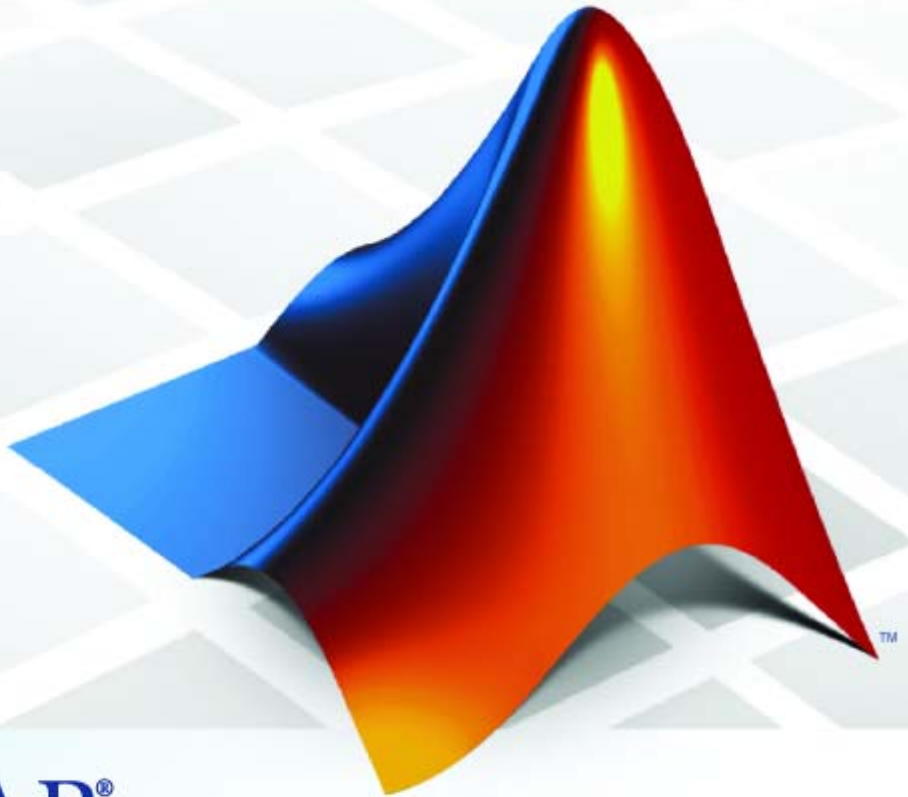


# EDA Simulator Link™ MQ 2

## User's Guide



**MATLAB®**  
& **SIMULINK®**

## How to Contact The MathWorks



[www.mathworks.com](http://www.mathworks.com)  
[comp.soft-sys.matlab](mailto:comp.soft-sys.matlab@mathworks.com)  
[www.mathworks.com/contact\\_TS.html](http://www.mathworks.com/contact_TS.html)

Web  
Newsgroup  
Technical Support



[suggest@mathworks.com](mailto:suggest@mathworks.com)  
[bugs@mathworks.com](mailto:bugs@mathworks.com)  
[doc@mathworks.com](mailto:doc@mathworks.com)  
[service@mathworks.com](mailto:service@mathworks.com)  
[info@mathworks.com](mailto:info@mathworks.com)

Product enhancement suggestions  
Bug reports  
Documentation error reports  
Order status, license renewals, passcodes  
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™ MQ User's Guide*

© COPYRIGHT 2003–2008 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](http://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](http://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)



## Getting Started

# 1

<b>Product Overview</b> .....	<b>1-3</b>
Integration with Other Products .....	1-3
Linking with MATLAB and the HDL Simulator .....	1-5
Linking with Simulink and the HDL Simulator .....	1-7
Communicating with MATLAB or Simulink and the HDL Simulator .....	1-8
<b>Requirements</b> .....	<b>1-9</b>
What You Need to Know .....	1-9
Required Products .....	1-9
<b>Setting Up Your Environment for the EDA Simulator</b>	
<b>Link™ MQ Software</b> .....	<b>1-12</b>
Installing the Link Software .....	1-12
Installing Related Application Software .....	1-12
Setting Up the HDL Simulator for Use with the Link Software .....	1-12
Using the EDA Simulator Link™ MQ Libraries .....	1-18
<b>Starting the HDL Simulator</b> .....	<b>1-23</b>
Starting ModelSim SE/PE from MATLAB .....	1-23
Starting the ModelSim Software from a Shell .....	1-25
<b>Workflow for Using the EDA Simulator Link™ MQ Software with MATLAB® Software</b> .....	<b>1-26</b>
<b>Workflow for Using the EDA Simulator Link™ MQ Software with Simulink® Software</b> .....	<b>1-27</b>
<b>Learning More About the EDA Simulator Link™ MQ Software</b> .....	<b>1-28</b>
Documentation Overview .....	1-28
Online Help .....	1-29

Demos and Tutorials .....	1-30
---------------------------	------

## Linking MATLAB® to ModelSim® Simulators

# 2

<b>MATLAB®-ModelSim® Workflow .....</b>	<b>2-2</b>
<b>Coding an EDA Simulator Link™ MQ MATLAB®</b>	
<b>Application .....</b>	<b>2-4</b>
Overview .....	2-4
Process for Coding an EDA Simulator Link™ MQ MATLAB	
Application .....	2-5
Coding HDL Modules for MATLAB Verification .....	2-7
Coding MATLAB Link Functions .....	2-12
<b>Associating a MATLAB® Link Function with an HDL</b>	
<b>Module .....</b>	<b>2-37</b>
Overview .....	2-37
Naming a MATLAB Link Function .....	2-37
Associating the HDL Module Component with the MATLAB	
Link Function .....	2-38
Specifying HDL Signal/Port and Module Paths for MATLAB	
Link Sessions .....	2-38
Specifying TCP/IP Values .....	2-40
Scheduling Options for a Link Session .....	2-40
<b>Running MATLAB® Link Sessions .....</b>	<b>2-48</b>
Overview .....	2-48
Process for Running MATLAB Link Sessions .....	2-48
Placing a MATLAB Test Bench or Component Function on	
the MATLAB Search Path .....	2-49
Starting the MATLAB Server .....	2-49
Checking the MATLAB Server's Link Status .....	2-51
Starting ModelSim SE/PE for Use with MATLAB .....	2-51
Applying Stimuli with the HDL Simulator force	
Command .....	2-51
Running a Link Session .....	2-52
Restarting a Link Session .....	2-54
Stopping a Link Session .....	2-54

<b>MATLAB and ModelSim Tutorial</b> .....	<b>2-56</b>
Tutorial Overview .....	<b>2-56</b>
Setting Up Tutorial Files .....	<b>2-56</b>
Starting the MATLAB Server .....	<b>2-57</b>
Setting Up ModelSim .....	<b>2-58</b>
Developing the VHDL Code .....	<b>2-60</b>
Compiling the VHDL File .....	<b>2-63</b>
Loading the Simulation .....	<b>2-63</b>
Developing the MATLAB Function .....	<b>2-66</b>
Running the Simulation .....	<b>2-68</b>
Shutting Down the Simulation .....	<b>2-72</b>

## Linking Simulink® to ModelSim® Simulators

# 3

<b>Simulink®-ModelSim® Workflow</b> .....	<b>3-2</b>
<b>Introduction to Cosimulation</b> .....	<b>3-5</b>
Creating a Hardware Model Design for Use in Simulink® Applications .....	<b>3-5</b>
The EDA Simulator Link™ MQ HDL Cosimulation Block .....	<b>3-8</b>
Communicating Between the HDL Simulator and Simulink® Software .....	<b>3-12</b>
<b>Preparing for Cosimulation</b> .....	<b>3-14</b>
Overview .....	<b>3-14</b>
How Simulink Drives Cosimulation Signals .....	<b>3-15</b>
Representation of Simulation Time .....	<b>3-15</b>
Handling Multirate Signals .....	<b>3-24</b>
Handling Frame-Based Signals .....	<b>3-24</b>
Avoiding Race Conditions in HDL Simulation .....	<b>3-32</b>
Block Simulation Latency .....	<b>3-32</b>
Interfacing with Continuous Time Signals .....	<b>3-37</b>
Setting Simulink Software Configuration Parameters ....	<b>3-38</b>
Simulink and HDL Simulator Communication Options ...	<b>3-39</b>
Starting the HDL Simulator .....	<b>3-39</b>

<b>Incorporating Hardware Designs into a Simulink®</b>	
<b>Model</b> .....	<b>3-40</b>
Overview .....	<b>3-40</b>
Specifying HDL Signal/Port and Module Paths for	
Cosimulation .....	<b>3-41</b>
Driving Clocks, Resets, and Enables .....	<b>3-43</b>
Defining the Block Interface .....	<b>3-45</b>
Specifying the Signal Datatypes .....	<b>3-56</b>
Configuring the Simulink and ModelSim SE/PE Timing	
Relationship .....	<b>3-58</b>
Configuring the Communication Link in the HDL	
Cosimulation Block .....	<b>3-59</b>
Specifying Pre- and Post-Simulation Tcl Commands with	
HDL Cosimulation Block Parameters Dialog Box .....	<b>3-62</b>
Programmatically Controlling the Block Parameters .....	<b>3-64</b>
Adding a Value Change Dump (VCD) File .....	<b>3-66</b>
<b>Running Cosimulation Sessions</b> .....	<b>3-70</b>
Starting the HDL Simulator for Use with Simulink .....	<b>3-70</b>
Determining an Available Socket Port Number .....	<b>3-71</b>
Checking the Connection Status .....	<b>3-71</b>
Managing a Simulink Cosimulation Session .....	<b>3-71</b>
<b>Simulink and ModelSim Tutorial</b> .....	<b>3-72</b>
Tutorial Overview .....	<b>3-72</b>
Developing the VHDL Code .....	<b>3-72</b>
Compiling the VHDL File .....	<b>3-73</b>
Creating the Simulink Model .....	<b>3-75</b>
Setting Up ModelSim for Use with Simulink .....	<b>3-84</b>
Loading Instances of the VHDL Entity for Cosimulation	
with Simulink .....	<b>3-85</b>
Running the Simulation .....	<b>3-86</b>
Shutting Down the Simulation .....	<b>3-89</b>
<b>toVCD Block Tutorial</b> .....	<b>3-90</b>



**EDA Simulator Link™ MQ MATLAB® Function  
Reference**

---

**4**

**EDA Simulator Link™ MQ Command Extensions  
for the HDL Simulator Reference**

---

**5**

**EDA Simulator Link™ MQ Simulink® Block  
Reference**

---

**6**

**Examples**

---

**A**

<b>Invoking ModelSim Using the MATLAB Function vsim .....</b>	<b>A-2</b>
<b>MATLAB and ModelSim Random Number Generator Tutorial .....</b>	<b>A-2</b>
<b>Frame-Based Processing .....</b>	<b>A-2</b>
<b>Simulink and ModelSim Inverter Tutorial .....</b>	<b>A-2</b>
<b>Generating a VCD File .....</b>	<b>A-3</b>

**B**

<b>Adding Libraries for ADMS Support .....</b>	<b>B-2</b>
<b>Linking MATLAB® or Simulink® Software to ModelSim in ADMS .....</b>	<b>B-3</b>
Starting ADMS for Use with EDA Simulator Link™	
MQ .....	<b>B-3</b>
Using Tcl Test Bench Commands with ADMS .....	<b>B-4</b>
Constraints .....	<b>B-4</b>

**EDA Simulator Link™ MQ Machine  
Configuration Requirements**

---

**C**

<b>Valid Configurations For Using the EDA Simulator Link™ MQ Software with MATLAB® Applications ..</b>	<b>C-2</b>
<b>Valid Configurations For Using the EDA Simulator Link™ MQ Software with Simulink® Software .....</b>	<b>C-4</b>

**TCP/IP Socket Communication**

---

**D**

<b>Choosing TCP/IP Socket Ports .....</b>	<b>D-2</b>
<b>Specifying TCP/IP Values .....</b>	<b>D-5</b>
<b>TCP/IP Services .....</b>	<b>D-6</b>

## Race Conditions in HDL Simulators

**E**

<b>Overview</b> .....	<b>E-2</b>
<b>Potential Race Conditions in Simulink® Link Sessions</b> .....	<b>E-3</b>
<b>Potential Race Conditions in MATLAB® Link Sessions</b> .....	<b>E-5</b>
<b>Further Reading</b> .....	<b>E-6</b>

**Index**



# Getting Started

---

Product Overview (p. 1-3)

Identifies typical applications and expected users, lists key product features, describes the EDA Simulator Link™ MQ cosimulation environment, and provides an overview of how you work with the integrated tool environment

Requirements (p. 1-9)

Describes what you need to know and what other products are required to use the EDA Simulator Link MQ software

Setting Up Your Environment for the EDA Simulator Link™ MQ Software (p. 1-12)

Explains how to install and set up the EDA Simulator Link MQ software

Starting the HDL Simulator (p. 1-23)

Explains and shows how to invoke the HDL simulator so that it will work with EDA Simulator Link MQ software

Workflow for Using the EDA Simulator Link™ MQ Software with MATLAB® Software (p. 1-26)

Describes very basic steps for creating MATLAB—HDL simulator applications

Workflow for Using the EDA  
Simulator Link™ MQ Software with  
Simulink® Software (p. 1-27)

Describes very basic steps for  
creating Simulink—HDL simulator  
cosimulation sessions

Learning More About the EDA  
Simulator Link™ MQ Software  
(p. 1-28)

Identifies and explains how to gain  
access to available documentation  
online help, demo, and tutorial  
resources

# Product Overview

**In this section...**

“Integration with Other Products” on page 1-3

“Linking with MATLAB and the HDL Simulator” on page 1-5

“Linking with Simulink and the HDL Simulator” on page 1-7

“Communicating with MATLAB or Simulink and the HDL Simulator” on page 1-8

## Integration with Other Products

The EDA Simulator Link™ MQ cosimulation interface integrates MathWorks™ tools into the Electronic Design Automation (EDA) workflow for field programmable gate array (FPGA) and application-specific integrated circuit (ASIC) development. The software provides a fast bidirectional link between the Mentor Graphics hardware description language (HDL) simulator, ModelSim®, and The MathWorks™ MATLAB® and Simulink® products for direct hardware design verification and cosimulation. The integration of these tools allows users to apply each product to the tasks it does best:

- ModelSim — Hardware modeling in HDL and simulation
- MATLAB — Numerical computing, algorithm development, and visualization
- Simulink — Simulation of system-level designs and complex models

---

**Note** ModelSim software may also be referred to as "the HDL simulator" throughout this document.

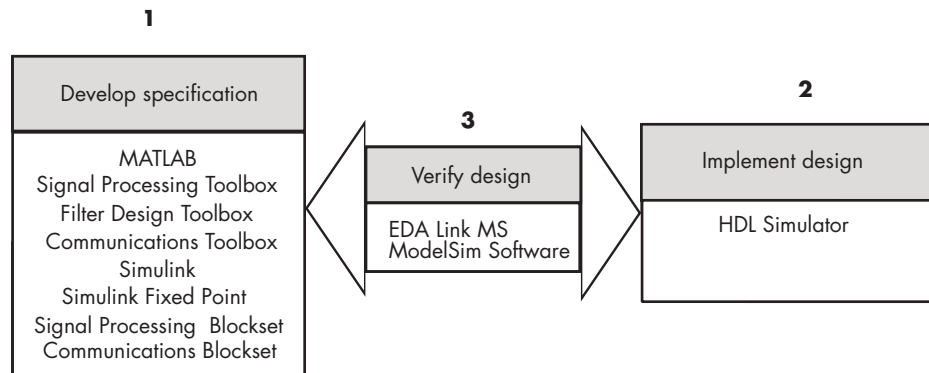
---

The EDA Simulator Link MQ software consists of MATLAB functions and HDL simulator commands for establishing the communication links between the HDL simulator and The MathWorks products. In addition, a library of Simulink blocks is available for including HDL simulator designs in Simulink models for cosimulation.

EDA Simulator Link MQ software streamlines FPGA and ASIC development by integrating tools available for

- 1 Developing specifications for hardware design reference models
- 2 Implementing a hardware design in HDL based on a reference model
- 3 Verifying the design against the reference design

The following figure shows how the HDL simulator and MathWorks products fit into this hardware design scenario.



As the figure shows, EDA Simulator Link MQ software connects tools that traditionally have been used discretely to perform specific steps in the design process. By connecting the tools, the link simplifies verification by allowing you to cosimulate the implementation and original specification directly. The end result is significant time savings and the elimination of errors inherent to manual comparison and inspection.

In addition to the preceding design scenario, EDA Simulator Link MQ software enables you to use the following:

- MATLAB or Simulink to create test signals and software test benches for HDL code
- MATLAB or Simulink to provide a behavioral model for an HDL simulation



- MATLAB analysis and visualization capabilities for real-time insight into an HDL implementation
- Simulink to translate legacy HDL descriptions into system-level views

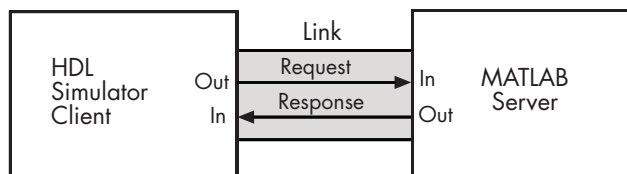
---

**Note** You can cosimulate a module using SystemVerilog and/or SystemC with MATLAB or Simulink using the EDA Simulator Link MQ software. Write simple wrappers around the SystemC and make sure that the SystemVerilog cosimulation connections are to ports or signals of data types supported by the link cosimulation interface.

---

## Linking with MATLAB and the HDL Simulator

When linked with MATLAB, the HDL simulator functions as the client, as the following figure shows.

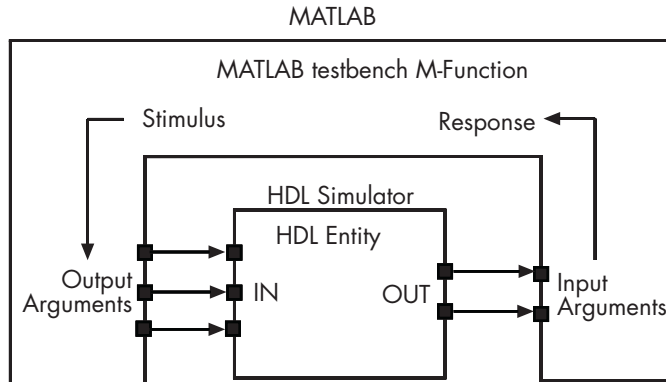


In this scenario, a MATLAB server function waits for service requests that it receives from an HDL simulator session. After receiving a request, the server establishes a communication link, and invokes a specified MATLAB function that computes data for, verifies, or visualizes the HDL module (coded in VHDL or Verilog) that is under simulation in the HDL simulator.

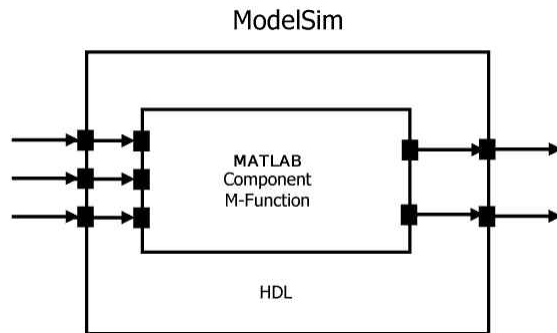
After the server is running, you can start and configure the HDL simulator or use with MATLAB with the supplied EDA Simulator Link MQ function `vsim`. Required and optional parameters allow you to specify the following:

- Tcl commands that execute as part of startup
- A specific ModelSim executable to start
- The name of a ModelSim DO file to store the complete startup script for future use or reference

The following figure shows how a MATLAB test bench function wraps around and communicates with the HDL simulator during a test bench simulation session.



The following figure shows how a MATLAB component function is wrapped around by and communicates with the HDL simulator during a component simulation session.



During the configuration process, EDA Simulator Link MQ software equips the HDL simulator with a set of customized command extensions you use to perform the following tasks:

- Load the HDL simulator with an instance of an HDL module to be tested with MATLAB

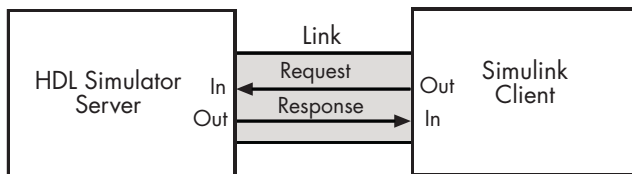
- Begin a MATLAB test bench or component session for that instance

When you begin a specific test bench session, you specify parameters that identify the following:

- The mode and, if appropriate, TCP/IP data necessary for connecting to a MATLAB server
- The MATLAB function that is associated with and executes on behalf of the HDL instance
- Timing specifications and other control data that specifies when the module's MATLAB function is to be called

## Linking with Simulink and the HDL Simulator

When linked with Simulink, the HDL simulator functions as the server, as shown in the following figure.



