `dirname` and the folder does *not* exist, you will get an error.

'runmode', 'mode'

Specifies how to start the HDL simulator. This property accepts the following valid values:

- 'Batch': Start the HDL simulator in the background with no window (Linux) or in a non-interactive command window (Windows).
- 'CLI': Start the HDL simulator in an interactive terminal window.
- 'GUI': Start the HDL simulator with the ModelSim graphical user interface.

This value defaults to 'GUI'.

'socketsimulink', 'tcp_spec'

Specifies TCP/IP socket communication for links between ModelSim and Simulink. For TCP/IP socket communication on a single computing system, the `tcp_spec` can consist of just a TCP/IP port number or service name. If you are setting up communication between computing systems, you must also specify the name or Internet address of the remote host. See “Specifying TCP/IP Values” for some valid `tcp_spec` examples.

For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports”

If ModelSim and Simulink run on the same computing system, you have the option of using shared memory for communication. Shared memory is the default mode of communication and takes effect if you do not specify `-socket <tcp-spec>` on the command line.

Note The function applies the communication mode specified by this property to all invocations of Simulink from ModelSim.

'startms', ['yes' | 'no']

Determines whether ModelSim will launch from `vsim`. This property defaults to `yes`, which launches ModelSim and creates a startup Tcl file. If `startms` is set to `no`, ModelSim does not launch, but the HDL simulator still creates a startup Tcl file.

This startup Tcl file contains pointers to MATLAB libraries. To run ModelSim on a machine without MATLAB, copy the startup Tcl file and MATLAB library files to the remote machine and start ModelSim manually. See “Using the EDA Simulator Link Libraries for HDL Cosimulation”.

'startupfile', 'pathname'

Each invocation of vsim creates a Tcl script that, when executed, compiles and launches the HDL simulator. By default, this function generates the filename of compile_and_launch.tcl in the folder specified by rundir.. With this property, you can specify the name and location of the generated Tcl script. If the file name already exists, that file's contents are overwritten. You can edit and use the generated file in a regular shell outside of MATLAB. For example:

```
sh> vsim -gui -do compile_and_launch.tcl
```

'tclstart', 'tcl_commands'

Specifies one or more Tcl commands to execute after ModelSim launches. Specify a command string or a cell array of command strings.

'vsimdir', 'pathname'

Specifies the path name to the ModelSim simulator executable (vsim.exe) to be started. By default, the function uses the first version of vsim.exe that it finds on the system path (defined by the path variable) . Use this option to start different versions of the ModelSim simulator or if the version of the simulator you want to run does not reside on the system path.

Examples

The following function call sequence changes the folder location to VHDLproj and then calls the function vsim. Because the call to vsim omits the 'vsimdir' and 'startupfile' properties, vsim uses the default vsim executable and creates a temporary DO file in a temporary folder. The 'tclstart' property specifies a Tcl command that loads an instance of a VHDL entity for MATLAB verification:

- The vsimmatlab command loads an instance of the VHDL entity parse in the library work for MATLAB verification.
- The matlabtb command begins the test bench session for an instance of entity parse, using TCP/IP socket communication on port 4449 and a test bench timing value of 10 ns.

```
cd VHDLproj % Change folder to ModelSim project folder
vsim('tclstart','vsimmatlab work.parse; matlabtb parse 10 ns -socket 4449')
```

The following function call sequence changes the folder location to VHDLproj and then calls the function vsim.

- Because the call to vsim omits the 'vsimdir' and 'startupfile' properties, vsim uses the default vsim executable and creates a DO file in a temporary folder.
- The 'tclstart' property specifies a Tcl command that loads the VHDL entity parse in the library work for cosimulation between vsim and Simulink.
- The 'socketsimulink' property specifies that TCP/IP socket communication on the same computer is to be used for links between Simulink and ModelSim, using socket port 4449.

```
cd VHDLproj % Change folder to ModelSim project folder
vsim('tclstart','vsimulink work.parse','socketsimulink','4449')
```