In this case, the HDL simulator responds to simulation requests it receives from cosimulation blocks in a Simulink model. You begin a cosimulation session from Simulink. After a session is started, you can use Simulink and the HDL simulator to monitor simulation progress and results. For example, you might add signals to an HDL simulator Wave window to monitor simulation timing diagrams.

Using the Block Parameters dialog for an HDL Cosimulation block, you can configure the following:

- Block input and output ports that correspond to signals (including internal signals) of an HDL module. You can specify sample times and fixed-point data types for individual block output ports if desired.
- Type of communication and communication settings used for exchanging data between the simulation tools.

- Rising-edge or falling-edge clocks to apply to your module. The period of each clock is individually specifiable.
- Tcl commands to run before and after the simulation.

EDA Simulator Link MQ software equips the HDL simulator with a set of customized command extensions. Using the supplied command extension `vsimulink`, you execute the HDL simulator with an instance of an HDL module for cosimulation with Simulink. After the module is loaded, you can start the cosimulation session from Simulink.

EDA Simulator Link MQ software also includes a block for generating value change dump (VCD) files. You can use VCD files generated with this block to perform the following tasks:

- View Simulink simulation waveforms in your HDL simulation environment
- Compare results of multiple simulation runs, using the same or different simulation environments
- Use as input to post-simulation analysis tools

## **Communicating with MATLAB or Simulink and the HDL Simulator**

The mode of communication that you use for a link between the HDL simulator and MATLAB or Simulink depends on whether your simulation application runs in a local, single-system configuration or in a network configuration. If the HDL simulator and The MathWorks products can run locally on the same system and your application requires only one communication channel, you have the option of choosing between shared memory and TCP/IP socket communication. Shared memory communication provides optimal performance and is the default mode of communication.

TCP/IP socket mode is more versatile. You can use it for single-system and network configurations. This option offers the greatest scalability. For more on TCP/IP socket communication, see Appendix D, “TCP/IP Socket Communication”.

## Requirements

In this section...
“What You Need to Know” on page 1-9
“Required Products” on page 1-9

### What You Need to Know

The documentation provided with the EDA Simulator Link™ MQ software assumes users have a moderate level of prerequisite knowledge in the following subject areas:

- Hardware design and system integration
- VHDL and/or Verilog
- ModelSim® simulators
- MATLAB™

Experience with Simulink and Simulink Fixed Point software is required for applying the Simulink component of the product.

Depending on your application, experience with the following MATLAB toolboxes and Simulink blocksets might also be useful:

- Signal Processing Toolbox™
- Filter Design Toolbox™
- Communications Toolbox™
- Signal Processing Blockset™
- Communications Blockset™
- Video and Image Processing Blockset™

### Required Products

EDA Simulator Link MQ software requires the following:

**Platform**

Visit the EDA Simulator Link MQ requirements page on The MathWorks Web site for specific platforms supported with the current release.

**Application software**

Requires ModelSim SE/PE, a Mentor Graphics product.

Visit the EDA Simulator Link MQ requirements page on The MathWorks Web site for specific versions supported with the current release.

**Application software  
required for cosimulation**

MATLAB

Simulink

Simulink Fixed Point

Fixed-Point Toolbox

**Optional application software**

Communications Blockset  
Signal Processing Blockset  
Filter Design Toolbox  
Signal Processing Toolbox  
Video and Image Processing Blockset

---

**Note** Many EDA Simulator Link MQ demos require one or more of the optional products listed.

---

**Platform-specific software**

On the Linux platform, the gcc c++ libraries (3.2 or later) are required by the EDA Simulator Link MQ software. You should install a recent version of the gcc c++ library on your computer. To determine which libraries are installed on your computer, type the command:

```
gcc -v
```

# Setting Up Your Environment for the EDA Simulator Link™ MQ Software

## In this section...

“Installing the Link Software” on page 1-12

“Installing Related Application Software” on page 1-12

“Setting Up the HDL Simulator for Use with the Link Software” on page 1-12

“Using the EDA Simulator Link™ MQ Libraries” on page 1-18

## Installing the Link Software

For details on how to install the EDA Simulator Link™ MQ software, see the MATLAB™ installation instructions.

## Installing Related Application Software

Based on your configuration decisions and the software required for your EDA Simulator Link MQ application, identify software you need to install and where you need to install it. For example, if you need to run multiple instances of the link MATLAB server on different machines, you need to install MATLAB and any applicable toolbox software on multiple systems. Each instance of MATLAB can run only one instance of the server.

For details on how to install the HDL simulator, see the installation instructions for that product. For information on installing MathWorks™ products, see the MATLAB installation instructions.

## Setting Up the HDL Simulator for Use with the Link Software

EDA Simulator Link MQ software provides a guided set-up script (syscheckmq) for configuring your simulator. This script works whether you have installed the link software and MATLAB on the same machine as the HDL simulator or installed them on different machines.



The set-up script creates a configuration file containing the location of the appropriate EDA Simulator Link MQ MATLAB and Simulink libraries. You can then include this configuration with any other calls you make using ModelSim SE/PE `vsim` from the HDL simulator. You only need to run this script once.

---

**Note** The EDA Simulator Link MQ configuration and diagnostic script works only on UNIX and Linux. Windows users: please see instructions below.

---

If you plan to use the MATLAB `vsim.m` function instead, no setup is required.

Refer to “Using the EDA Simulator Link™ MQ Libraries” on page 1-18 for the correct link application library for your platform. Then see “Starting ModelSim SE/PE from MATLAB” on page 1-23.

After you have created your configuration files, see “Starting the ModelSim Software from a Shell” on page 1-25.

### Using the Configuration and Diagnostic Script for UNIX/Linux

`syscheckmq` provides an easy way to configure your simulator setup to work with the EDA Simulator Link MQ software.

The following is an example of running `syscheckmq` on a Linux 64 machine with EDA Simulator Link MQ libraries in a different location than where they were first installed and specifying a TCP/IP connection.

Start `syscheckmq`:

```
% syscheckmq
*****

Kernel name: Linux
Kernel release: 2.6.11.4-20a-smp
Machine: x86_64
*****
```

The script first returns the location of the HDL simulator installation (vsim.exe). If it does not find an installation, you receive an error message. Either provide the path to the installation or quit the script and install ModelSim SE/PE. You are then prompted to accept this installation or provide a path to another one, after which you receive a message confirming the HDL simulator installation:

```
Found /hub/share/apps/ModelSim/v60e/se/modeltech/bin/vsim on the path.
Press Enter to use the path we found or enter another one:
```

```
*****
/hub/share/apps/ModelSim/v60e/se/modeltech/bin/vsim -version
Model Technology ModelSim SE-64 vsim 6.0e Simulator 2005.06 Jun 17 2005
ModelSim mode: 32 bits
*****
```

Next, the script needs to know if the EDA Simulator Link MQ libraries are in the default directory (where they were first installed) or if you have moved them to another directory. If you have the HDL simulator and MATLAB on separate machines, move the link libraries to the ModelSim SE/PE machine.

```
Select method to search for EDA Simulator Link MQ libraries:
1. Use libraries installed with EDA Simulator Link MQ.
2. Prompt me to specify the direct path to the libraries.
2
Enter the path to liblfmhd1c_gcc41.so and liblfmhd1s_gcc41.so:
/tmp/lfmconfig/linux64
Found /tmp/lfmconfig/linux64/liblfmhd1c_gcc41.so
and /tmp/lfmconfig/linux64/liblfmhd1s_gcc41.so.
```

The script then runs a dependency checker to check for supporting libraries. If any of the libraries cannot be found, you probably need to append your environment path to find them.

```
*****
Running dependency checker "ldd /tmp/lfmconfig/linux64/liblfmhd1c_gcc41.so".
Dependency checker passed.
Dependency status:
```

```

librt.so.1 => /lib64/tls/librt.so.1 (0x00002aaaaac0a000)
libpthread.so.0 => /lib64/tls/libpthread.so.0 (0x00002aaaaad12000)
libstdc++.so.6 => /devel/Ahdl/nightly/matlab/sys/os/glnxa64/libstdc++.so
.6 (0x00002aaaaae27000)
libm.so.6 => /lib64/tls/libm.so.6 (0x00002aaaaab029000)
libgcc_s.so.1 => /devel/Ahdl/nightly/matlab/sys/os/glnxa64/libgcc_s.so.1
(0x00002aaaaab180000)
libc.so.6 => /lib64/tls/libc.so.6 (0x00002aaaaab28d000)
/lib64/ld-linux-x86-64.so.2 (0x0000555555555000)
*****

```

This next step loads the EDA Simulator Link MQ libraries and compiles a test module to verify the libraries loaded correctly.

Press Enter to load EDA Simulator Link MQ or enter 'n' to skip this test:

```

Reading /mathworks/hub/share/apps/ModelSim/v60e/se/modeltech/linux_x86_64/./modelsim.ini
"worklfx24154" maps to directory worklfx24154. (Default mapping)
Model Technology ModelSim SE-64 vlog 6.0e Compiler 2005.06 Jun 17 2005
-- Compiling module d24154

```

Top level modules:

```

d24154

```

```

*****

```

```

Reading /mathworks/hub/share/apps/ModelSim/v60e/se/modeltech/tcl/vsim/pref.tcl

```

```

# 6.0e

```

```

# vsim -do exit -foreign {matlabclient /tmp/lfmconfig/linux64/liblhmhldc_gcc41.so}
-c worklfx24154.d24154

```

```

# // ModelSim SE-64 6.0e Jun 17 2005 Linux 2.6.11.4-20a-smp

```

```

# //

```

```

# // Copyright Mentor Graphics Corporation 2005

```

```

# // All Rights Reserved.

```

```

# //

```

```

# // THIS WORK CONTAINS TRADE SECRET AND

```

```

# // PROPRIETARY INFORMATION WHICH IS THE PROPERTY

```

```

# // OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS

```

```
# // AND IS SUBJECT TO LICENSE TERMS.
# //
# Loading worklfx24154.d24154
# Loading /tmp/lfmconfig/linux64/liblfmhdlc_gcc41.so
# exit

*****

EDA Simulator Link MQ libraries loaded successfully.
*****
```

Next, the script checks a TCP connection. If you choose to skip this step, the configuration file specifies use of shared memory. Both shared memory and socket configurations are in the configuration file; depending on your choice, one configuration or the other is commented out.

Press Enter to check for TCP connection or enter 'n' to skip this test:

Enter an available port [5001]

Enter remote host [localhost]

Press Enter to continue

```
ttcp_glnx -t -p5001 localhost
Connection successful
```

Lastly, the script creates the configuration file, unless for some reason you choose not to do so at this time.

```
*****

Press Enter to Create Configuration files or 'n' to skip this step:

*****

Created template files simulink24255.arg and matlab24255.arg. Inspect and modify
if necessary.
*****
```

Diagnosis Completed

The template file names, in this example `simulink24255.arg` and `matlab24255.arg`, have different names each time you run this script.

After the script is complete, you can leave the configuration files where they are or move them to wherever it is convenient.

## Using the Configuration and Diagnostic Script with Windows

`syscheckmq` does not run on Windows. To use the configuration script on Windows, create two files according to the following instructions:

- 1 Create a MATLAB configuration file and name it. There are no file-naming restrictions. Enter the following text:

```
//Command file for MATLAB EDA Simulator Link MQ.
//Loading of foreign Library, usage example: vsim -f matlab14455.arg entity.
//You can manually change the following line to point to the correct library.
//The default location of the 32-bit Windows library is at
//MATLABROOT/toolbox/modelsim/windows32/liblfmhdc_vs05.dll.

-foreign "matlabclient c:/path/liblfmhdc_vs05.dll"
```

where *path* is the path to the particular EDA Simulator Link MQ shared library you want to invoke (in this example. See “Using the EDA Simulator Link™ MQ Libraries” on page 1-18).

For more information on the `-foreign` option, refer to your ModelSim documentation.

The comments in the above text are optional.

- 2 Create a Simulink configuration file and name it. There are no file-naming restrictions. Enter the following text:

```
//Command file for Simulink EDA Simulator Link MQ.
//Loading of foreign Library, usage example: vsim -f simulink14455.arg entity.
//You can manually change the following line to point to the correct library.
//For example the default location of the 32-bit Windows library is at
//MATLABROOT/toolbox/modelsim/windows32/liblfmhdc_vs05.so.
```

```
//For socket connection uncomment and modify the following line:  
-foreign "simlinkserver c:/path/liblfmhds_vs05.so ; -socket 5001"  
  
//For shared connection uncomment and modify the following line:  
//-foreign "simlinkserver c:/path/liblfmhds_vs05.so"
```

Where *path* is the path to the particular EDA Simulator Link MQ shared library you want to invoke. See “Using the EDA Simulator Link™ MQ Libraries” on page 1-18.

---

**Note** If you are going to use a TCP/IP socket connection, first confirm that you have an available port to put in this configuration file. Then, comment out whichever type of communication you will not be using.

---

The comments in the above text are optional.

After you have finished creating the configuration files, you can leave the files where they are or move them to another location that is convenient.

## Using the EDA Simulator Link™ MQ Libraries

In general, you want to use the same compiler for all libraries linked into the same executable. The link software provides many versions of the same library compilers that are available with the HDL simulators (usually some version of GCC). Using the same libraries ensures compatibility with other C++ libraries that may get linked into the HDL simulator, including SystemC libraries.

If you have any of these conditions, choose the version of the EDA Simulator Link MQ library that matches the compiler used for that code:

- Link other third-party applications into your HDL simulator.
- Compile and link in SystemC code as part of your design or testbench.
- Write custom C/C++ applications and link them into your HDL simulator.

If you do not link any other code into your HDL simulator, you can use any version of the supplied libraries. A default library version is understood by the `vsim` MATLAB command.

---

**Note** EDA Simulator Link MQ software supports running in 32-bit mode on a 64-bit Solaris machine, but it does not support running on a 32-bit Solaris platform.

---

## Library Names

The EDA Simulator Link MQ libraries are named according to the following format:

```
productdir/arch/lib{product_short_name}{client_server_tag}_{compiler_tag}.{libext}
```

where

<code>productdir</code>	<code>modelsim</code>
<code>arch</code>	<code>linux32, linux64, solaris32, solaris64, or windows32</code>
<code>product_short_name</code>	<code>lfm</code>
<code>client_server_tag</code>	<code>c or s (MATLAB or Simulink)</code>
<code>compiler_tag</code>	<code>gcc32, gcc33, gcc40, gcc41, spro11, or vs05</code>
<code>libext</code>	<code>dll or so</code>

Not all combinations are supported. See “Default Libraries” on page 1-19 for valid combinations.

## Default Libraries

EDA Simulator Link MQ scripts fully support the use of designated default libraries.

With the EDA Simulator Link MQ software, the default library for each platform is the version compiled using the same compiler that The MathWorks

uses to compile MATLAB and Simulink. The following table lists all the libraries shipped with the link software. The default libraries for each platform are in bold text.

Platform	MATLAB Library	Simulink Library
Linux32, Linux64	<b>liblfmhdlc_gcc41.so</b> liblfmhdlc_gcc32.so liblfmhdlc_gcc40.so	<b>liblfmhdls_gcc41.so</b> liblfmhdls_gcc32.so liblfmhdls_gcc40.so
Solaris32, Solaris64	<b>liblfmhdlc_spro11.so</b> liblfmhdlc_gcc33.so	<b>liblfmhdls_spro11.so</b> liblfmhdls_gcc33.so
Windows32	<b>liblfmhdlc_vs05.dll</b> liblfmhdlc_gcc32.dll liblfmhdlc_gcc33.dll	<b>liblfmhdls_vs05.dll</b> liblfmhdls_gcc32.dll liblfmhdls_gcc33.dll

### Using an Alternative Library

You can use a different HDL-side library by specifying it explicitly using the `libfile` parameter to the `vsim` MATLAB command. You should choose the version of the library that matches the compiler and system libraries you are using for any other C/C++ libraries linked into the HDL simulator. Depending on the version of your HDL simulator, you may need to explicitly set additional paths in the `LD_LIBRARY_PATH` environment variable.

Depending on the version of your HDL simulator, you may need to explicitly set additional paths in the `LD_LIBRARY_PATH` environment variable. For example, if you want to use a nondefault library:

- 1 Copy the system libraries from the MATLAB installation (found in `matlabroot/sys/os/platform`) to the machine with the HDL simulator (where `matlabroot` is your MATLAB installation and `platform` is one of the above architecture, e.g., `linux32`).
- 2 Modify the `LD_LIBRARY_PATH` environment variable to add the path to the system libraries that were copied in step 1.



**Example: EDA Simulator Link MQ Alternative Library Using vsim.** In this example, you run the 32-bit Solaris version of ModelSim 6 software on the same 64-bit Solaris machine which is running MATLAB. Because you want to incorporate some SystemC designs, you are using the EDA Simulator Link MQ version compiled with GCC 3.3. You can download the appropriate version of GCC with its associated system libraries from Mentor Graphics, instead of using the default library version compiled with SunPro 11.

In MATLAB:

```
>> currPath = getenv('PATH');
>> currLdPath = getenv('LD_LIBRARY_PATH');
>> setenv('PATH', ['/tools/modelsim-6.1e/bin:' currPath]);
>> setenv('LD_LIBRARY_PATH', ['/tools/mtigcc/gcc-3.3-sunos510/lib:' currLdPath]);
>> setenv('MTI_VCO_MODE', '32');
>> vsim('tclstart', { 'vlib work', 'vcom inverter.vhd', 'vsimulink inverter' }, ...
    'libfile', 'liblfmhds_gcc33');
```

You change the *PATH* to ensure that you get the correct version of the ModelSim software. You change the *LD\_LIBRARY\_PATH* because the HDL simulator does not automatically add the necessary path to the system libraries. The EDA Simulator Link MQ function *vsim* automatically detects the use of the 32-bit version of the HDL simulator and uses the solaris32 library directory in the link software installation; there is no need to specify the *libdir* parameter in this case.

The library resolution can be verified using *ldd* from within the ModelSim GUI:

```
vsim> exec ldd /path/to/liblfmhds_gcc33.so
#      librt.so.1 =>      /lib/librt.so.1
#      libpthread.so.1 => /lib/libpthread.so.1
#      libstdc++.so.5 =>  /tools/mtigcc/gcc-3.3-sunos510/lib/libstdc++.so.5
#      libm.so.2 =>      /lib/libm.so.2
#      libgcc_s.so.1 =>   /tools/mtigcc/gcc-3.3-sunos510/lib/libgcc_s.so.1
#      libaio.so.1 =>    /lib/libaio.so.1
#      libmd5.so.1 =>    /lib/libmd5.so.1
#      libc.so.1 =>      /lib/libc.so.1
#
#      /platform/SUNW,Sun-Blade-1000/lib/libmd5_psr.so.1
#      /platform/SUNW,Sun-Blade-1000/lib/libc_psr.so.1
```

**Example: EDA Simulator Link MQ Alternate Library Using System Shell.**

This example shows how to load a ModelSim session by explicitly specifying the EDA Simulator Link MQ library (either the default or one of the alternatives). By explicitly using a system shell, you can execute this example on the same machine as MATLAB, on a different machine, and even on a machine with a different operating system.

In this example, you are running the 64-bit Linux version of QuestaSim 6.2c. It does not matter which machine is running MATLAB. Instead of using the EDA Simulator Link MQ default library version compiled with GCC 4.1.1, you are using the version compiled with GCC 4.0.2. You can download the appropriate version of GCC with its associated system libraries from Mentor Graphics.

In a csh-compatible system shell:

```
csh> setenv PATH /tools/modelsim-6.2c/bin:${PATH}
csh> setenv LD_LIBRARY_PATH /tools/mtigcc/gcc-4.0.2-linux_x86_64/lib64:${LD_LIBRARY_PATH}
csh> setenv MTI_VCO_MODE 64
csh> vlib work
csh> vcom +acc+inverter inverter.vhd
csh> vsim +acc+inverter -foreign
      "matlabclient /tools/matlab-7b/toolbox/modelsim/linux64/liblfmhdc_gcc40.so" inverter.vhd
```

You change the *PATH* to ensure that you get the correct version of the ModelSim software. You change the *LD\_LIBRARY\_PATH* because the HDL simulator does not automatically add the necessary path to the system libraries unless you are working with 6.2+ and have placed GCC at the root of the ModelSim installation.

You can check the proper library resolution using `ldd` as in the previous example.

## Starting the HDL Simulator

### In this section...

“Starting ModelSim SE/PE from MATLAB” on page 1-23

“Starting the ModelSim Software from a Shell” on page 1-25

### Starting ModelSim SE/PE from MATLAB

Start ModelSim® SE/PE directly from MATLAB® or Simulink® by calling the MATLAB function `vsim`. This function starts and configures the HDL simulator for use with the EDA Simulator Link™ MQ software. By default, the function starts the first version of the simulator executable (`vsim.exe`) that it finds on the system path (defined by the `path` variable), using a temporary DO file that is overwritten each time the HDL simulator starts.

To start ModelSim SE/PE from MATLAB, enter `vsim` at the MATLAB command prompt:

```
>> vsim
```

You can customize the DO startup file and communication mode to be used between MATLAB or Simulink and the HDL simulator by specifying the call to `vsim` with property name/property value pairs. Refer to `vsim` reference documentation for specific information regarding the property name/property value pairs.

See “`vsim` Examples” on page 1-23 for examples of using `vsim` with various property/name value pairs and other parameters.

When you specify a communication mode using `vsim`, the function applies the specified communication mode to all MATLAB or Simulink/ModelSim SE/PE sessions.

### `vsim` Examples

The following example changes the directory location to `VHDLproj` and then calls the function `vsim`. Because the command line omits the `'vsimdir'` and `'startupfile'` properties, `vsim` creates a temporary DO file. The `'tclstart'`

property specifies Tcl commands that load and initialize the HDL simulator for test bench instance modsimrand.

```
cd VHDLproj
vsim('tclstart',...
     'vsimmatlab modsimrand; matlabbtb modsimrand 10 ns -socket 4449')
```

The following example changes the directory location to VHDLproj and then calls the function vsim. Because the function call omits the 'vsimdir' and 'startupfile' properties, vsim creates a temporary DO file. The 'tclstart' property specifies a Tcl command that loads the VHDL entity parse in library work for cosimulation between vsim and Simulink. The 'socketsimulink' property specifies TCP/IP socket communication on the same computer, using socket port 4449.

```
cd VHDLproj
vsim('tclstart', 'vsimulink work.parse', 'socketsimulink', '4449')
```

The following example has the HDL compilation and simulation commands run automatically when you start the ModelSim software from MATLAB.

```
vsim('tclstart',
     {'vlib work', 'vlog +acc clocked_inverter.v hdl_top.v', 'vsim +acc hdl_top' });
```

This next example loads the HDL simulation just as in the previous example but it also loads in the Link to Simulink library, uses socket number 5678 to communicate with cosimulation blocks in Simulink models, and uses an HDL time precision of 10 ps.

```
vsim('tclstart',
     {'vlib work', 'vlog -novopt clocked_inverter.v hdl_top.v',
      'vsimulink hdl_top -socket 5678 -t 10ps'});
```

Or

```
vsim('tclstart',
     {'vlib work', 'vlog -novopt clocked_inverter.v hdl_top.v',
      'vsimulink hdl_top -t 10ps'},
     'socketsimulink', 5678);
```

## Starting the ModelSim Software from a Shell

To start the HDL simulator from a shell and include the EDA Simulator Link MQ libraries, you need to first run the configuration script. See “Setting Up the HDL Simulator for Use with the Link Software” on page 1-12.

After you have the configuration files, you can start the ModelSim software from the shell by typing:

```
% vsim -f matlabconfigfile modelname
```

`matlabconfigfile` should be the name of the MATLAB configuration file you created either with the guided script (Linux/UNIX) or by creating the file yourself (Windows). If you are connecting to Simulink, this should be the name of the Simulink configuration file. Either way, you must also specify the path to the configuration file if it does not reside in the same directory as `vsim.exe`.

The configuration file mainly defines the `-foreign` option to `vsim` which in turn loads the EDA Simulator Link MQ shared library and specifies its entry point.

You can also specify any other existing configuration files you may also be using with this call.

If you are performing this step manually, the following use of `-foreign` with `vsim` loads the EDA Simulator Link MQ client shared library and specifies its entry point:

```
% vsim -foreign matlabclient /path/library
```

where `path` is the path to this particular EDA Simulator Link MQ library. See “Using the EDA Simulator Link™ MQ Libraries” on page 1-18 to find the correct library name for your machine.

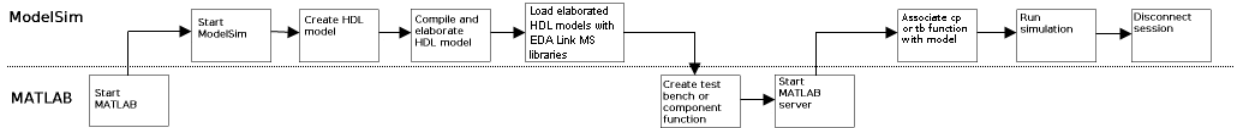
---

**Note** You can also issue this exact same command from inside the HDL simulator.

---

## Workflow for Using the EDA Simulator Link™ MQ Software with MATLAB® Software

The following diagram illustrates the steps necessary to create and run a MATLAB® test bench or component session.



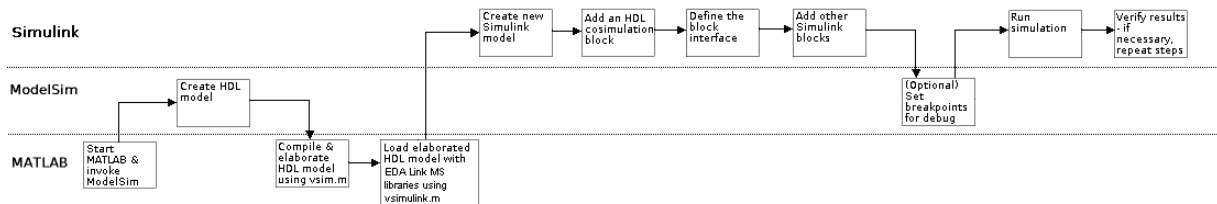
This diagram assumes you have used the configuration and diagnostic script (either UNIX/Linux or Windows) and are starting ModelSim SE/PE outside of MATLAB software.

---

**Note** This workflow is a recommendation only. You might create, compile, and elaborate your HDL module differently than illustrated before starting MATLAB software. The preceding illustration is simply one possible workflow.

---

## Workflow for Using the EDA Simulator Link™ MQ Software with Simulink® Software



This diagram assumes you are using the configuration and diagnostic script (either UNIX/Linux or Windows) and starting ModelSim SE/PE outside of MATLAB software.

**Note** This workflow is a recommendation only. You might create, compile, and elaborate your HDL module differently than illustrated before starting MATLAB and Simulink software. The preceding illustration is simply one possible workflow.

## Learning More About the EDA Simulator Link™ MQ Software

<b>In this section...</b>
“Documentation Overview” on page 1-28
“Online Help” on page 1-29
“Demos and Tutorials” on page 1-30

### Documentation Overview

The following documentation is available with this product.

Chapter 1, “Getting Started”	Explains what the product is, the steps for installing and setting it up, how you might apply it to the hardware design process, and how to gain access to product documentation and online help. Directs you to product demos and tutorials.
Chapter 2, “Linking MATLAB® to ModelSim® Simulators”	Explains how to code HDL models and MATLAB® functions for EDA Simulator Link™ MQ MATLAB applications. Provides details on how the link interface maps HDL data types to MATLAB data types and vice versa. Explains how to start and control HDL simulator and MATLAB test bench and component sessions.
Chapter 3, “Linking Simulink® to ModelSim® Simulators”	Explains how to use the HDL simulator and Simulink for cosimulation modeling.
Chapter 4, “EDA Simulator Link™ MQ MATLAB® Function Reference”	Describes EDA Simulator Link MQ functions for use with MATLAB.



Chapter 5, “EDA Simulator Link™ MQ Command Extensions for the HDL Simulator Reference”	Describes EDA Simulator Link MQ commands for use with the HDL simulator.
Chapter 6, “EDA Simulator Link™ MQ Simulink® Block Reference”	Describes EDA Simulator Link MQ blocks for use with Simulink.
Appendix B, “ADMS Support”	Describes cosimulation support for digital blocks within the ADvanceMS (ADMS) environment.
Appendix C, “EDA Simulator Link™ MQ Machine Configuration Requirements”	Explains the machine configurations permissible when linking the HDL simulator to MATLAB or Simulink
Appendix D, “TCP/IP Socket Communication”	Provides information for choosing TCP/IP socket ports.
Appendix E, “Race Conditions in HDL Simulators”	Describes ways to avoid race conditions in hardware cosimulations with MATLAB and Simulink

## Online Help

The following online help is available:

- Online help in the MATLAB Help browser. Click the EDA Simulator Link MQ product link in the browser’s Contents.
- M-help for EDA Simulator Link MQ MATLAB functions and HDL simulator commands. This help is accessible with the MATLAB doc and help commands. For example, enter the command

```
doc hdldaemon
```

or

```
help hdldaemon
```

at the MATLAB command prompt.

- Block reference pages accessible through the Simulink interface.

## Demos and Tutorials

The EDA Simulator Link MQ software provides demos and tutorials to help you get started.

The demos give you a quick view of the product's capabilities and examples of how you might apply the product. You can run them with limited product exposure. You can find the EDA Simulator Link MQ demos with the online documentation. To access demos, type at the MATLAB command prompt:

```
>> demos
```

Select **Links and Targets > EDA Simulator Link™ MQ** from the navigational pane.

Tutorials provide procedural instruction on how to apply the product. Some focus on features while others focus on application scenarios. The tutorials listed here have a feature focus and each addresses use of the ModelSim software with either MATLAB or Simulink.

- “MATLAB and ModelSim Tutorial” on page 2-56
- “Simulink and ModelSim Tutorial” on page 3-72
- “toVCD Block Tutorial” on page 3-90

# Linking MATLAB<sup>®</sup> to ModelSim<sup>®</sup> Simulators

---

MATLAB<sup>®</sup>-ModelSim<sup>®</sup> Workflow  
(p. 2-2)

Provides a high-level view of the steps involved in coding and running MATLAB functions for use with the EDA Simulator Link<sup>™</sup> MQ interface.

Coding an EDA Simulator Link<sup>™</sup> MQ MATLAB<sup>®</sup> Application (p. 2-4)

Explains how to code HDL modules and MATLAB functions for use with EDA Simulator Link MQ software. Provides details on how the EDA Simulator Link MQ interface maps HDL data types to MATLAB data types and vice versa.

Associating a MATLAB<sup>®</sup> Link Function with an HDL Module  
(p. 2-37)

Describes scheduling and communications options for a MATLAB link session with ModelSim SE/PE.

Running MATLAB<sup>®</sup> Link Sessions  
(p. 2-48)

Explains how to start and control ModelSim SE/PE and MATLAB link sessions.

MATLAB and ModelSim Tutorial  
(p. 2-56)

Guides you through the process of setting up and running a sample ModelSim and MATLAB test bench session.

## MATLAB®-ModelSim® Workflow

The following table lists the steps necessary to create and run a MATLAB® test bench or component session.

In MATLAB...	In ModelSim SE/PE...
<b>1</b> Start the MATLAB application and invoke the ModelSim® simulator (see “Starting the HDL Simulator” on page 1-23)	
	<b>2</b> Create the HDL model. <b>3</b> Compile and elaborate the HDL model. <b>4</b> Load elaborated HDL model with EDA Simulator Link™ MQ libraries. See “Loading an HDL Design for Verification” on page 2-12.

In MATLAB...	In ModelSim SE/PE...
<p><b>5</b> Create test bench or component function (see “Coding an EDA Simulator Link™ MQ MATLAB® Application” on page 2-4).</p> <p><b>6</b> Start the server. See “Starting the MATLAB Server” on page 2-49.</p>	
	<p><b>7</b> Use <code>matlabcp</code>, <code>matlabtb</code>, or <code>matlabtbeval</code> to associate the function you wrote in step 5 with a module of the loaded model currently in the HDL simulator (see “Associating the HDL Module Component with the MATLAB Link Function” on page 2-38). For additional scheduling and communication options, see “Scheduling Options for a Link Session” on page 2-40. See also the reference pages for <code>matlabcp</code>, <code>matlabtb</code>, and <code>matlabtbeval</code>.</p> <p><b>8</b> Run the simulation.</p> <p><b>9</b> Disconnect the session by using <code>nomatlabtb</code>.</p>

## Coding an EDA Simulator Link™ MQ MATLAB® Application

### In this section...

“Overview” on page 2-4

“Process for Coding an EDA Simulator Link™ MQ MATLAB Application” on page 2-5

“Coding HDL Modules for MATLAB Verification” on page 2-7

“Coding MATLAB Link Functions” on page 2-12

### Overview

The EDA Simulator Link™ MQ software provides a means for verifying and visualizing ModelSim® HDL modules within the MATLAB® environment. You do this by coding an HDL model and a MATLAB function that can share data with the HDL model. This chapter discusses the programming, interfacing, and scheduling conventions for MATLAB functions that communicate with the HDL simulator.

EDA Simulator Link MQ software supports two types of MATLAB functions that interface to HDL modules:

- *MATLAB test bench* functions let you verify the performance of the HDL model, or of components within the model. A test bench function drives values onto signals connected to input ports of an HDL design under test, and receives signal values from the output ports of the module.

The syntax of a MATLAB test bench function is

```
function [iport, tnext] = MyFunctionName(oport, tnow, portinfo)
```

- *MATLAB component* functions simulate the behavior of components 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.

The syntax of a MATLAB component function is

```
function [oport, tnext] = MyFunctionName(iport, tnow, portinfo)
```

These two types of MATLAB functions are referred to collectively as MATLAB *link functions*, and a test bench or component session may be referred to as a MATLAB *link session*.

The programming, interfacing, and scheduling conventions for test bench functions and MATLAB component functions are almost identical. Most of this chapter focuses on test bench functions, but in general all operations can be performed on and with both link functions. The test bench section is followed by a discussion of MATLAB component functions and how to use them.

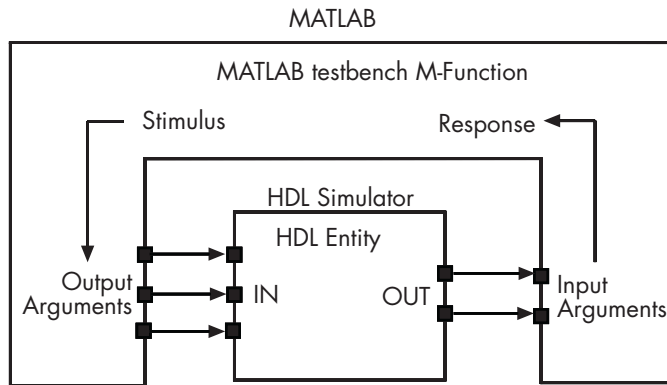
Refer to Appendix C, “EDA Simulator Link™ MQ Machine Configuration Requirements” for valid machine configurations.

## **Process for Coding an EDA Simulator Link™ MQ MATLAB Application**

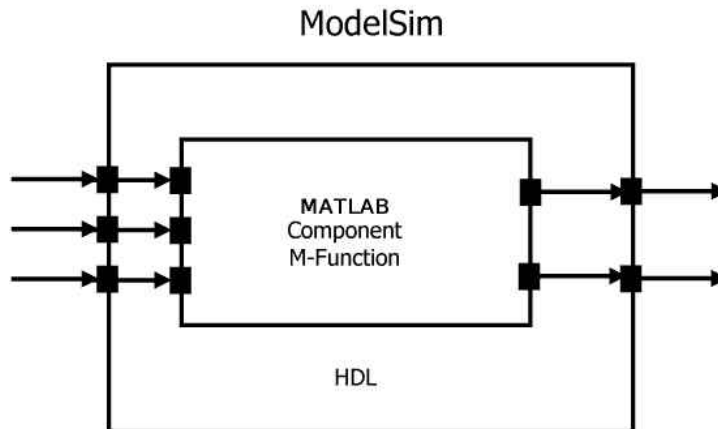
This section provides an overview of the steps required to develop an HDL model for use with MATLAB and the EDA Simulator Link MQ software. To program the HDL component of an EDA Simulator Link MQ application,

- 1** Code the HDL model for MATLAB verification (see “Coding HDL Modules for MATLAB Verification” on page 2-7).
- 2** Compile the HDL model (see “Compiling and Debugging the HDL Model” on page 2-11).
- 3** Code the required MATLAB test bench or MATLAB component functions (see “Coding MATLAB Link Functions” on page 2-12).
- 4** Place the MATLAB functions on the MATLAB search path

The following figure shows how a MATLAB function wraps around and communicates with the HDL simulator during a test bench simulation session.

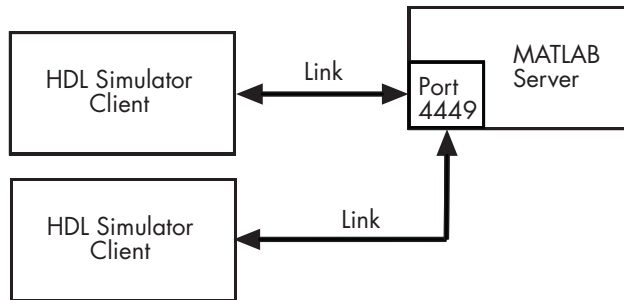


The following figure shows how a MATLAB component function is wrapped around by and communicates with the HDL simulator during a component simulation session.



When linked with MATLAB, the HDL simulator functions as the client, MATLAB as the server. The following figure shows a multiple-client scenario connecting to the server at TCP/IP socket port 4449.





The MATLAB server can service multiple simultaneous HDL simulator sessions and HDL modules. However, you should adhere to recommended guidelines to ensure the server can track the I/O associated with each module and session. The MATLAB server, which you start with the supplied MATLAB function `hdldaemon`, waits for connection requests from instances of ModelSim SE/PE running on the same or different computers. When the server receives a request, it executes the specified MATLAB function you have coded to perform tasks on behalf of a module in your HDL design. Parameters that you specify when you start the server indicate whether the server establishes shared memory or TCP/IP socket communication links.

Refer to Appendix C, “EDA Simulator Link™ MQ Machine Configuration Requirements” for valid machine configurations.

## Coding HDL Modules for MATLAB Verification

- “Overview” on page 2-8
- “Choosing an HDL Module Name” on page 2-8
- “Specifying Port Direction Modes” on page 2-8
- “Specifying Port Data Types” on page 2-9
- “Sample VHDL Entity Definition” on page 2-10
- “Compiling and Debugging the HDL Model” on page 2-11
- “Loading an HDL Design for Verification” on page 2-12

## Overview

The most basic element of communication in the EDA Simulator Link MQ interface is the HDL module. The interface passes all data between the HDL simulator and MATLAB as port data. The EDA Simulator Link MQ software works with any existing HDL module. However, when coding an HDL module that is targeted for MATLAB verification, you should consider its name, the types of data to be shared between the two environments, and the direction modes. The sections within this chapter cover these topics.

## Choosing an HDL Module Name

Although not required, when naming the HDL module, consider choosing a name that also can be used as a MATLAB function name. (Generally, naming rules for VHDL or Verilog and MATLAB are compatible.) By default, EDA Simulator Link MQ software assumes that an HDL module and its simulation function share the same name. See “Naming a MATLAB Link Function” on page 2-37.

For details on MATLAB function-naming guidelines, see “MATLAB Programming Tips” on files and file names in the MATLAB documentation.

## Specifying Port Direction Modes

In your module statement, you must specify each port with a direction mode (input, output, or bidirectional). The following table defines the three modes:

Use VHDL Mode...	Use Verilog Mode...	For Ports That...
IN	input	Represent signals that can be driven by a MATLAB function
OUT	output	Represent signal values that are passed to a MATLAB function
INOUT	inout	Represent bidirectional signals that can be driven by or pass values to a MATLAB function

## Specifying Port Data Types

This section describes how to specify data types compatible with MATLAB for ports in your HDL modules. For details on how the EDA Simulator Link MQ interface converts data types for the MATLAB environment, see “Performing Data Type Conversions” on page 2-20.

---

**Note** If you use unsupported types, the EDA Simulator Link MQ software issues a warning and ignores the port at run-time. For example, if you define your interface with five ports, one of which is a VHDL access port, at run-time the interface displays a warning and your M-code sees only four ports.

---

**Port Data Types for VHDL Entities.** In your entity statement, you must define each port that you plan to test with MATLAB with a VHDL data type that is supported by the EDA Simulator Link MQ software. The interface can convert scalar and array data of the following VHDL types to comparable MATLAB types:

- STD\_LOGIC, STD\_ULOGIC, BIT, STD\_LOGIC\_VECTOR, STD\_ULOGIC\_VECTOR, and BIT\_VECTOR
- INTEGER and NATURAL
- REAL
- TIME
- Enumerated types, including user-defined enumerated types and CHARACTER

The interface also supports all subtypes and arrays of the preceding types.