Purpose	Load instantiated HDL module for verification with ModelSim and MATLAB
Syntax	<code>vsimmatlab <instance> [<vsim_args>]</code>
Description	<p>The <code>vsimmatlab</code> command loads the specified instance of an HDL module for verification and sets up ModelSim so it can establish a communication link with MATLAB. ModelSim opens a simulation workspace and displays a series of messages in the command window as it loads the HDL module's packages and architectures.</p> <p>This command is generally issued in the HDL simulator. It also may be run from the HDL simulator prompt or from a Tcl script shell (<code>tclsh</code>).</p>
Arguments	<p><code><instance></code> Specifies the instance of an HDL module to load for verification.</p> <p><code><vsim_args></code> Specifies one or more ModelSim <code>vsim</code> command arguments. For details, see the description of <code>vsim</code> in the ModelSim documentation.</p>
Examples	<p>The following command loads the HDL module instance <code>parse</code> from library <code>work</code> for verification and sets up ModelSim so it can establish a communication link with MATLAB:</p> <pre>ModelSim> vsimmatlab work.parse</pre>

vsimulink

Purpose Load instantiated HDL module for cosimulation with ModelSim and Simulink

Syntax vsimulink <instance>
[-socket <tcp_spec>] [<vsim_args>]

Description The vsimulink command loads the specified instance of an HDL module for cosimulation and sets up ModelSim so it can establish a communication link with Simulink. ModelSim opens a simulation workspace and displays a series of messages in the command window as it loads the HDL module's packages and architectures.

This command is issued in the HDL simulator.

Argument <instance>
Specifies the instance of an HDL module to load for cosimulation.

-socket <tcp_spec>
Specifies TCP/IP socket communication for the link between ModelSim and MATLAB. This setting overrides the setting specified with the MATLAB vsim function. The <tcp_spec> can consist of a TCP/IP socket port number or service name (alias). For example, you might specify port number 4449 or the service name matlabservice.

For more information on choosing TCP/IP socket ports, see "Specifying TCP/IP Values".

If you run the HDL simulator and MATLAB on the same computer, you have the option of using shared memory for communication. Shared memory is the default mode of communication and takes effect if you do not specify -socket <tcp-spec> on the command line.

Note The communication mode that you specify with the `vsimulink` command must match what you specify for the communication mode when you configure EDA Simulator Link blocks in your Simulink model. For more information on modes of communication, see “Communications for HDL Cosimulation”. For more information on establishing the Simulink end of the communication link, see “Configuring the Communication Link in the HDL Cosimulation Block”

<vsim_args>

Specifies one or more ModelSim `vsim` command arguments. For details, see the description of `vsim` in the ModelSim documentation.

Examples

The following command loads the HDL module instance `parse` from library `work` for cosimulation and sets up ModelSim so it can establish a communication link with Simulink:

```
ModelSim> vsimulink work.parse
```

waitForHdlClient

Purpose Wait until specified event ID is obtained or time-out occurs

Syntax

```
waitForHdlClient(TimeOut,EventID)  
waitForHdlClient(TimeOut)  
waitForHdlClient  
output = waitForHdlClient(TimeOut,EventID)
```

Description

`waitForHdlClient(TimeOut,EventID)` waits for the expected HDL simulator event ID to arrive at the MATLAB server (`hdldaemon`) before processing continues. If the expected event ID arrives before the number of seconds specified by the *TimeOut* parameter, the value returned by the HDL simulator is the HDL simulator process identification (PID). Otherwise, the returned value is `-1`.

`waitForHdlClient(TimeOut)` waits for `EventID = 1` for *TimeOut* seconds.

`waitForHdlClient` waits for `EventID = 1` for 60 seconds.

`output = waitForHdlClient(TimeOut,EventID)` returns the process identification (PID) in *output*. Although you are not required to provide an output variable, MATLAB returns an error if a time-out occurs and the output argument is not specified.

Inputs

TimeOut

Number of seconds to wait for a response from the HDL simulator

EventID

The HDL simulator event ID. *EventID* must be a positive number less than the maximum value of a 32-bit signed integer. The value should match the event ID sent by the `notifyMatlabServer` command in the HDL simulator.

EventID can be either a scalar or vector value. If *EventID* is a vector, the function return a value only if all elements of the vector have been collected or if a time-out occurs. The returned output value is the same size as the event ID, and each element of

the output variable is the detected PID of the HDL simulator that sends the corresponding event ID element.

Outputs

output

Output variable for holding returned value from call to `waitForHdlClient`. Contains either the HDL simulator process identification (PID) or `-1` if an error occurs.