---

**Note** The EDA Simulator Link MQ software does not support VHDL extended identifiers for the following components:

- Port and signal names used in cosimulation
- Enum literals when used as array indices of port and signal names used in cosimulation

Basic identifiers for VHDL are supported.

---

**Port Data Types for Verilog Modules.** In your module definition, you must define each port that you plan to test with MATLAB with a Verilog port data type that is supported by the EDA Simulator Link MQ software. The interface can convert data of the following Verilog port types to comparable MATLAB types:

- reg
- integer
- wire

---

**Note** EDA Simulator Link MQ software does not support Verilog escaped identifiers for port and signal names used in cosimulation. Simple identifiers for Verilog are supported.

---

### Sample VHDL Entity Definition

The sample VHDL code fragment below defines the entity decoder. By default, the entity is associated with MATLAB test bench function decoder.

The keyword PORT marks the start of the entity's port clause, which defines two IN ports—`isum` and `qsum`—and three OUT ports—`adj`, `dvalid`, and `odata`. The output ports drive signals to MATLAB function input ports for processing. The input ports receive signals from the MATLAB function output ports.

Both input ports are defined as vectors consisting of five standard logic values. The output port `adj` is also defined as a standard logic vector, but consists of

only two values. The output ports `dvalid` and `odata` are defined as scalar standard logic ports. For information on how the EDA Simulator Link MQ interface converts data of standard logic scalar and array types for use in the MATLAB environment, see “Performing Data Type Conversions” on page 2-20.

```
ENTITY decoder IS
PORT (
    isum    : IN std_logic_vector(4 DOWNTO 0);
    qsum    : IN std_logic_vector(4 DOWNTO 0);
    adj     : OUT std_logic_vector(1 DOWNTO 0);
    dvalid  : OUT std_logic;
    odata   : OUT std_logic);
END decoder ;
```

## Compiling and Debugging the HDL Model

After you create or edit your HDL source files, use the HDL simulator compiler to compile and debug the code.

You have the option of invoking the compiler from menus in the ModelSim SE/PE graphic interface or from the command line with the `vcom` command. The following sequence of ModelSim commands creates and maps the design library work and compiles the VHDL file `modsimrand.vhd`:

```
ModelSim> vlib work
ModelSim> vmap work work
ModelSim> vcom modsimrand.vhd
```

The following sequence of ModelSim commands creates and maps the design library work and compiles the Verilog file `test.v`:

```
ModelSim> vlib work
ModelSim> vmap work work
ModelSim> vlog test.v
```

For more examples, see the EDA Simulator Link MQ tutorials. For details on using the ModelSim compiler, see the ModelSim SE/PE documentation.

### **Loading an HDL Design for Verification**

After you start the HDL simulator from MATLAB with a call to `vsim`, load an instance of an HDL module for verification with the HDL simulator command `vsimmatlab`. At this point, it is assumed that you have coded and compiled your HDL model as explained in “Coding HDL Modules for MATLAB Verification” on page 2-7. Issue the HDL simulator command `vsimmatlab` for each instance of an entity or module in your model that you want to cosimulate. For example:

```
vsimmatlab work.modsimrand
```

This command loads the EDA Simulator Link MQ library, opens a simulation workspace for `modsimrand`, and displays a series of messages in the HDL simulator command window as the simulator loads the entity (see demo for remaining code).

### **Coding MATLAB Link Functions**

- “Process for Coding MATLAB Link Functions” on page 2-12
- “Defining Link Functions and Link Function Parameters” on page 2-14
- “Performing Data Type Conversions” on page 2-20
- “Sample MATLAB Test Bench Function” on page 2-29

### **Process for Coding MATLAB Link Functions**

When coding a MATLAB function that is to verify or visualize an HDL module or component, you must adhere to specific coding conventions, understand the data type conversions that occur, and program data type conversions for operating on data and returning data to the HDL simulator.

To code a MATLAB link function that is to verify or visualize an HDL module or component, perform the following steps:

- 1 Learn the syntax for a MATLAB link function (see “Defining Link Functions and Link Function Parameters” on page 2-14).

- 2** Understand how EDA Simulator Link MQ software converts HDL modules data for use in the MATLAB environment (see “Performing Data Type Conversions” on page 2-20).
- 3** Choose a name for the MATLAB function (see “Choosing an HDL Module Name” on page 2-8).
- 4** Define expected parameters in the function definition line (see “Defining Link Functions and Link Function Parameters” on page 2-14).
- 5** Determine the types of port data being passed into the function (see “Defining Link Functions and Link Function Parameters” on page 2-14).
- 6** Extract and, if appropriate for the simulation, apply information received in the `portinfo` structure (see “Gaining Access to and Applying Port Information” on page 2-18).
- 7** Convert data for manipulation in the MATLAB environment, as necessary (see “Converting HDL Data to Send to MATLAB” on page 2-20 or ).
- 8** Convert data that needs to be returned to the HDL simulator (see “Converting Data for Return to the HDL Simulator” on page 2-26).

---

**Examples in This Chapter Use the Oscfilter Demo** In the Oscillator demo, a VHDL model implements an oscillator (`simple_osc`) whose output is wired to the input of a stub component (`osc_filter`). The sole purpose of `osc_filter` is to invoke a MATLAB component function (`oscfilter`). The `osc_filter` component passes its input signal into the MATLAB function and receives signal data returned by the function. The `oscfilter` function implements a smoothing filter that filters the signal at the model's base rate and at two oversampling (4x and 8x) rates.

You may find it helpful to refer to the demo files while reading this discussion. The directory `matlabroot\toolbox\modelsim\modelsimdemos\vhdl\osc` contains the VHDL source code files:

- `simple_osc.vhd`: Contains entity and architecture definitions for oscillator component.
- `osc_filter.vhd`: Contains entity and architecture (stub) definitions for filter component.
- `osc_top.vhd`: Top-level VHDL behavioral model; instantiates and connects oscillator and filter components.

This directory also contains the demo M-files:

- `modsimosc.m`: Top-level demo script; starts up `hdldaemon` and the HDL simulator, passing in a cell array of Tel commands to `vsim`. The Tel commands direct compilation and simulation of the model
- `oscfilter.m`: MATLAB component function, called from the HDL simulator; performs filter computations.

---

### Defining Link Functions and Link Function Parameters

The syntax of a MATLAB component function is

```
function [oport, tnext] = MyFunctionName(iport, tnow, portinfo)
```

The syntax of a MATLAB test bench function is



```
function [iport, tnext] = MyFunctionName(oport, tnow, portinfo)
```

Note that the input/output arguments (*iport* and *oport*) for a MATLAB component function are the reverse of the port arguments for a MATLAB test bench function. That is, the MATLAB component function returns signal data to the *outputs* and receives data from the *inputs* of the associated HDL module.

Initialize the function outputs to empty values at the beginning of the function as in the following example:

```
tnext = [];
oport = struct();
```

For more information on using *tnext* and *tnow* for simulation scheduling, see “Scheduling Options for a Link Session” on page 2-40.

The following table describes each of the link function parameters and the roles they play in each of the functions.

Parameter	Test Bench	Component
<i>iport</i>	<i>Output</i> Structure that forces (by deposit) values onto signals connected to input ports of the associated HDL module.	<i>Input</i> Structure that receives signal values from the input ports defined for the associated HDL module at the time specified by <i>tnow</i>
<i>tnext</i>	<i>Output, optional</i> Specifies the time at which the HDL simulator schedules the next callback to MATLAB. <i>tnext</i> should be initialized to an empty value ( <code>[]</code> ). If <i>tnext</i> is not later updated, no new entries are added to the simulation schedule.	<i>Output, optional</i> Same as test bench

<b>Parameter</b>	<b>Test Bench</b>	<b>Component</b>
oport	<i>Input</i> Structure that receives signal values from the output ports defined for the associated HDL module at the time specified by tnow	<i>Output</i> Structure that forces (by deposit) values onto signals connected to output ports of the associated HDL module.
tnow	<i>Input</i> Receives the simulation time at which the MATLAB function is called. By default, time is represented in seconds. For more information see “Scheduling Options for a Link Session” on page 2-40.	Same as test bench
portinfo	<i>Input</i> For the first call to the function only (at the start of the simulation), portinfo receives a structure whose fields describe 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. You can use the port information to create a generic MATLAB function that operates differently depending on the port information supplied at startup. For more information on port data, see “Gaining Access to and Applying Port Information” on page 2-18.	Same as test bench

---

**Note** When importing VHDL signals, signal names in `iport`, `oport`, and `portinfo` are returned in all capitals.

---

**Oscfilter Function Example.** The following code is the function definition of the `oscfilter` MATLAB component function.

```
function [oport,tnext] = oscfilter(iport, tnow, portinfo)
```

Note that the function name `oscfilter`, differs from the entity name `u_osc_filter`. Therefore, the component function name must be passed in explicitly to the `matlabcp` command that connects the function to the associated HDL instance using the `-mfunc` parameter.

The function definition specifies all required input and output parameters, as listed below.

<code>oport</code>	Forces (by deposit) values onto the signals connected to the entity's output ports, <code>filter1x_out</code> , <code>filter4x_out</code> and <code>filter8x_out</code> .
<code>tnext</code>	Specifies a time value that indicates when the HDL simulator will execute the next callback to the MATLAB function.
<code>iport</code>	Receives HDL signal values from the entity's input port, <code>osc_in</code> .
<code>tnow</code>	Receives the current simulation time.
<code>portinfo</code>	For the first call to the function, receives a structure that describes the ports defined for the entity.

The following figure shows the relationship between the HDL entity's ports and the MATLAB function's `iport` and `oport` parameters.



**Gaining Access to and Applying Port Information.** EDA Simulator Link MQ software passes information about the entity or module under test in the `portinfo` structure. The `portinfo` structure is passed as the third argument to the function. It is passed only in the first call to your MATLAB function. The information passed in the `portinfo` structure is useful for validating the entity or module under simulation. The information is supplied in three fields, as indicated below. The content of these fields depends on the type of ports defined for the VHDL entity or Verilog module.

```
portinfo.field1.field2.field3
```

The following table lists possible values for each field and identifies the port types for which the values apply.

### HDL Port Information

Field...	Can Contain...	Which...	And Applies to...
<i>field1</i>	in	Indicates the port is an input port	All port types
	out	Indicates the port is an output port	All port types
	inout	Indicates the port is a <b>bidirectional port</b>	All port types
	tyscale	Indicates the simulator resolution limit in seconds as specified in the HDL simulator	All types
<i>field2</i>	<i>portname</i>	Is the name of the port	All port types

**HDL Port Information (Continued)**

<b>Field...</b>	<b>Can Contain...</b>	<b>Which...</b>	<b>And Applies to...</b>
<i>field3</i>	type	Identifies the port type  For VHDL: integer, real, time, or enum  For Verilog: 'verilog_logic' identifies port types reg, wire, integer	All port types
	<i>right (VHDL only)</i>	The VHDL RIGHT attribute	VHDL integer, natural, or positive port types
	<i>left (VHDL only)</i>	The VHDL LEFT attribute	VHDL integer, natural, or positive port types
	size	VHDL: The size of the matrix containing the data  Verilog: The size of the bit vector containing the data	All port types
	label	VHDL: A character literal or label  Verilog: the string '01ZX'	VHDL: Enumerated types, including predefined types BIT, STD_LOGIC, STD_ULOGIC, BIT_VECTOR, and STD_LOGIC_VECTOR  Verilog: All port types

The first call to the MATLAB function has three arguments including the portinfo structure. Checking the number of arguments is one way that you can ensure that portinfo was passed. For example:

```
if(nargin ==3)
    tscale = portinfo.tscale;
end
```

### Performing Data Type Conversions

To successfully use the EDA Simulator Link MQ software with the HDL simulator and MATLAB or Simulink, you need to understand the data type conversions that the EDA Simulator Link MQ software performs to transmit and receive data between HDL modules and the MATLAB environment.

**Converting HDL Data to Send to MATLAB.** If your ModelSim SE/PE application needs to send HDL data to a MATLAB function, it may be necessary for you to first convert the data to a type supported by MATLAB and the EDA Simulator Link MQ software.

To program a MATLAB function for an HDL model, you must understand the type conversions required by your application. You may also need to handle differences between the array indexing conventions used by the HDL you are using and MATLAB (see section below).

The data types of arguments passed in to the function determine the following:

- The types of conversions required before data is manipulated
- The types of conversions required to return data to the HDL simulator

The following table summarizes how the EDA Simulator Link MQ software converts supported VHDL data types to MATLAB types based on whether the type is scalar or array.

### VHDL-to-MATLAB Data Type Conversions

VHDL Types...	As Scalar Converts to...	As Array Converts to...
STD_LOGIC, STD_ULOGIC, and BIT	A character that matches the character literal for the desired logic state.	
STD_LOGIC_VECTOR, STD_ULOGIC_VECTOR, BIT_VECTOR, SIGNED, and UNSIGNED		A column vector of characters (as defined in VHDL Conversions for the HDL Simulator on page 2-26) with one bit per character.

**VHDL-to-MATLAB Data Type Conversions (Continued)**

<b>VHDL Types...</b>	<b>As Scalar Converts to...</b>	<b>As Array Converts to...</b>
Arrays of STD_LOGIC_VECTOR, STD_ULOGIC_VECTOR, BIT_VECTOR, SIGNED, and UNSIGNED		An array of characters (as defined above) with a size that is equivalent to the VHDL port size.
INTEGER and NATURAL	Type int32.	Arrays of type int32 with a size that is equivalent to the VHDL port size.
REAL	Type double.	Arrays of type double with a size that is equivalent to the VHDL port size.

**VHDL-to-MATLAB Data Type Conversions (Continued)**

VHDL Types...	As Scalar Converts to...	As Array Converts to...
TIME	Type double for time values in seconds and type int64 for values representing simulator time increments (see the description of the 'time' option in “Starting the MATLAB Server” on page 2-49).	Arrays of type double or int64 with a size that is equivalent to the VHDL port size.
Enumerated types	Character array (string) that contains the MATLAB representation of a VHDL label or character literal. For example, the label high converts to 'high' and the character literal 'c' converts to ''c''.	Cell array of strings with each element equal to a label for the defined enumerated type. Each element is the MATLAB representation of a VHDL label or character literal. For example, the vector (one, '2', three) converts to the column vector ['one'; ''2''; 'three']. A user-defined enumerated type that contains only character literals, converts to a vector or array of characters as indicated for the types STD_LOGIC_VECTOR, STD_ULONGIC_VECTOR, BIT_VECTOR, SIGNED, and UNSIGNED.

The following table summarizes how the EDA Simulator Link MQ software converts supported Verilog data types to MATLAB types. Only scalar data types are supported for Verilog.



## Verilog-to-MATLAB Data Type Conversions

Verilog Types...	Converts to...
wire, reg	A character or a column vector of characters that matches the character literal for the desired logic states (bits).
integer	A 32-element column vector of characters that matches the character literal for the desired logic states (bits).

**Array Indexing Differences Between MATLAB and HDL.** In multidimensional arrays, the same underlying OS memory buffer maps to different elements in MATLAB and the HDL simulator (this mapping only reflects different ways the different languages offer for naming the elements of the same array). Be careful when using `matlabtb` and `matlabcp` functions to assign and interpret values consistently in both applications.

In HDL, a multidimensional array declared as:

```
type matrix_2x3x4 is array (0 to 1, 4 downto 2) of std_logic_vector(8 downto 5);
```

has a memory layout as follows:

```
bit  01 02 03 04  05 06 07 08  09 10 11 12  13 14 15 16  17 18 19 20  21 22 23 24
-
dim1 0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1  1  1
dim2 4  4  4  4  3  3  3  3  2  2  2  2  4  4  4  4  3  3  3  3  2  2  2  2
dim3 8  7  6  5  8  7  6  5  8  7  6  5  8  7  6  5  8  7  6  5  8  7  6  5
```

This same layout corresponds to the following MATLAB 4x3x2 matrix:

```
bit  01 02 03 04  05 06 07 08  09 10 11 12  13 14 15 16  17 18 19 20  21 22 23 24
-
dim1 1  2  3  4  1  2  3  4  1  2  3  4  1  2  3  4  1  2  3  4  1  2  3  4
dim2 1  1  1  1  2  2  2  2  3  3  3  3  1  1  1  1  2  2  2  2  3  3  3  3
dim3 1  1  1  1  1  1  1  1  1  1  1  1  2  2  2  2  2  2  2  2  2  2  2  2
```

Therefore, if H is the HDL array and M is the MATLAB matrix, the following indexed values are the same:

```

b1  H(0,4,8) = M(1,1,1)
b2  H(0,4,7) = M(2,1,1)
b3  H(0,4,6) = M(3,1,1)
b4  H(0,4,5) = M(4,1,1)
b5  H(0,3,8) = M(1,2,1)
b6  H(0,3,7) = M(2,2,1)
...
b19 H(1,3,6) = M(3,2,2)
b20 H(1,3,5) = M(4,2,2)
b21 H(1,2,8) = M(1,3,2)
b22 H(1,2,7) = M(2,3,2)
b23 H(1,2,6) = M(3,3,2)
b24 H(1,2,5) = M(4,3,2)

```

You can extend this indexing to N-dimensions. In general, the dimensions—if numbered from left to right—are reversed. The right-most dimension in HDL corresponds to the left-most dimension in MATLAB.

**Converting Data for Manipulation.** Depending on how your simulation MATLAB function uses the data it receives from the HDL simulator, the function may need to convert data to a different type before manipulating it. The following table lists circumstances under which such conversions are required.

### Required Data Conversions

If the Function Needs to...	Then...
Compute numeric data that is received as a type other than double	Use the <code>double</code> function to convert the data to type <code>double</code> before performing the computation. For example:  <pre>datas(inc+1) = double(idata);</pre>

**Required Data Conversions (Continued)**

<b>If the Function Needs to...</b>	<b>Then...</b>
Convert a standard logic or bit vector to an unsigned integer	<p>Use the <code>mv12dec</code> function to convert the data to an unsigned decimal value. For example:</p> <pre>uval = mv12dec(oport.val')</pre> <p>This example assumes the standard logic or bit vector is composed of the character literals '1' and '0' only. These are the only two values that can be converted to an integer equivalent.</p> <p>The <code>mv12dec</code> function converts the binary data that the MATLAB function receives from the entity's <code>osc_in</code> port to unsigned decimal values that MATLAB can compute.</p> <p>See <code>mv12dec</code> for more information on this function.</p>
Convert a standard logic or bit vector to a signed integer	<p>Use the following application of the <code>mv12dec</code> function to convert the data to a signed decimal value. For example:</p> <pre>suval = mv12dec(oport.val') - 2^length(oport.val);</pre> <p>This example assumes the standard logic or bit vector is composed of the character literals '1' and '0' only. These are the only two values that can be converted to an integer equivalent.</p>

**Examples**

The following code excerpt illustrates data type conversion of data passed in to a callback:

```
InDelayLine(1) = InputScale * mv12dec(iport.osc_in')/2^(Nbits-1);
```

This example tests port values of VHDL type STD\_LOGIC and STD\_LOGIC\_VECTOR by using the all function as follows:

```
all(oport.val == '1' | oport.val
== '0')
```

This example returns True if all elements are '1' or '0'.

**Converting Data for Return to the HDL Simulator.** If your simulation MATLAB function needs to return data to the HDL simulator, it may be necessary for you to first convert the data to a type supported by the EDA Simulator Link MQ software. The following tables list circumstances under which such conversions are required for VHDL and Verilog.

### VHDL Conversions for the HDL Simulator

To Return Data to an IN Port of Type...	Then...
STD_LOGIC, STD_ULONGIC, or BIT	Declare the data as a character that matches the character literal for the desired logic state. For STD_LOGIC and STD_ULONGIC, the character can be 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', or '-'. For BIT, the character can be '0' or '1'. For example:  <pre>iport.s1 = 'X'; %STD_LOGIC iport.bit = '1'; %BIT</pre>
STD_LOGIC_VECTOR, STD_ULONGIC_VECTOR, BIT_VECTOR, SIGNED, or UNSIGNED	Declare the data as a column vector or row vector of characters (as defined above) with one bit per character. For example:  <pre>iport.s1v = 'X10ZZ'; %STD_LOGIC_VECTOR iport.bitv = '10100'; %BIT_VECTOR iport.uns = dec2mvl(10,8); %UNSIGNED, 8 bits</pre>
Array of STD_LOGIC_VECTOR, STD_ULONGIC_VECTOR, BIT_VECTOR, SIGNED, or UNSIGNED	Declare the data as an array of type character with a size that is equivalent to the VHDL port size. See “Array Indexing Differences Between MATLAB and HDL” on page 2-23.

**VHDL Conversions for the HDL Simulator (Continued)**

<b>To Return Data to an IN Port of Type...</b>	<b>Then...</b>
INTEGER or NATURAL	<p>Declare the data as an array of type <code>int32</code> with a size that is equivalent to the VHDL array size. Alternatively, convert the data to an array of type <code>int32</code> with the MATLAB <code>int32</code> function before returning it. Be sure to limit the data to values with the range of the VHDL type. If necessary, check the right and left fields of the <code>portinfo</code> structure. For example:</p> <pre>iport.int = int32(1:10)';</pre>
REAL	<p>Declare the data as an array of type <code>double</code> with a size that is equivalent to the VHDL port size. For example:</p> <pre>iport.dbl = ones(2,2);</pre>
TIME	<p>Declare a VHDL <code>TIME</code> value as time in seconds, using type <code>double</code>, or as an integer of simulator time increments, using type <code>int64</code>. You can use the two formats interchangeably and what you specify does not depend on the <code>hdldaemon 'time'</code> option (see “Starting the MATLAB Server” on page 2-49), which applies to IN ports only. Declare an array of <code>TIME</code> values by using a MATLAB array of identical size and shape. All elements of a given port are restricted to time in seconds (type <code>double</code>) or simulator increments (type <code>int64</code>), but otherwise you can mix the formats. For example:</p> <pre>iport.t1 = int64(1:10)'; %Simulator time                     %increments iport.t2 = 1e-9; %1 nsec</pre>

### VHDL Conversions for the HDL Simulator (Continued)

To Return Data to an IN Port of Type...	Then...
Enumerated types	<p>Declare the data as a string for scalar ports or a cell array of strings for array ports with each element equal to a label for the defined enumerated type. The 'label' field of the portinfo structure lists all valid labels (see “Gaining Access to and Applying Port Information” on page 2-18). Except for character literals, labels are not case sensitive. In general, you should specify character literals completely, including the single quotes, as shown in the first example below.</p> <pre data-bbox="556 685 1053 772"> iport.char = {'A', 'B'}; %Character                 %literal iport.undef = 'mylabel'; %User-defined label                     </pre>
Character array for standard logic or bit representation	<p>Use the dec2mv1 function to convert the integer. For example:</p> <pre data-bbox="556 876 1023 911"> oport.slva =dec2mv1([23 99],8)';                     </pre> <p>This example converts two integers to a 2-element array of standard logic vectors consisting of 8 bits.</p>

### Verilog Conversions for the HDL Simulator

To Return Data to an input Port of Type...	Then...
reg, wire	<p>Declare the data as a character or a column vector of characters that matches the character literal for the desired logic state ('0' or '1'). For example:</p> <pre data-bbox="734 1354 964 1388"> iport.bit = '1';                     </pre>
integer	<p>Declare the data as a 32-element column vector of characters (as defined above) with one bit per character.</p>

## Sample MATLAB Test Bench Function

This section uses a sample MATLAB function to identify sections of a MATLAB test bench function required by the EDA Simulator Link MQ software. The full text of the code used in this sample can be seen in the section M-Function Example: manchester\_decoder.m on page 2-33.

---

**Note** The example uses a VHDL entity and MATLAB function code drawn from the decoder portion of the Manchester Receiver demo. For the complete VHDL and M-code listings, see the following files:

- `matlabroot\toolbox\modelsim\modelsimdemos\vhdl\manchester\decoder.vhd`
  - `matlabroot\toolbox\modelsim\modelsimdemos\manchester_decoder.m`
- 

The first step to coding a MATLAB test bench function is to understand how the data modeled in the VHDL entity maps to data in the MATLAB environment. The VHDL entity decoder is defined as follows:

```
ENTITY decoder IS
PORT (
    isum    : IN std_logic_vector(4 DOWNTO 0);
    qsum    : IN std_logic_vector(4 DOWNTO 0);
    adj     : OUT std_logic_vector(1 DOWNTO 0);
    dvalid  : OUT std_logic;
    odata   : OUT std_logic
);
END decoder ;
```

The following discussion highlights key lines of code in the definition of the manchester\_decoder MATLAB function:

### 1 Specify the MATLAB function name and required parameters.

The following code is the function declaration of the manchester\_decoder MATLAB function.

```
function [iport,tnext] = manchester_decoder(oport,tnow,portinfo)
```

See “Defining Link Functions and Link Function Parameters” on page 2-14.

The function declaration performs the following actions:

- Names the function. This declaration names the function `manchester_decoder`, which differs from the entity name `decoder`. Because the names differ, the function name must be specified explicitly later when the entity is initialized for verification with the `matlabtb` or `matlabtbeval` HDL simulator command. See “Naming a MATLAB Link Function” on page 2-37.
- Defines required argument and return parameters. A MATLAB test bench function *must* return two parameters, `oport` and `tnext`, and pass three arguments, `oport`, `tnow`, and `portinfo`, and *must* appear in the order shown. See “Defining Link Functions and Link Function Parameters” on page 2-14.

Note that the function outputs must be initialized to empty values, as in the following code example:

```
tnext = [];  
oport = struct();
```

Recommended practice is to initialize the function outputs at the beginning of the function.

The following figure shows the relationship between the entity’s ports and the MATLAB function’s `oport` and `oport` parameters.



For more information on the required MATLAB link function parameters, see “Defining Link Functions and Link Function Parameters” on page 2-14.

### **2 Make note of the data types of ports defined for the entity being simulated.**



The EDA Simulator Link MQ software converts HDL data types to comparable MATLAB data types and vice versa. As you develop your MATLAB function, you must know the types of the data that it receives from the HDL simulator and needs to return to the HDL simulator.

The VHDL entity defined for this example consists of the following ports:

### VHDL Example Port Definitions

Port	Direction	Type...	Converts to/Requires Conversion to...
isum	IN	STD_LOGIC_VECTOR(4 DOWNTO 0)	A 5-bit column or row vector of characters where each bit maps to standard logic character 0 or 1.
qsum	IN	STD_LOGIC_VECTOR(4 DOWNTO 0)	A 5-bit column or row vector of characters where each bit maps to standard logic character 0 or 1.
adj	OUT	STD_LOGIC_VECTOR(1 DOWNTO 0)	A 2-element column vector of characters. Each character matches a corresponding character literal that represents a logic state and maps to a single bit.

**VHDL Example Port Definitions (Continued)**

<b>Port</b>	<b>Direction</b>	<b>Type...</b>	<b>Converts to/Requires Conversion to...</b>
dvalid	OUT	STD_LOGIC	A character that matches the character literal representing the logic state.
odata	OUT	STD_LOGIC	A character that matches the character literal representing the logic state.

For more information on interface data type conversions, see “Performing Data Type Conversions” on page 2-20.

**3 Set up any required timing parameters.**

The tnext assignment statement sets up timing parameter tnext such that the simulator calls back the MATLAB function every nanosecond.

```
tnext = tnow+1e-9;
```

**4 Convert output port data to appropriate MATLAB data types for processing.**

The following code excerpt illustrates data type conversion of output port data.

```
%% Compute one row and plot
isum = isum + 1;
adj(isum) = mvl2dec(oport.adj');
data(isum) = mvl2dec([oport.dvalid oport.odata]);
.
.
.
```

The two calls to `mv12dec` convert the binary data that the MATLAB function receives from the entity's output ports, `adj`, `dvalid`, and `odata` to unsigned decimal values that MATLAB can compute. The function converts the 2-bit transposed vector `oport.adj` to a decimal value in the range 0 to 4 and `oport.dvalid` and `oport.odata` to the decimal value 0 or 1.

“Performing Data Type Conversions” on page 2-20 provides a summary of the types of data conversions to consider when coding simulation MATLAB functions.

## 5 Convert data to be returned to the HDL simulator.

The following code excerpt illustrates data type conversion of data to be returned to the HDL simulator.

```

if isum == 17
    iport.isum = dec2mvl(isum,5);
    iport.qsum = dec2mvl(qsum,5);
else
    iport.isum = dec2mvl(isum,5);
end

```

The three calls to `dec2mvl` convert the decimal values computed by MATLAB to binary data that the MATLAB function can deposit to the entity's input ports, `isum` and `qsum`. In each case, the function converts a decimal value to 5-element bit vector with each bit representing a character that maps to a character literal representing a logic state.

“Converting Data for Return to the HDL Simulator” on page 2-26 provides a summary of the types of data conversions to consider when returning data to the HDL simulator.

### M-Function Example: `manchester_decoder.m`

```

function [iport,tnext] = manchester_decoder(oport,tnow,portinfo)
% MANCHESTER_DECODER Test bench for VHDL 'decoder'
% [IPORT,TNEXT]=MANCHESTER_DECODER(OPORT,TNOW,PORTINFO) -
% Implements a test of the VHDL decoder entity which is part
% of the Manchester receiver demo. This test bench plots
% the IQ mapping produced by the decoder.
%

```

```
%      iport          oport
%      +-----+
% isum -(5)->|          |-(2)-> adj
% qsum -(5)->| decoder |-(1)-> dvalid
%          |          |-(1)-> odata
%      +-----+
%
%
% isum - Inphase Convolution value
% qsum - Quadrature Convolution value
% adj  - Clock adjustment ('01','00','10')
% dvalid - Data validity ('1' = data is valid)
% odata - Recovered data stream
%
% Adjust = 0 (00b), generate full 16 cycle waveform

% Copyright 2003-2004 The MathWorks, Inc.
% $Revision: 1.1.4.1 $ $Date: 2007/08/31 00:55:58 $

persistent isum;
persistent qsum;
%persistent ga;
persistent x;
persistent y;
persistent adj;
persistent data;
global testisdone;
% This useful feature allows you to manually
% reset the plot by simply typing: >manchester_decoder
tnext = [];
iport = struct();

if nargin == 0,
    isum = [];
    return;
end

if exist('portinfo') == 1
    isum = [];
end
```

```

tnext = tnow+1e-9;
if isempty(isum), %% First call
    scale = 9;
    isum = 0;
    qsum = 0;
    for k=1:2,
        ga(k) = subplot(2,1,k);
        axis([-1 17 -1 17]);
        ylabel('Quadrature');
        line([0 16],[8 8], 'Color', 'r', 'LineStyle', ':', 'LineWidth', 1)
        line([8 8],[0 16], 'Color', 'r', 'LineStyle', ':', 'LineWidth', 1)
    end
    xlabel('Inphase');
    subplot(2,1,1);
    title('Clock Adjustment (adj)');
    subplot(2,1,2);
    title('Data with Validity');
    iport.isum = '00000';
    iport.qsum = '00000';
    return;
end

% compute one row, then plot
isum = isum + 1;
adj(isum) = bin2dec(oport.adj');
data(isum) = bin2dec([oport.dvalid oport.odata]);

if isum == 17,
    subplot(2,1,1);
    for k=0:16,
        if adj(k+1) == 0, % Bang on!
            line(k,qsum, 'color', 'k', 'Marker', 'o');
        elseif adj(k+1) == 1, %
            line(k,qsum, 'color', 'r', 'Marker', '<');
        else
            line(k,qsum, 'color', 'b', 'Marker', '>');
        end
    end
    subplot(2,1,2);
    for k=0:16,

```

```
        if data(k+1) < 2, % Invalid
            line(k,qsum,'color','r','Marker','X');
        else
            if data(k+1) == 2, %Valid and 0!
                line(k,qsum,'color','g','Marker','o');
            else
                line(k,qsum,'color','k','Marker','.');
            end
        end
    end
end

isum = 0;
qsum = qsum + 1;
if qsum == 17,
    qsum = 0;
    disp('done');
    tnext = []; % suspend callbacks
    testisdone = 1;
    return;
end
iport.isum = dec2bin(isum,5);
iport.qsum = dec2bin(qsum,5);
else
    iport.isum = dec2bin(isum,5);
end
end
```

## Associating a MATLAB® Link Function with an HDL Module

### In this section...

“Overview” on page 2-37

“Naming a MATLAB Link Function” on page 2-37

“Associating the HDL Module Component with the MATLAB Link Function” on page 2-38

“Specifying HDL Signal/Port and Module Paths for MATLAB Link Sessions” on page 2-38

“Specifying TCP/IP Values” on page 2-40

“Scheduling Options for a Link Session” on page 2-40

### Overview

This section describes establishing a relationship between the link function and the HDL model in the ModelSim® simulator by naming the link function (either implicitly or explicitly) and using scheduling options (action based on a specific time or event and registering callbacks) for the MATLAB® link session.

### Naming a MATLAB Link Function

You can name and specify a MATLAB link function however you like, so long as you adhere to MATLAB function and file naming guidelines. By default, the EDA Simulator Link™ MQ software assumes the name for a MATLAB function matches the name of the HDL module that the function verifies or visualizes. For example, if you name the HDL module `mystdlogic`, the EDA Simulator Link MQ software assumes the corresponding MATLAB function is `mystdlogic` and resides in the file `mystdlogic.m`.

Should you name the m-function or m-file something different than the HDL instance, you must specify the `-mfunc` parameter of one of the link functions and provide the m-function name.

For details on MATLAB function naming guidelines, see "MATLAB Programming Tips" on files and file names in the MATLAB documentation.

## Associating the HDL Module Component with the MATLAB Link Function

By default, the EDA Simulator Link MQ software assumes the name for a MATLAB function matches the name of the HDL module that the function verifies or visualizes. See “Naming a MATLAB Link Function” on page 2-37.

In the Oscillator demo, the HDL model instantiates an HDL entity as the component `u_osc_filter` (see `osc_top.vhd`). After the HDL simulator compiles and loads the HDL model, an association must be formed between the `u_osc_filter` component and the MATLAB component function `oscfilter`. To do this, the HDL simulator command `matlabcp` is invoked when the simulation is set up.

```
matlabcp u_osc_filter -mfunc oscfilter
```

The `matlabcp` command instructs the HDL simulator to call back the `oscfilter` function when `u_osc_filter` executes in the simulation.

## Specifying HDL Signal/Port and Module Paths for MATLAB Link Sessions

The rules stated in this section are for signal/port and module path specifications for MATLAB link sessions. Other specifications may work but are not guaranteed to work in this or future releases.

In the following example,

```
matlabcp u_osc_filter -mfunc oscfilter
```

`u_osc_filter` is the top level component. However, if you are specifying a subcomponent, you must follow valid module path specifications for MATLAB link sessions.

---

**Note** HDL designs generally do have hierarchy; that is the reason for this syntax. This is not a file name hierarchy.

---



## Path Specifications for MATLAB Link Sessions with Verilog Top Level

These path specifications rules must be followed:

- Path specification must start with a top-level module name.
- Path specification can include "." or "/" path delimiters, but cannot include a mixture.
- The leaf module or signal must match the HDL language of the top-level module.

The following are valid signal and module path specification examples:

```
top.port_or_sig
/top/sub/port_or_sig
top
top/sub
top.sub1.sub2
```

The following are invalid signal and module path specification examples:

```
top.sub/port_or_sig
:sub:port_or_sig
:
:sub
```

## Path Specifications for MATLAB Link Sessions with VHDL Top Level

These path specifications rules must be followed:

- Path specification may include the top-level module name but it is not required.
- Path specification can include "." or "/" path delimiters, but cannot include a mixture.
- The leaf module or signal must match the HDL language of the top-level module.

The following are valid signal and module path specification examples:

```
top.port_or_sig
/sub/port_or_sig
top
top/sub
top.sub1.sub2
```

The following are invalid signal and module path specification examples:

```
top.sub/port_or_sig
:sub:port_or_sig
:
:sub
```

### Specifying TCP/IP Values

When providing TCP/IP information for a MATLAB link function, you can choose a TCP/IP port number or TCP/IP port alias or service name.

If the HDL simulator and MATLAB are running on the same system, the TCP/IP specification identifies a unique TCP/IP socket port to be used for the link. If the two applications are running on different systems, you must specify a remote hostname or Internet address in addition to the socket port. See Appendix D, “TCP/IP Socket Communication” for more detail in specifying TCP/IP values.

For example,

```
vsim> matlabcp u_osc_filter -mfunc oscfilter -socket 4449
```

A remote connection might look like this:

```
>matlabcp u_osc_filter -mfunc oscfilter -socket computer93:4449
```

or

```
>matlabcp u_osc_filter -mfunc oscfilter -socket 4449@computer23
```

### Scheduling Options for a Link Session

There are two ways to schedule the invocation of a link function:

- Using the arguments to the `matlabcp` or `matlabtb` functions (“Scheduling Link Functions Using Link Function Parameters” on page 2-41)
- Inside the MATLAB m-function using the `tnext` parameter (“Scheduling Link Functions Using the `tnext` Parameter of an M-Function” on page 2-45)

You can schedule a MATLAB simulation function to execute under any of the following conditions:

- At a time that the MATLAB function passes to the HDL simulator with the `tnext` parameter
- Based on a time specification that can include discrete time values, repeat intervals, and a stop time
- When a specified signal experiences a rising edge—changes from '0' to '1'
- When a specified signal experiences a falling edge—changes from '1' to '0'
- Based on a sensitivity list—when a specified signal changes state

Decide on a combination of options that best meet your test bench application requirements. For details on using the `tnext` parameter and information on setting other scheduling parameters, see “Scheduling Link Functions Using the `tnext` Parameter of an M-Function” on page 2-45.

### **Scheduling Link Functions Using Link Function Parameters**

By default, the EDA Simulator Link MQ software invokes a MATLAB test bench function once (when time equals 0). If you want to apply more control and execute the MATLAB function more than once, decide on scheduling options that specify when and how often the EDA Simulator Link MQ software is to invoke the relevant MATLAB function. Depending on your choices, you may need to modify the function or specify specific arguments when you begin a MATLAB test bench session with the `matlabtb` function.

In addition, the `matlabtb` function can include parameters that control when the MATLAB function executes.

You must specify at least one instance of a VHDL entity or Verilog module in your HDL model. By default, the command establishes a shared memory communication link and associates the specified instance with a MATLAB

function that has the same name as the instance. See “Associating the HDL Module Component with the MATLAB Link Function” on page 2-38.

The `matlabtbval` function executes the MATLAB function immediately, while `matlabtb` provides several options for scheduling MATLAB function execution. The following table lists the various scheduling options.

### Simulation Scheduling Options

To Specify MATLAB Function Execution...	Include...	Where...
At explicit times	<code>time[, ...]</code>	<p><code>time</code> represents one of <math>n</math> time values, past time 0, at which the MATLAB function executes.</p> <p>For example:</p> <p style="text-align: center;"><code>10 ns, 10 ms, 10 sec</code></p> <p>The MATLAB function executes when time equals 0 and then 10 nanoseconds, 10 milliseconds, and 10 seconds from time zero.</p> <hr/> <p><b>Note</b> 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.</p> <hr/>

**Simulation Scheduling Options (Continued)**

<b>To Specify MATLAB Function Execution...</b>	<b>Include...</b>	<b>Where...</b>
At a combination of explicit times and repeatedly at an interval	time[, ...] -repeat n	<p>time represents one of n time values at which the MATLAB function executes and the n specified with -repeat represents an interval between MATLAB function executions. The interface applies the union of the two options.</p> <p>For example:</p> <p style="padding-left: 40px;">5 ns -repeat 10 ns</p> <p>The MATLAB function executes at time equals 0 ns, 5 ns, 15 ns, 25 ns, and so on.</p>

**Simulation Scheduling Options (Continued)**

<b>To Specify MATLAB Function Execution...</b>	<b>Include...</b>	<b>Where...</b>
When a specific signal experiences a rising or falling edge	-rising signal[, ...] -falling signal[, ...]	signal represents a path name of a signal defined as a logic type—STD_LOGIC, BIT, X01, and so on.
On change of signal values (sensitivity list)	-sensitivity signal[, ...]	<p>signal represents a path name of a signal defined as any type. If the value of one or more signals in the specified list changes, the interface invokes the MATLAB function.</p> <hr/> <p><b>Note</b> Use of this option for INOUT ports can result in double calls.</p> <hr/> <p>If you specify the option with no signals, the interface is sensitive to value changes for all signals.</p> <p>For example:</p> <pre style="margin-left: 40px;">-sensitivity /randnumgen/dout</pre> <p>The MATLAB function executes if the value of dout changes.</p>

---

**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 ModelSim SE/PE rules to resolve signal specifications.

---

Consider the following matlabbtb command:

```
VSIM n> matlabtb modsimrand -rising /modsimrand/clock
-socket 4449
```

This command links an instance of the entity `modsimrand` to function `modsimrand.m`, which executes within the context of MATLAB based on specified timing parameters. In this case, the MATLAB function is called when the signal `/modsimrand/clock` experiences a rising edge.