Examples

Wait for event ID 2 for 120 seconds.

```
>> hdlpid = waitForHdlClient(120, 2);
```

See Also

`notifyMatlabServer`

wrapverilog

Purpose Apply VHDL wrapper to Verilog module

Syntax wrapverilog [-nocompile] <verilog_module>

Description

Note wrapverilog will be removed in a future version. EDA Simulator Link now supports Verilog models directly, without requiring a VHDL wrapper.

The wrapverilog command applies a VHDL wrapper to the specified Verilog module and then automatically compiles the resulting VHDL file. You can then use your wrapped Verilog module with EDA Simulator Link.

This command is issued in the HDL simulator.

Before executing the wrapverilog command on a Verilog file, you must compile and load the Verilog module in ModelSim, as in the following example.

```
vlib work
vmap work work
vlog myverilogmod.v
vsim myverilogmod
wrapverilog [-nocompile] myverilogmod
```

Arguments

<verilog_module>

Specifies the Verilog module to which a VHDL wrapper is to be applied. The module you specify must be in a valid ModelSim design library when you issue the command.

-nocompile

Suppresses automatic compilation of the resulting VHDL file, *verilog_module_wrap.vhd*.

Examples

The following command applies a VHDL wrapper to Verilog module `myverilogmod.v` and writes the output to `myverilogmod_wrap.vhd`. The `-nocompile` option suppresses automatic compilation.

```
ModelSim> wrapverilog -nocompile myverilogmod
```


A

- action property
 - description of 4-3
- arguments
 - for hdlSimmatlab command 4-18
 - for hdlSimulink command 4-19
 - for matlabcp command 4-35
 - for matlabb command 4-48
 - for matlabbbeval command 4-61
 - for pingHdlSim function 4-73
 - for tclHdlSim function 4-75
 - for vsimmatlab command 4-81
 - for vsimulink command 4-82
- Auto fill
 - in Ports pane of HDL Cosimulation block 2-2

B

- block input ports parameter
 - description of 2-2
- block output ports parameter
 - description of 2-2
- blocks
 - HDL Cosimulation
 - description of 2-2
 - To VCD File
 - description of 2-26

C

- cancel option 4-48
- Clocks pane
 - description of 2-2
- configuremodelsim function
 - description of 4-3
- Connection pane
 - description of 2-2
- Cosimulation timing
 - absolute mode 2-2
 - relative mode 2-2

D

- dec2mvl function
 - description of 4-9
- dialogs
 - for HDL Cosimulation block 2-2
 - for To VCD File block 2-26
- direct feedthrough option
 - for eliminating block latency 2-2

E

- examples
 - configuremodelsim function 4-3
 - dec2mvl function 4-9
 - hdlSimmatlab command 4-18
 - hdlSimulink command 4-19
 - launchDiscovery function 4-21
 - matlabcp command 4-35
 - matlabb command 4-48
 - matlabbbeval command 4-61
 - mv12dec function 4-64
 - nclaunch function 4-65
 - nomatlabb command 4-70
 - pingHdlSim function 4-73
 - tclHdlSim function 4-75
 - vsim function 4-76
 - vsimmatlab command 4-81
 - vsimulink command 4-82

F

- falling option 4-48
- falling-edge clocks
 - description of 2-2
- files
 - VCD 2-29
- FPGA implementation functions
 - reference for 3-4
- functions 4-1
 - hdlSimmatlab

- description of 4-18
- hdlsimulink
 - description of 4-19
- makefpga project 4-32
- matlabcp
 - description of 4-35
- matlabtb
 - description of 4-48
- matlabtbeval
 - description of 4-61
- nomatlabtb 4-70
- notifyMatlabServer 4-71
- setupxilinx tools 4-74
- tdkfgasetup 4-10
- waitForHdlClient 4-84
- See also* MATLAB functions
- functions for generating FPGA projects
 - makefpga project 4-32
 - setupxilinx tools 4-74
 - tdkfgasetup 4-10

G

- generating FPGA projects
 - from command line 4-32

H

- HDL Cosimulation block
 - description of 2-2
- HDL cosimulation functions
 - reference for 3-2
- HDL cosimulation library
 - reference for 1-2
- HDL simulator running on this computer
 - parameter
 - description of 2-2
- hdlsimdir property
 - with launchDiscovery function 4-21
 - with nclaunch function 4-65

- hdlsimmatlab command
 - description of 4-18
- hdlsimulink command
 - description of 4-19
- Host name parameter
 - description of 2-2

I

- Incisive simulator commands
 - hdlsimmatlab
 - description of 4-18
- INOUT ports
 - specifying 2-2