Arguments in the command line specify the following:

<code>modsimrand</code>	That an instance of the entity <code>modsimrand</code> be linked with the MATLAB function <code>modsimrand</code> .
<code>-rising /modsimrand/clock</code>	That the MATLAB function <code>modsimrand</code> be called when the signal <code>/modsimrand/clock</code> changes from '0' to '1'.
<code>-socket 4449</code>	That TCP/IP socket port 4449 be used to establish a communication link with MATLAB.

To verify that the `matlabtb` or `matlabtbeval` command established a connection, change your input focus to MATLAB and call the function `hdldaemon` with the 'status' option as follows:

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

If a connection exists, the function returns the message

```
HLDaemon socket server is running on port 4449 with 1 connection
```

## Scheduling Link Functions Using the `tnext` Parameter of an M-Function

You can control the callback timing of a MATLAB test bench function by using that function's `tnext` parameter. This parameter passes a time value to the HDL simulator, which gets added to the MATLAB function's simulation schedule. If the function returns a null value (`[]`), no new entries are added to the schedule.

You can set the value of `tnext` to a value of type `double` or `int64`. The following table explains how the interface converts each type of data for use in the HDL simulator environment.

**Time Representations for `tnext` Parameter**

If You Specify a...	The Interface...
double value	Converts the value to seconds. For example, the following value converts to the simulation time nearest to 1 nanosecond as a multiple of the current HDL simulator time resolution.  <code>tnext = 1e-9</code>
int64 value	Converts to an integer multiple of the current HDL simulator time resolution limit. For example, the following value converts to 100 ticks of the current time resolution.  <code>tnext=int64(100)</code>

---

**Note** The `tnext` parameter represents time from the start of the simulation. Therefore, `tnext` should always be greater than `tnow`. If it is less, nothing is scheduled.

---

**Example.** In the Oscillator demo, the `oscfilter` function calculates a time interval at which callbacks should be executed. This interval is calculated on the first call to `oscfilter` and is stored in the variable `fastestrate`. The variable `fastestrate` is the sample period of the fastest oversampling rate supported by the filter, derived from a base sampling period of 80 ns.

The following assignment statement sets the timing parameter `tnext`, which schedules the next callback to the MATLAB component function, relative to the current simulation time (`tnow`).

```
tnext = tnow + fastestrate;
```



A new value for `tnext` is returned each time the function is called.

## Running MATLAB® Link Sessions

In this section...
“Overview” on page 2-48
“Process for Running MATLAB Link Sessions” on page 2-48
“Placing a MATLAB Test Bench or Component Function on the MATLAB Search Path” on page 2-49
“Starting the MATLAB Server” on page 2-49
“Checking the MATLAB Server’s Link Status” on page 2-51
“Starting ModelSim SE/PE for Use with MATLAB” on page 2-51
“Applying Stimuli with the HDL Simulator force Command” on page 2-51
“Running a Link Session” on page 2-52
“Restarting a Link Session” on page 2-54
“Stopping a Link Session” on page 2-54

### Overview

The EDA Simulator Link™ MQ software offers flexibility in how you start and control an HDL model test bench or component session with MATLAB® software. A MATLAB link session is the application of a `matlabtb`, `matlabtbeval`, or `matlabcp` function.

### Process for Running MATLAB Link Sessions

To start and control the execution of a simulation in the MATLAB environment, perform the following steps:

- 1 Place MATLAB link function on the MATLAB search path.
- 2 Check the MATLAB server’s link status.
- 3 Start the MATLAB server.
- 4 Launch ModelSim SE/PE for use with MATLAB.

- 5 Load an HDL model in ModelSim SE/PE for simulation and verification with MATLAB.
- 6 Decide on how you want to schedule invocations of the MATLAB test bench function.
- 7 Register callbacks for the MATLAB link session.
- 8 Apply test bench stimuli.
- 9 Run and monitor the test bench session.
- 10 Restart simulator during a test bench session.
- 11 Stop a test bench session.

## Placing a MATLAB Test Bench or Component Function on the MATLAB Search Path

The MATLAB function associated with an HDL component must be on the MATLAB search path or reside in the current working directory (see the MATLAB `cd` function). To verify whether the function is accessible, use the MATLAB `which` function. The following call to `which` checks whether the function `MyVhdlFunction` is on the MATLAB search path:

```
which MyVhdlFunction
D:\work\modelsim\MySym\MyVhdlFunction.m
```

If the specified function is on the search path, `which` displays the complete path to the function's M-file. If the function is not on the search path, `which` informs you that the file was not found.

To add a MATLAB function to the MATLAB search path, open the Set Path window by clicking **File > Set Path**, or use the `addpath` command. Alternatively, for temporary access, you can change the MATLAB working directory to a desired location with the `cd` command.

## Starting the MATLAB Server

Start the MATLAB server as follows:

- 1 Start MATLAB.
- 2 In the MATLAB Command Window, call the `hdldaemon` function with property name/property value pairs that specify whether the EDA Simulator Link MQ software is to perform the following tasks:
  - Use shared memory or TCP/IP socket communication
  - Return time values in seconds or as 64-bit integers

Use the following syntax:

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

For example, the following command specifies using socket communication on port 4449 and a 64-bit time resolution format for the MATLAB function's output ports.

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

See `hdldaemon` reference documentation for when and how to specify property name/property value pairs and for more examples of using `hdldaemon`.

---

**Note** The communication mode that you specify (shared memory or TCP/IP sockets) must match what you specify for the communication mode when you initialize the HDL simulator for use with a MATLAB link session using the `matlabtb` or `matlabtbeval` HDL simulator command. In addition, if you specify TCP/IP socket mode, the socket port that you specify with this function and the HDL simulator command must match. For more information on modes of communication, see “Choosing TCP/IP Socket Ports” on page D-2. For more information on establishing the HDL simulator end of the communication link, see “Associating the HDL Module Component with the MATLAB Link Function” on page 2-38.

---

The MATLAB server can service multiple simultaneous HDL simulator modules and clients. However, your M-code must track the I/O associated with each entity or client.

---

**Note** You cannot begin an EDA Simulator Link MQ transaction between MATLAB and the HDL simulator from MATLAB. The MATLAB server simply responds to function call requests that it receives from the HDL simulator.

---

## Checking the MATLAB Server's Link Status

The first step to starting an HDL simulator and MATLAB test bench session is to check the MATLAB server's link status. Is the server running? If the server is running, what mode of communication and, if applicable, what TCP/IP socket port is the server using for its links? You can retrieve this information by using the MATLAB function `hdldaemon` with the `'status'` option. For example:

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

The function displays a message that indicates whether the server is running and, if it is running, the number of connections it is handling. For example:

```
HLDaemon socket server is running on port 4449 with 0 connections
```

If the server is not running, the message reads

```
HLDaemon is NOT running
```

See 'Link Status' in the `hdldaemon` reference documentation for information on determining the mode of communication and the TCP/IP socket in use.

## Starting ModelSim SE/PE for Use with MATLAB

Start ModelSim SE/PE directly from MATLAB by calling the MATLAB function `vsim`. See "Starting the HDL Simulator" on page 1-23 for instructions on using `vsim`.

## Applying Stimuli with the HDL Simulator force Command

After you establish a link between the HDL simulator and MATLAB, you are ready to apply stimuli to the test bench environment. One way of applying

stimuli is through the `iport` parameter of the linked MATLAB function. This parameter forces signal values by deposit.

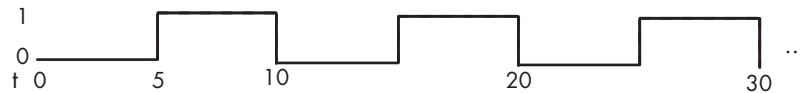
Other ways include issuing force commands in the ModelSim main window or using the **Edit > Clock** option in the **ModelSim Signals** window.

For example, consider the following sequence of force commands:

```
VSIM n> force clk 0 0 ns, 1 5 ns -repeat 10 ns
VSIM n> force clk_en 1 0
VSIM n> force reset 0 0
```

These commands drive the following signals:

- The `clk` signal to 0 at 0 nanoseconds after the current simulation time and to 1 at 5 nanoseconds after the current HDL simulation time. This cycle repeats starting at 10 nanoseconds after the current simulation time, causing transitions from 1 to 0 and 0 to 1 every 5 nanoseconds, as the following diagram shows.



For example,

```
force /foobar/clk 0 0, 1 5 -repeat 10
```

- The `clk_en` signal to 1 at 0 nanoseconds after the current simulation time.
- The `reset` signal to 0 at 0 nanoseconds after the current simulation time.

## Running a Link Session

A typical sequence for running a simulation interactively from the main HDL simulator window is shown below:

- 1 Start the simulation by entering the HDL simulator run command.

The run command offers a variety of options for applying control over how a simulation runs. For example, you can specify that a simulation run for several time steps.

The following command instructs the HDL simulator to run the loaded simulation for 50000 time steps:

```
run 50000
```

- 2** Set breakpoints in the HDL and MATLAB code to verify and analyze simulation progress and correctness.

How you set breakpoints in the HDL simulator will vary depending on what simulator application you are using. The following list demonstrates some ways you can set breakpoints in the MATLAB environment:

- Click next to an executable statement in the breakpoint alley of the Editor/Debugger
- Click the Set/Clear Breakpoint button on the toolbar
- Select **Set/Clear Breakpoint** on the **Breakpoints** menu
- Select **Set/Clear Breakpoint** on the context menu
- Call the `dbstop` function

- 3** Step through the simulation and examine values.

How you step through the simulation in the HDL simulator will vary depending on what simulator application you are using. The following list demonstrates some ways you can step through code in the MATLAB environment.

- Click the Step, Step In, or Step Out toolbar button
- Select the **Step**, **Step In**, or **Step Out** option on the **Debug** menu
- Select the **Go Until Cursor** menu option
- Call the `dbstep` function

- 4** When you block execution of the MATLAB function, the HDL simulator also blocks and remains blocked until you clear all breakpoints in the function's M-code.

- 5** Resume the simulation, as needed.

How you resume the simulation in the HDL simulator will vary depending on what simulator application you are using. The following

list demonstrates ways you can resume a simulation in the MATLAB environment.

- Click the Continue toolbar button
- Select the **Continue**, **Run**, or **Save and Run** option on the **Debug** menu
- Call the dbcont function

The following HDL simulator command resumes a simulation:

```
run -continue
```

For more information on HDL simulator and MATLAB debugging features, see the appropriate HDL simulator documentation and MATLAB online help or documentation.

### Restarting a Link Session

Because the HDL simulator issues the service requests during a MATLAB test bench session, you must restart a test bench session from the HDL simulator. To restart a session, perform the following steps:

- 1** Make the HDL simulator your active window, if your input focus was not already set to that application.
- 2** Reload HDL design elements and reset the simulation time to zero.
- 3** Reissue the matlabb command.

---

**Note** To restart a simulation that is in progress, issue a break command and end the current simulation session before restarting a new session.

---

### Stopping a Link Session

When you are ready to stop a test bench session, it is best to do so in an orderly way to avoid possible corruption of files and to ensure that all application tasks shut down appropriately. You should stop a session as follows:



- 1** Make the HDL simulator your active window, if your input focus was not already set to that application.
- 2** Halt the simulation. You must quit the simulation at the HDL simulator side or MATLAB may hang until the simulator is quit.
- 3** Close your project.
- 4** Exit the HDL simulator, if you are finished with the application.
- 5** Quit MATLAB, if you are finished with the application. If you want to shut down the server manually, stop the server by calling `hdldaemon` with the 'kill' option:

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

For more information on closing HDL simulator sessions, see the HDL simulator documentation.

## MATLAB and ModelSim Tutorial

In this section...
“Tutorial Overview” on page 2-56
“Setting Up Tutorial Files” on page 2-56
“Starting the MATLAB Server” on page 2-57
“Setting Up ModelSim” on page 2-58
“Developing the VHDL Code” on page 2-60
“Compiling the VHDL File” on page 2-63
“Loading the Simulation” on page 2-63
“Developing the MATLAB Function” on page 2-66
“Running the Simulation” on page 2-68
“Shutting Down the Simulation” on page 2-72

### Tutorial Overview

This chapter guides you through the basic steps for setting up an EDA Simulator Link™ MQ application that uses MATLAB to verify a simple HDL design. In this tutorial, we develop, simulate, and verify a model of a pseudorandom number generator based on the Fibonacci sequence. The model is coded in VHDL.

---

**Note** This tutorial requires MATLAB, ModelSim, and the EDA Simulator Link MQ software.

---

### Setting Up Tutorial Files

To ensure that others can access copies of the tutorial files, set up a directory for your own tutorial work:

- 1 Create a directory outside the scope of your MATLAB installation directory into which you can copy the tutorial files. The directory must be writable. This tutorial assumes that you create a directory named MyPlayArea.

- 2 Copy the following files to the directory you just created:

```
matlabroot\toolbox\modelsim\modelsimdemos\modsimrand_plot.m  
matlabroot\toolbox\modelsim\modelsimdemos\VHDL\modsimrand\modsimrand.vhd
```

## Starting the MATLAB Server

This section describes starting MATLAB, setting up the current directory for completing the tutorial, starting the product's MATLAB server component, and checking for client connections, using the server's TCP/IP socket mode. These instructions assume you are familiar with the MATLAB user interface.

Perform the following steps:

- 1 Start MATLAB.
- 2 Set your MATLAB current directory to the directory you created in “Setting Up Tutorial Files” on page 2-56.
- 3 Check whether the MATLAB server is running. Do this by calling the function `hdldaemon` with the 'status' option in the MATLAB Command Window as shown below.

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

If the server is not running, the function displays

```
HLDaemon is NOT running
```

If the server is running in TCP/IP socket mode, the message reads

```
HLDaemon socket server is running on Port portnum with 0 connections
```

If the server is running in shared memory mode, the message reads

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

- 4 If the server is not currently running, skip to step 5. If the server is currently running, shut down the server by typing

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

The following message appears, confirming that the server was shut down.

```
HDLDaemon server was shut down
```

- 5** The next step is to start the server in TCP/IP socket mode. To do this, call `hdldaemon` with the property name/property value pair `'socket' 0`. The value 0 specifies that the operating system assign the server a TCP/IP socket port that is available on your system. For example

```
hdldaemon('socket', 0)
```

The server informs you that it has started by displaying the following message. The `portnum` will be specific to your system:

```
HDLDaemon socket server is running on Port portnum with 0 connections
```

Other options that you can specify in the `hdldaemon` function call include the following:

- Shared memory communication instead of TCP/IP socket communication
- Whether time will be returned as scaled or a 64-bit integer

For details on how to specify the various options, see the description of `hdldaemon`.

---

**Note** The `hdldaemon` function can handle multiple connections that are initiated by multiple commands from a single ModelSim session or multiple sessions.

---

## Setting Up ModelSim

This section describes the basic procedure for starting ModelSim and setting up a ModelSim design library. These instructions assume you are familiar with the ModelSim user interface.

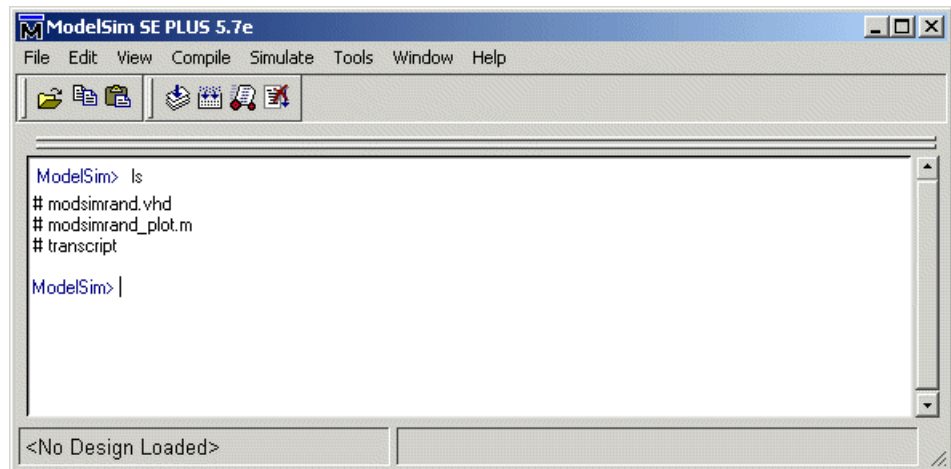
Perform the following steps:

- 1 Start ModelSim from the MATLAB environment by calling the function `vsim` in the MATLAB Command Window.

```
vsim
```

This function launches and configures ModelSim for use with the EDA Simulator Link MQ software. The first directory of ModelSim matches your MATLAB current directory.

- 2 Verify the current ModelSim directory. You can verify that the current ModelSim directory matches the MATLAB current directory by entering the `ls` command in the ModelSim command window.

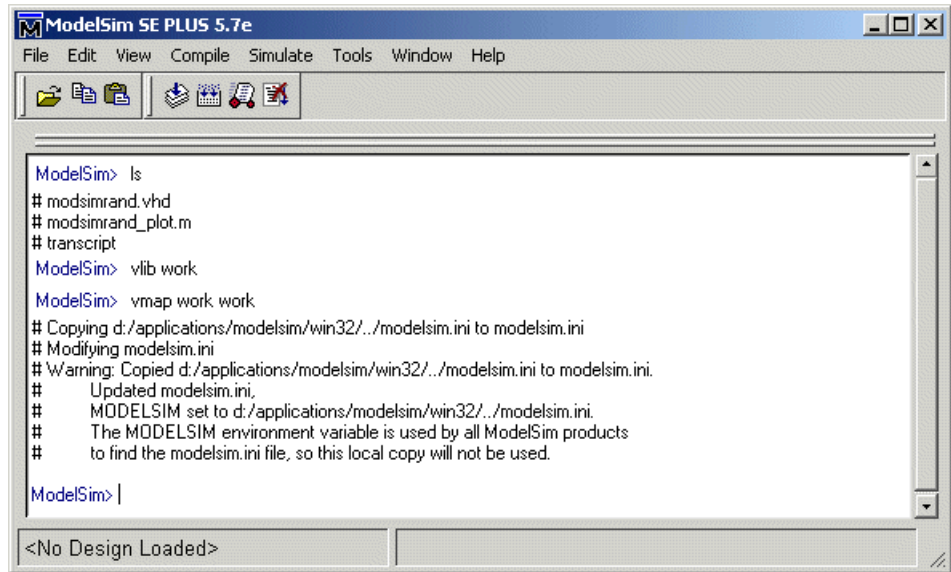


The command should list the files `modsimrand.vhd`, `modsimrand_plot.m`, and `transcript`.

- 3 Create a design library to hold your demo compilation results. To create the library and required `_info` file, enter the `vlib` and `vmap` commands as follows:

```
ModelSim> vlib work
```

```
ModelSim> vmap work work
```



```
ModelSim SE PLUS 5.7e
File Edit View Compile Simulate Tools Window Help

ModelSim> ls
# modsimrand.vhd
# modsimrand_plot.m
# transcript
ModelSim> vlib work

ModelSim> vmap work work
# Copying d:/applications/modelsim/win32/./modelsim.ini to modelsim.ini
# Modifying modelsim.ini
# Warning: Copied d:/applications/modelsim/win32/./modelsim.ini to modelsim.ini.
# Updated modelsim.ini.
# MODELSIM set to d:/applications/modelsim/win32/./modelsim.ini.
# The MODELSIM environment variable is used by all ModelSim products
# to find the modelsim.ini file, so this local copy will not be used.

ModelSim> |

<No Design Loaded>
```

---

**Note** You must use the ModelSim **File** menu or `vlib` command to create the library directory to ensure that the required `_info` file is created. Do not create the library with operating system commands.