L

- launchDiscovery function
 - description of 4-21

M

- MATLAB functions 4-1
 - configuremodelsim
 - description of 4-3
 - dec2mvl
 - description of 4-9
 - launchDiscovery
 - description of 4-21
 - mvl2dec
 - description of 4-64
 - nclaunch
 - description of 4-65
 - pingHdlSim
 - description of 4-73
 - tclHdlSim
 - description of 4-75
 - vsim
 - description of 4-76
- matlabcp command
 - description of 4-35

matlabtb command
 description of 4-48

matlabtbeval command
 description of 4-61

-mfunc option
 with matlabcp command 4-35
 with matlabtb command 4-48
 with matlabtbeval command 4-61

ModelSim commands
 vsimmatlab
 description of 4-81
 vsimulink
 description of 4-82

mv12dec function
 description of 4-64

N

nclaunch function
 description of 4-65

nomatlabtb command 4-70

Number of input ports parameter 2-26

Number of output ports parameter
 description of 2-26

O

options
 for hdlsimulink command 4-19
 for matlabcp command 4-35
 for matlabtb command 4-48
 for matlabtbeval command 4-61
 for vsimulink command 4-82
 property
 with configuremodelsim function 4-3
 with launchDiscovery function 4-21
 with nclaunch function 4-65
 with vsim function 4-76

Output sample time parameter
 description of 2-2

P

parameters
 for HDL Cosimulation block 2-2
 for To VCD File block 2-26

path specification
 for ports/signals and modules in Simulink
 with HDL Cosimulation block 2-2

pingHdlSim function
 description of 4-73

port names
 specifying paths in Simulink
 with HDL Cosimulation block 2-2

Port number or service parameter
 description of 2-2

Ports pane
 Auto fill option 2-2
 description of 2-2
 Enable direct feedthrough option 2-2

Post-simulation command parameter
 description of 2-2

properties
 action 4-3
 for configuremodelsim function 4-3
 for launchDiscovery function 4-21
 for nclaunch function 4-65
 for vsim function 4-76
 nclaunchdir
 with nclaunch function 4-65
 socketsimulink 4-21 4-65 4-76
 startupfile 4-21 4-65 4-76
 tclstart
 with configuremodelsim function 4-3
 with launchDiscovery function 4-21
 with nclaunch function 4-65
 with vsim function 4-76
 vsimdir
 with configuremodelsim function 4-3
 with vsim function 4-76

property option
 for configuremodelsim function 4-3

- for launchDiscovery function 4-21
- for nclaunch function 4-65
- for vsim function 4-76

Prsimulation command parameter

- description of 2-2

R

- repeat option 4-35
- rising option 4-35

rising-edge clocks

- description of 2-2

S

sending messages to MATLAB

- notifyMatlabServer function 4-71

- sensitivity option 4-35

setupxilinx tools

- function for generating FPGA projects 4-74

Shared memory parameter

- description of 2-2

signal names

- specifying paths in Simulink
 - with HDL Cosimulation block 2-2

signals

- read/write access required 2-2

- socket option
 - with hdl simulink command 4-19
 - with matlabcp command 4-35
 - with matlabb command 4-48
 - with matlabbtbeval command 4-61
 - with vsimulink command 4-82

socketsimulink property

- description of 4-21 4-65 4-76

startupfile property

- description of 4-21 4-65 4-76

T

Tcl commands

- added to startup script via
 - launchDiscovery 4-21
- added to startup script via nclaunch 4-65
- hdlsim matlab 4-18
- hdlsimulink 4-19
- specified in Tcl pane of HDL Cosimulation block 2-2

Tcl pane

- description of 2-2

tclHdlSim function

- description of 4-75

tclstart property

- with configuremodelsim function 4-3
- with launchDiscovery function 4-21
- with nclaunch function 4-65
- with vsim function 4-76

time scale, VCD file 2-29

Timescales pane

- description of 2-2

To VCD File block

- description of 2-26

V

VCD file name parameter

- description of 2-26

VCD files

- format of 2-29

Virtual Platform simulation functions

- reference for 3-5

vsim function

- description of 4-76

vsimdir property

- with configuremodelsim function 4-3
- with vsim function 4-76

vsimmatlab command

- description of 4-81

vsimulink command

- description of 4-82

W

waiting for the HDL simulator

waitForHdlClient function 4-84