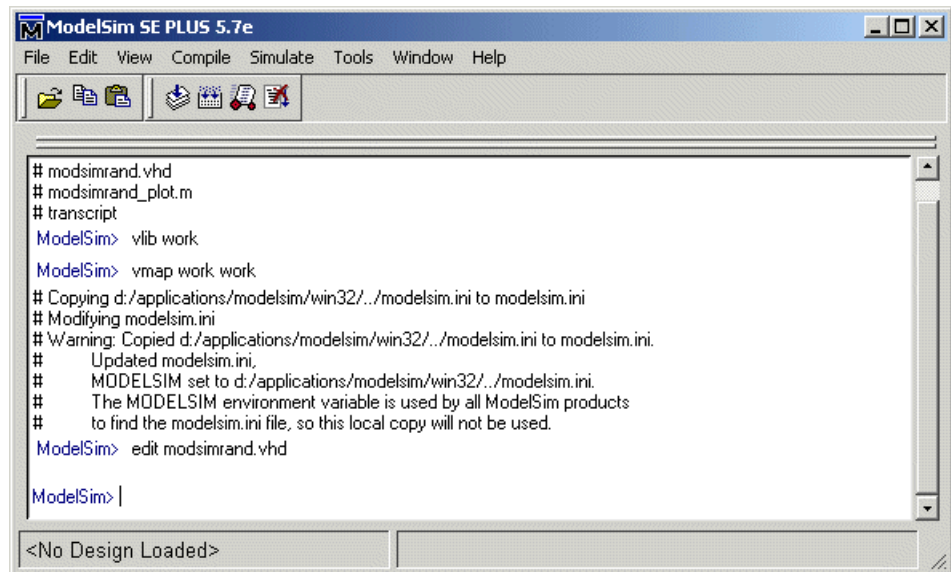
---

### Developing the VHDL Code

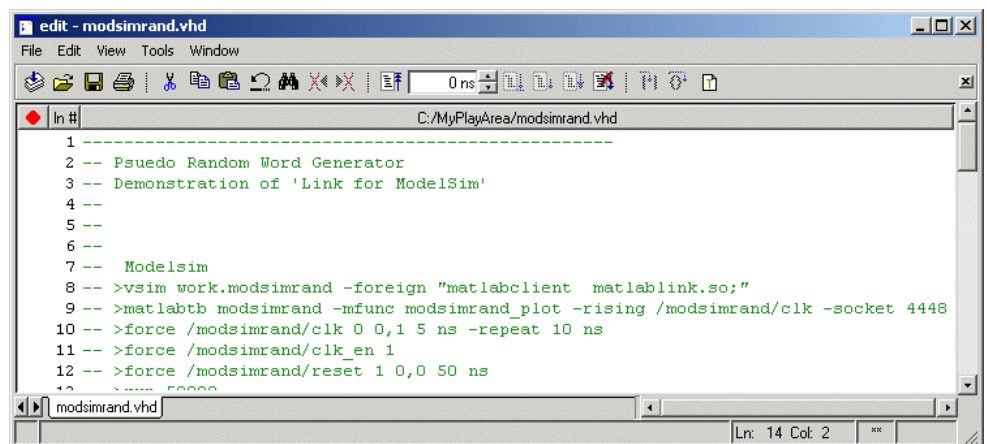
After setting up a design library, typically you would use the ModelSim Editor to create and modify your HDL code. For this tutorial, open and examine the existing file `modsimrand.vhd`. This section highlights areas of code in `modsimrand.vhd` that are of interest for a ModelSim and MATLAB test bench:

- 1 Open `modsimrand.vhd` in the edit window with the `edit` command, as follows:

```
ModelSim> edit modsimrand.vhd
```



ModelSim opens its **edit** window and displays the VHDL code for `modsimrand.vhd`.



- 2 Search for ENTITY `modsimrand`. This line defines the VHDL entity `modsimrand`:

```
ENTITY modsimrand IS
```

```
PORT (  
    clk      : IN std_logic ;  
    clk_en   : IN std_logic ;  
    reset    : IN std_logic ;  
    dout     : OUT std_logic_vector (31 DOWNT0 0);  
END modsimrand;
```

This entity will be verified in the MATLAB environment. Note the following:

- By default, the MATLAB server assumes that the name of the MATLAB function that verifies the entity in the MATLAB environment is the same as the entity name. You have the option of naming the MATLAB function explicitly. However, if you do not specify a name, the server expects the function name to match the entity name. In this example, the MATLAB function name is `modsimrand_plot` and does not match.
- The entity must be defined with a `PORT` clause that includes at least one port definition. Each port definition must specify a port mode (IN, OUT, or INOUT) and a VHDL data type that is supported by the EDA Simulator Link MQ software. For a list of the supported types, see “Coding HDL Modules for MATLAB Verification” on page 2-7.

The entity `modsimrand` in this example is defined with three input ports `clk`, `clk_en`, and `reset` of type `STD_LOGIC` and output port `dout` of type `STD_LOGIC_VECTOR`. The output port passes simulation output data out to the MATLAB function for verification. The optional input ports receive clock and reset signals from the function. Alternatively, the input ports can receive signals from ModelSim force commands.

For more information on coding port entities for use with MATLAB, see “Coding HDL Modules for MATLAB Verification” on page 2-7.

- 3** Browse through the rest of `modsimrand.vhd`. The remaining code defines a behavioral architecture for `modsimrand` that writes a randomly generated Fibonacci sequence to an output register when the clock experiences a rising edge.
- 4** Close the ModelSim **edit** window.

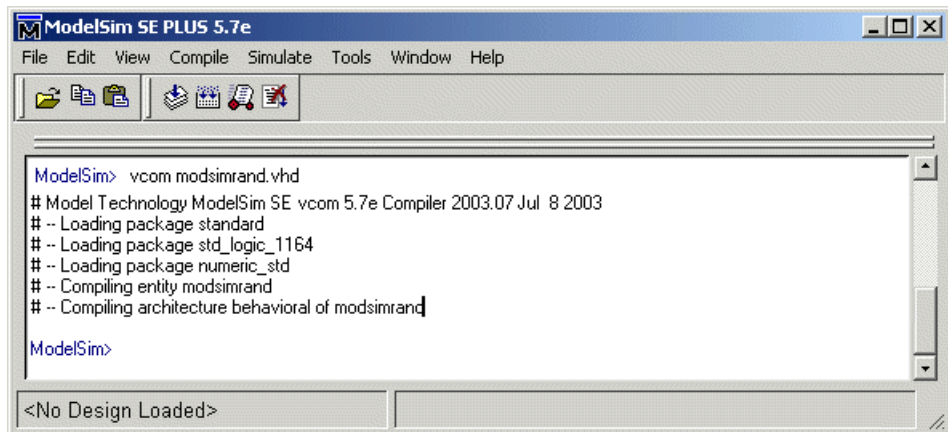


## Compiling the VHDL File

After you create or edit your VHDL source files, compile them. As part of this tutorial, compile `modsimrand.vhd`. One way of compiling the file is to click the file name in the project workspace and select **Compile > Compile All**. Another alternative is to specify `modsimrand.vhd` with the `vcom` command, as follows:

```
ModelSim> vcom modsimrand.vhd
```

If the compilation succeeds, informational messages appear in the command window and the compiler populates the work library with the compilation results.



## Loading the Simulation

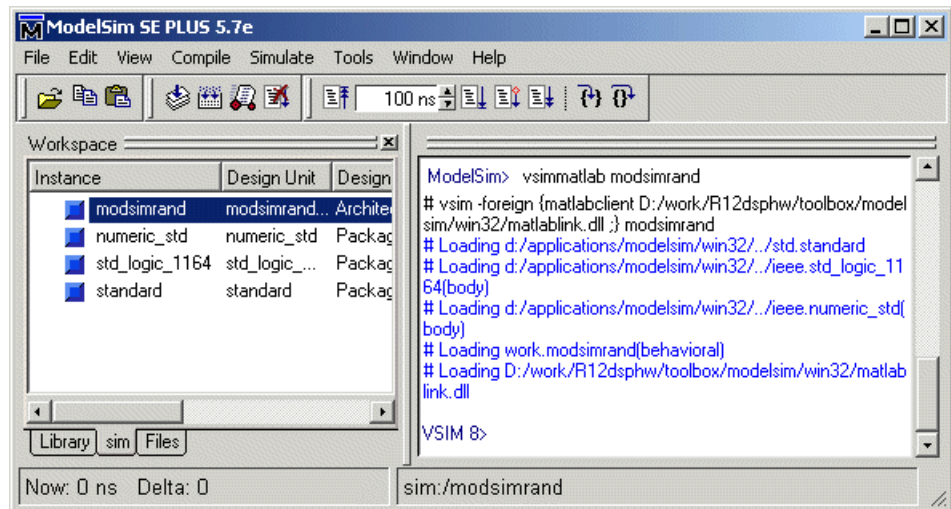
After you successfully compile the VHDL source file, you are ready to load the model for simulation. This section explains how to load an instance of entity `modsimrand` for simulation:

- 1 Load the instance of `modsimrand` for verification. To load the instance, specify the `vsimmatlab` command as follows:

```
ModelSim> vsimmatlab modsimrand
```

The `vsimmatlab` command starts the ModelSim simulator, `vsim`, specifically for use with MATLAB. You can specify `vsimmatlab` with any combination of valid ModelSim `vsim` command parameters and options.

ModelSim displays a series of messages in the command window as it loads the entity's packages and architecture.



- 2 Initialize the simulator for verifying `modsimrand` with MATLAB. You initialize ModelSim by using the `matlabtb` or `matlabtbval` ModelSim command. These commands define the communication link and a callback to a MATLAB function that executes in MATLAB on behalf of ModelSim. In addition, the `matlabtb` commands can specify parameters that control when the MATLAB function executes.

For this tutorial, enter the following `matlabtb` command:

```
VSIM n> matlabtb modsimrand -mfunc modsimrand_plot
-rising /modsimrand/clock -socket portnum
```

Keep in mind that the port number or service name that you specify with `-socket` must match the port value returned by or specified with the call to `hdldaemon` that started the MATLAB server.

Arguments in the command line specify the following:

<code>modsimrand</code>	The instance of the VHDL entity that is to be associated with a MATLAB function.
<code>-mfunc modsimrand_plot</code>	The MATLAB function to be called on behalf of HDL instance <code>modsimrand</code> .
<code>-rising /modsimrand/clock</code>	That the function <code>modsimrand_plot.m</code> be called when the signal <code>/modsimrand/clock</code> changes from '0' to '1'. Note the signal is specified in a full path name format. If you do not specify a full path name, the command applies ModelSim rules to resolve signal specifications.
<code>-socket portnum</code>	The TCP/IP socket port <code>portnum</code> to be used to establish a communication link with MATLAB.

This command links an instance of the entity `modsimrand` to the function `modsimrand_plot.m`, which executes within the context of MATLAB based on specified timing parameters. In this case, the MATLAB function is called when the signal `/modsimrand/clock` experiences a rising edge.

---

**Note** By default, the EDA Simulator Link MQ 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.

---

- 3** Initialize clock and reset input signals. You can drive simulation input signals using several mechanisms, including ModelSim force commands and an `iport` parameter (see “Coding MATLAB Link Functions” on page 2-12). For now, enter the following force commands:

```
VSIM n> force /modsimrand/clock 0 0 ns, 1 5 ns -repeat 10 ns
VSIM n> force /modsimrand/clock_en 1
VSIM n> force /modsimrand/reset 1 0, 0 50 ns
```

The first command forces the `clock` signal to value 0 at 0 nanoseconds and to 1 at 5 nanoseconds. After 10 nanoseconds, the cycle starts to repeat every 10 nanoseconds. The second and third force commands set `clock_en` to 1 and `reset` to 1 at 0 nanoseconds and to 0 at 50 nanoseconds.

The ModelSim environment is ready to run a simulation. Now you need to set up the MATLAB function.

### Developing the MATLAB Function

The EDA Simulator Link MQ software verifies HDL hardware in MATLAB as a function. Typically, at this point you would create or edit a MATLAB function that meets EDA Simulator Link MQ requirements. For this tutorial, open and examine the existing file `modsimrand_plot.m`.

`modsimrand_plot.m` is a lower-level component of the MATLAB Random Number Generator Demo. Plotting code within `modsimrand_plot.m` is not discussed in the section below. This tutorial focuses only on those parts of `modsimrand_plot.m` that are required for MATLAB to verify a VHDL model.

Perform the following steps:

- 1 Open `modsimrand_plot.m` in the MATLAB Edit/Debug window. For example:

```
edit modsimrand_plot.m
```

- 2 Look at line 1. This is where you specify the MATLAB function name and required parameters:

```
function [iport,tnext] = modsimrand_plot(oport,tnow,portinfo)
```

This function definition is significant because it represents the communication channel between MATLAB and ModelSim. When coding the function definition, consider the following:

- By default, the EDA Simulator Link MQ software assumes the function name is the same as the name of the VHDL entity that it services. However, you can name the function differently, as in this case. The name of the VHDL entity is `modsimrand` and the name of the function is `modsimrand_plot`. Because the names differ, you must explicitly specify the function name when you request service from ModelSim.
- You *must* define the function with two output parameters, `iport` and `tnext`, and three input parameters, `oport`, `tnow`, and `portinfo`.

For more information on the required MATLAB function parameters, see “Coding MATLAB Link Functions” on page 2-12.

- You can use the `iport` parameter to drive input signals instead of, or in addition to, using other signal sources, such as ModelSim force commands. Depending on your application, you might use any combination of input sources. However, keep in mind that if multiple sources drive signals to a single `iport`, a resolution function is required for handling signal contention.

- 3** Initialize the function outputs `iport` and `tnext` to empty values, as in the following code excerpt:

```
tnext = [];  
iport = struct();
```

- 4** Make note of the data types of ports defined for the entity under simulation. The EDA Simulator Link MQ software converts VHDL data types to comparable MATLAB data types and vice versa. As you develop your MATLAB function, you must know the types of the data that it receives from and needs to return to ModelSim. This tutorial includes the following port data type definitions and conversions:

- The entity defined for this tutorial consists of three input ports of type `STD_LOGIC` and an output port of type `STD_LOGIC_VECTOR`.
- Data of type `STD_LOGIC_VECTOR` consists of a column vector of characters with one bit per character.
- The interface converts scalar data of type `STD_LOGIC` to a character that matches the character literal for the corresponding enumerated type.

For more information on interface data type conversions, see “Performing Data Type Conversions” on page 2-20.

- 5** Search for `oport.dout`. This line of code shows how the data that a MATLAB function receives from ModelSim might need to be converted for use in the MATLAB environment:

```
ud.buffer(cyc) = mv12dec(oport.dout)
```

In this case, the function receives `STD_LOGIC_VECTOR` data on `oport`. The function `mv12dec` converts the bit vector to a decimal value that can be used

in arithmetic computations. “Performing Data Type Conversions” on page 2-20 provides a summary of the types of data conversions to consider when coding your own MATLAB functions.

6 Browse through the rest of `modsimrand_plot.m`.

### Running the Simulation

This section explains how to start and monitor this simulation, and rerun it, if necessary:

When you have completed as many simulation runs as desired, shut down the simulation as described in the next section.

### Running the Simulation for the First Time

Before running the simulation for the first time, you must verify the client connection. You may also want to set breakpoints for debugging.

Perform the following steps:

- 1 Open ModelSim and MATLAB windows.
- 2 In MATLAB, verify the client connection by calling `hdldaemon` with the 'status' option:

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

This function returns a message indicating a connection exists:

```
HDLDaemon socket server is running on port 4795 with 1 connection
```

---

**Note** If you attempt to run the simulation before starting the `hdldaemon` in MATLAB, you will receive the following warning:

```
#ML Warn - MATLAB server not available (yet),  
The entity 'modsimrand' will not be active
```

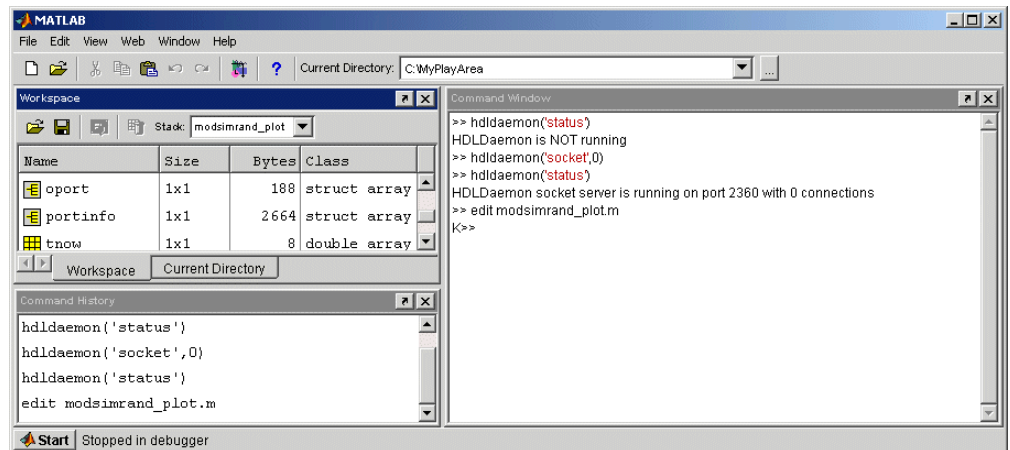
---

- 3 Open `modsimrand_plot.m` in the MATLAB Edit/Debug window.
- 4 Search for `oport.dout` and set a breakpoint at that line by clicking next to the line number. A red breakpoint marker will appear.
- 5 Return to ModelSim and enter the following command in the command window:

```
Vsim n> run 80000
```

This command instructs ModelSim to advance the simulation 80,000 time steps (80,000 nanoseconds using the default time step period). Because you previously set a breakpoint in `modsimrand_plot.m`, however, the simulation runs in MATLAB until it reaches the breakpoint.

ModelSim is now blocked and remains blocked until you explicitly unblock it. While the simulation is blocked, note that MATLAB displays the data that ModelSim passed to the MATLAB function in the **Workspace** window.



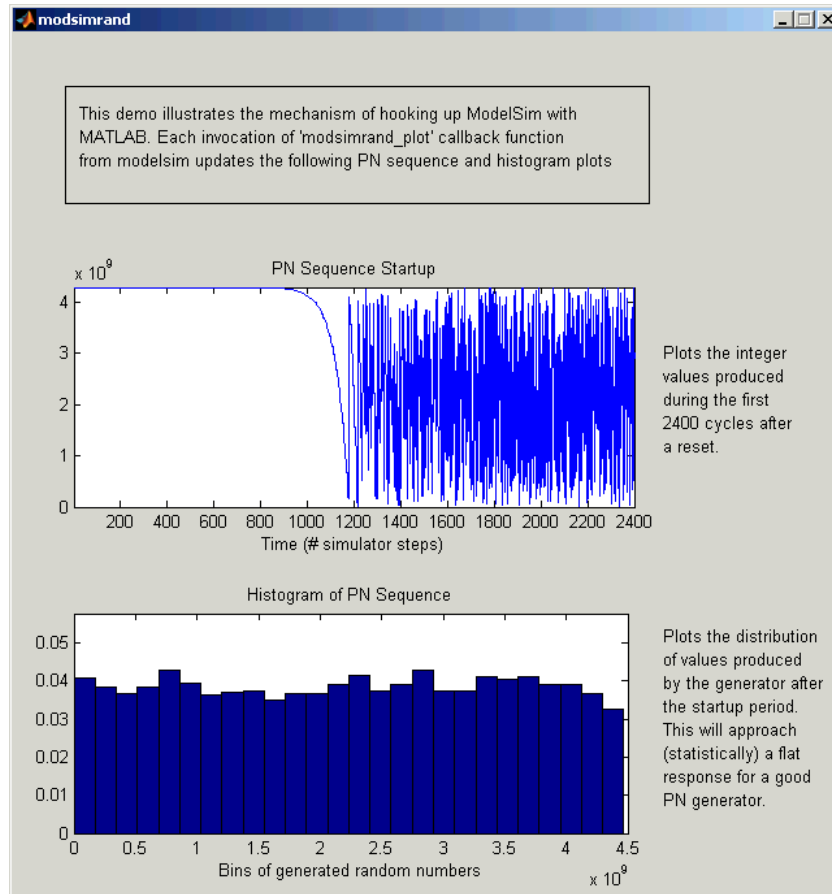
Note also that a ModelSim figure window opens. This window is used to plot data generated by the simulation. Initially, it is empty.

- 6 Examine `oport`, `portinfo`, and `tnow`. Observe that `tnow`, the current simulation time, is set to 0. Also notice that, because the simulation has reached a breakpoint during the first call to `modsimrand_plot`, the `portinfo` is visible in the MATLAB workspace.

- 7 Click **Debug > Continue** in the MATLAB Edit/Debug window. The next time the breakpoint is reached, notice that portinfo is no longer visible in the MATLAB workspace. This is because portinfo is passed in only on the first function invocation. Also note that the value of tnow advances from 0 to 5e-009.
- 8 Clear the breakpoint by clicking the red breakpoint marker.
- 9 Unblock ModelSim and continue the simulation by clicking **Debug > Continue** in the MATLAB Edit/Debug window.

The simulation runs to completion. As the simulation progresses, it plots generated data in a figure window. When the simulation completes, the figure window appears as shown below.





The simulation runs in MATLAB until it reaches the breakpoint that you just set. Continue the simulation/debugging session as desired.

## Rerunning the Simulation

If you want to run the simulation again, you must restart the simulation in ModelSim, re-initialize the clock, and reset input signals. To do this:

- 1 Close the figure window.
- 2 Restart the simulation with the following command:

```
VSIM n> restart
```

The **Restart** dialog box appears. Leave all the options enabled and click **Restart**.

---

**Note** The **Restart** button clears the simulation context established by a `matlabcp` or `matlabtb` command. Thus, after restarting ModelSim, you must reissue the previous command or issue a new command.

---

**3** Reissue the `matlabtb` command.

```
VSIM n> matlabtb modsimrand -mfunc modsimrand_plot  
-rising /modsimrand/clock -socket portnum
```

**4** Open `modsimrand_plot.m` in the MATLAB Edit/Debug window.

**5** Set a breakpoint at the same line as in the previous run.

**6** Return to ModelSim and re-enter the following commands to reinitialize clock and input signals:

```
Vsim n> force /modsimrand/clock 0 0,1 5 ns -repeat 10 ns  
Vsim n> force /modsimrand/clock_en 1  
Vsim n> force /modsimrand/reset 1 0, 0 50 ns
```

**7** Enter a command to start the simulation, for example:

```
Vsim n> run 80000
```

## Shutting Down the Simulation

This section explains how to shut down a simulation in an orderly way.

In ModelSim, perform the following steps:

- 1** Stop the simulation on the client side by selecting **Simulate > End Simulation** or entering the quit command.
- 2** Quit ModelSim.

In MATLAB, just quit the application.

To shut down the server without closing MATLAB, you have the option of calling `hdldaemon` with the `'kill'` option:

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

The following message appears, confirming that the server was shut down:

```
HDLDaemon server was shut down
```



# Linking Simulink<sup>®</sup> to ModelSim<sup>®</sup> Simulators

---

Simulink <sup>®</sup> -ModelSim <sup>®</sup> Workflow (p. 3-2)	Provides a high-level view of the steps involved in coding and running a Simulink cosimulation for use with the EDA Simulator Link <sup>™</sup> MQ software.
Introduction to Cosimulation (p. 3-5)	Provides an introduction to the process for integrating EDA Simulator Link MQ blocks into a Simulink <sup>®</sup> design.
Preparing for Cosimulation (p. 3-14)	Describes the different procedures required for HDL model cosimulation
Incorporating Hardware Designs into a Simulink <sup>®</sup> Model (p. 3-40)	Explains how to add the HDL Cosimulation block to Simulink and configure the block for your HDL module
Running Cosimulation Sessions (p. 3-70)	Describes how to run, test, and optimize your cosimulation
Simulink and ModelSim Tutorial (p. 3-72)	Guides you through the basic steps for setting up an EDA Simulator Link MQ session that uses Simulink to verify a simple VHDL model.
toVCD Block Tutorial (p. 3-90)	Guides you through the basic steps for adding a toVCD block to a Simulink model for use with cosimulation

## Simulink®-ModelSim® Workflow

The following table lists the steps necessary to cosimulate an HDL design using Simulink® software.

In MATLAB...	In ModelSim SE/PE...	In Simulink...
<p><b>1</b> Start the MATLAB® application and invoke the ModelSim® simulator (see “Starting the HDL Simulator” on page 1-23)</p>		
	<p><b>2</b> Create the HDL model.</p>	
<p><b>3</b> Compile and elaborate the HDL model using vsim.</p> <p>Load elaborated HDL model with EDA Simulator Link™ MQ libraries. See “Loading an HDL Design for Verification” on page 2-12.</p>		

In MATLAB...	In ModelSim SE/PE...	In Simulink...
		<ul style="list-style-type: none"><li data-bbox="951 331 1273 392"><b>4</b> Create a new Simulink model.</li><li data-bbox="951 423 1310 579"><b>5</b> Add an HDL Cosimulation block (see “Incorporating Hardware Designs into a Simulink® Model” on page 3-40).</li><li data-bbox="951 611 1299 706"><b>6</b> Define the block interface (see “Defining the Block Interface” on page 3-45).</li><li data-bbox="951 737 1310 829"><b>7</b> Add other Simulink blocks to complete the Simulink model.</li></ul>

In MATLAB...	In ModelSim SE/PE...	In Simulink...
	<p><b>8</b> (Optional) Set breakpoints for interactive HDL debug.</p>	
		<p><b>9</b> Run the simulation.</p> <p><b>10</b> Verify that the revised model runs as expected. If it does not, then:</p> <ul style="list-style-type: none"> <li><b>a</b> Modify the VHDL or Verilog code and simulate it in the HDL simulator.</li> <li><b>b</b> Determine whether you need to reconfigure the HDL Cosimulation block. If you do, repeat steps 7 and 10.</li> </ul> <p><b>11</b> Consider using a To VCD File block to verify cosimulation results.</p>



## Introduction to Cosimulation

### In this section...

“Creating a Hardware Model Design for Use in Simulink® Applications” on page 3-5

“The EDA Simulator Link™ MQ HDL Cosimulation Block” on page 3-8

“Communicating Between the HDL Simulator and Simulink® Software” on page 3-12

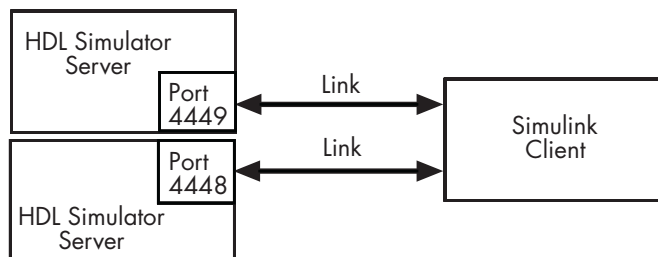
### Creating a Hardware Model Design for Use in Simulink® Applications

After you decide to include Simulink® software as part of your EDA flow, think about its role:

- Will you start by developing an HDL application using ModelSim® SE/PE, and possibly MATLAB® software, and then test the results at a system level in Simulink?
- Will you start with a system-level model in Simulink with “black box hardware components” and, after the model runs as expected, replace the black boxes with HDL Cosimulation blocks?
- What other Simulink blocksets might apply to your application? Blocksets of particular interest for EDA applications include the Communications Blockset, Signal Processing Blockset, and Simulink Fixed Point software.
- Will you set up HDL Cosimulation blocks as a subsystem in your model?
- What sample times will be used in the model? Will any sample times need to be scaled?
- Will you generate a Value Change Dump (VCD) file?

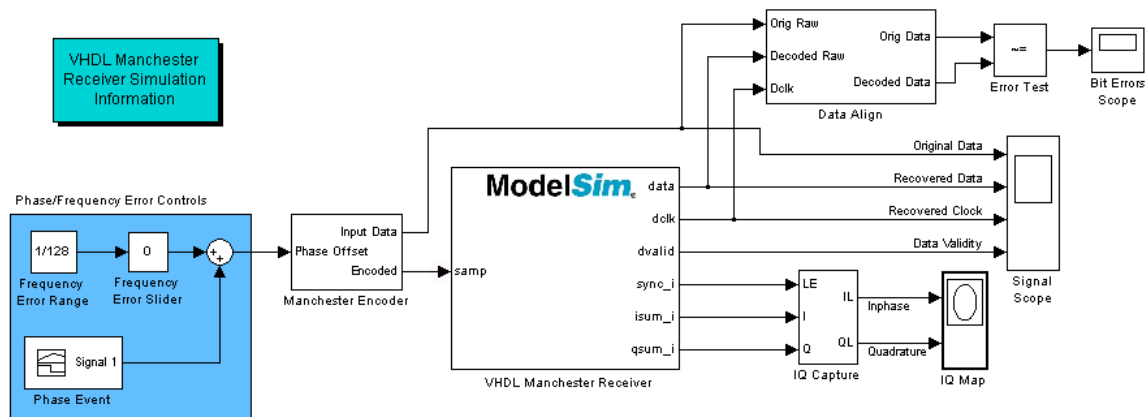
After you answer these questions, use Simulink to build your simulation environment.

As the following figure shows, multiple cosimulation blocks in a Simulink model can request the service of multiple instances of the HDL simulator, using unique TCP/IP socket ports.



When linked with Simulink, the HDL simulator functions as the server. Using the EDA Simulator Link™ MQ communications interface, an 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.

This figure shows a sample Simulink model that includes an HDL Cosimulation block.



Before running this model you must first launch ModelSim.  
You can launch ModelSim on this computer using either a shared memory link or a TCP/IP socket link.

Shared memory link:

- 1) Be sure that the 'Connection' tab of the Cosimulation block dialog is set as follows:  
     'ModelSim running on this computer' is checked  
     and 'Shared memory' is selected
- 2) Execute the following MATLAB command:  
     `vsim('tclstart',manchestercmds)`
- 3) Start the Simulink simulation.

```
vsim('tclstart',manchestercmds)
%Double-click here to launch a new ModelSim
```

ModelSim Startup Command(Shared Memory)

TCP/IP socket link:

- 1) Be sure that the 'Connection' tab of the Cosimulation block dialog is set as follows:  
     'ModelSim running on this computer' is checked  
     and 'Socket' is selected  
     'Port number or service' matches the port number used  
     in the command below.
- 2) Execute the following MATLAB command:  
     `vsim('tclstart',manchestercmds,'socketsimulink',4442)`
- 3) Start the Simulink simulation.

```
vsim('tclstart',manchestercmds,'socketsimulink',4442)
%Double-click here to launch a new ModelSim
```

ModelSim Startup Command(TCP/IP Socket)

The HDL Cosimulation block models a Manchester receiver that is coded in HDL. Other blocks and subsystems in the model include the following:

- Frequency Error Range block, Frequency Error Slider block, and Phase Event block
- Manchester encoder subsystem
- Data alignment subsystem
- Inphase/Quadrature (I/Q) capture subsystem
- Error Rate Calculation block from the Communications Blockset software

- Bit Errors block
- Data Scope block
- Discrete-Time Scatter Plot Scope block from the Communications Blockset software

For information on getting started with Simulink, see the Simulink online help or documentation.

## The EDA Simulator Link™ MQ HDL Cosimulation Block

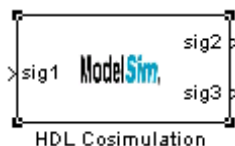
The EDA Simulator Link MQ HDL Cosimulation Block links hardware components that are concurrently simulating in the HDL simulator to the rest of a Simulink model.

Two potential use cases follow:

- A single HDL Cosimulation block fits into the framework of a larger system-oriented Simulink model.
- The Simulink model is a collection of HDL Cosimulation blocks, each representing a specific hardware component.

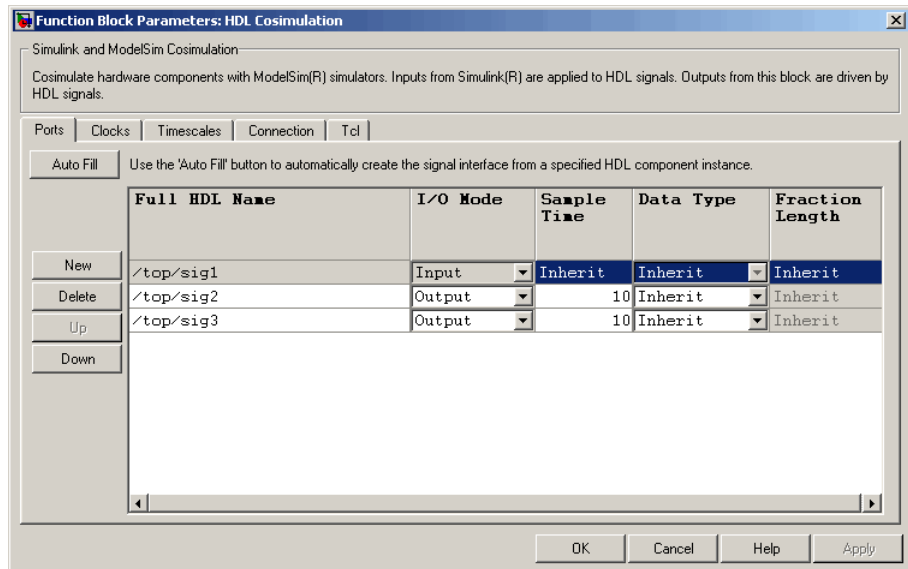
The block mask contains panels for entering port and signal information, setting communication modes, adding clocks, specifying pre- and post-simulation Tcl commands, and defining the timing relationship.

After you code one of your model's components in VHDL or Verilog and simulate it in the HDL simulator environment, you integrate the HDL representation into your Simulink model as an HDL Cosimulation block. This block, located in the Simulink Library, within the EDA Simulator Link MQ block library, is shown below.

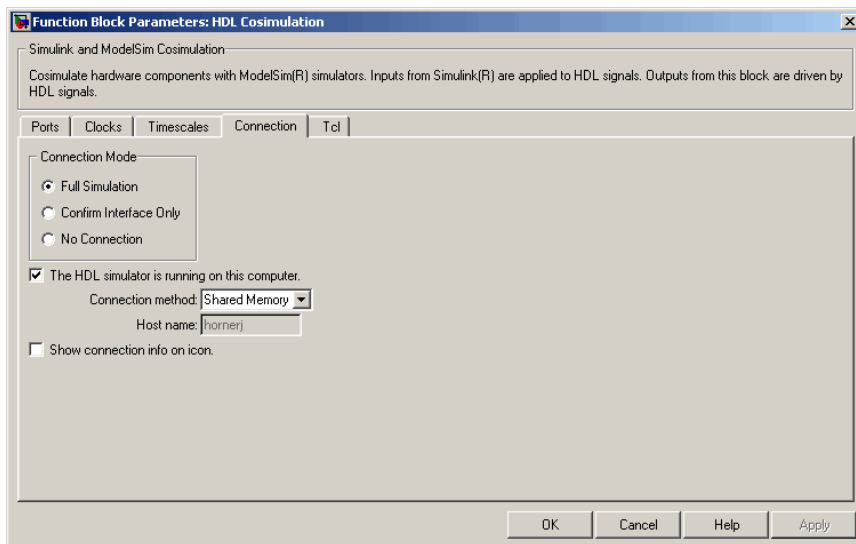


You configure an HDL Cosimulation block by specifying values for parameters in a block parameters dialog. The HDL Cosimulation block parameters dialog consists of tabbed panes that specify the following:

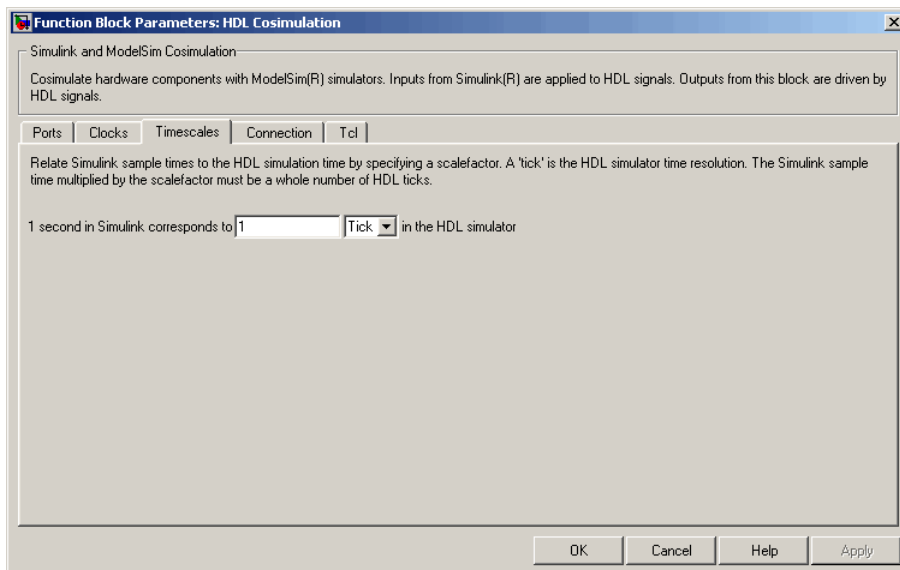
- **Ports Pane:** Block input and output ports that correspond to signals, including internal signals, of your HDL design, and an output sample time. See “Ports Pane” on page 6-3 in the Chapter 6, “EDA Simulator Link™ MQ Simulink® Block Reference”.



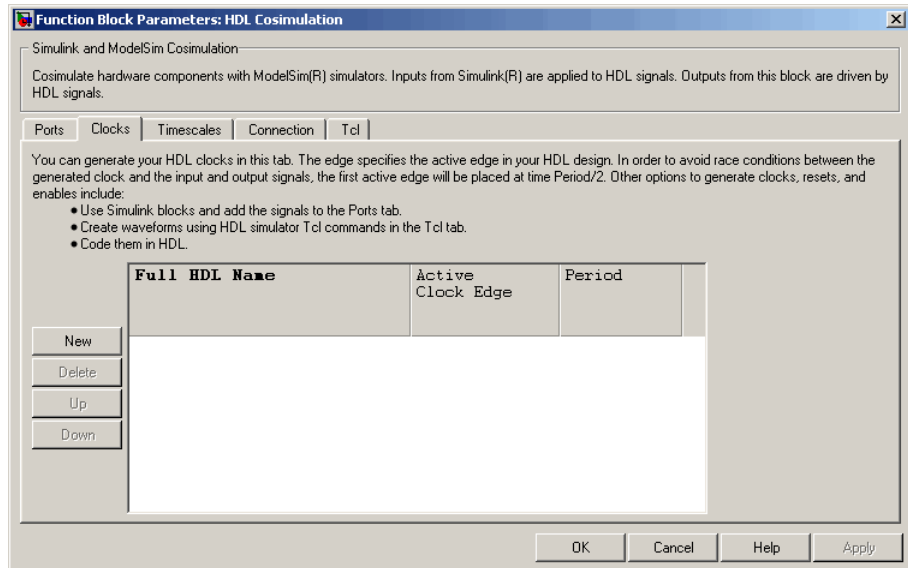
- **Connection Pane:** Type of communication and communication settings to be used for exchanging data between simulators. See “Connection Pane” on page 6-10 in the Chapter 6, “EDA Simulator Link™ MQ Simulink® Block Reference”.



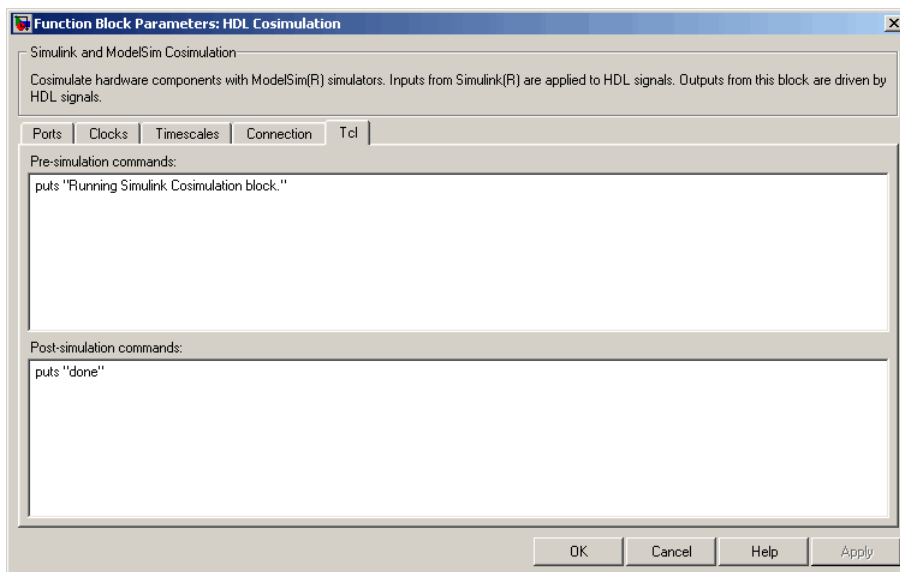
- **Timescales Pane:** Timing relationship between Simulink and the HDL simulator. See “Timescales Pane” on page 6-13 in the Chapter 6, “EDA Simulator Link™ MQ Simulink® Block Reference”.



- **Clocks Pane:** Optional rising-edge and falling-edge clocks to apply to your model. See “Clocks Pane” on page 6-15 in the Chapter 6, “EDA Simulator Link™ MQ Simulink® Block Reference”.



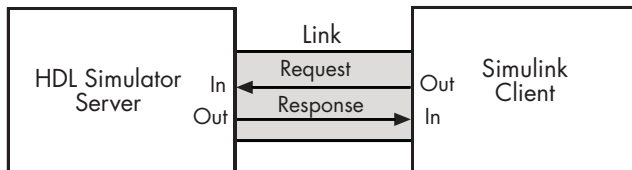
- **Tcl Pane:** Tcl commands to run before and after a simulation. See “Tcl Pane” on page 6-18 in the Chapter 6, “EDA Simulator Link™ MQ Simulink® Block Reference”.



**Note** Make sure that signals being used in cosimulation have read/write access (this is done through the HDL simulator—see product documentation for details). This rule applies to all signals on the **Ports**, **Clocks**, and **Tcl** panes.

## Communicating Between the HDL Simulator and Simulink® Software

When linked with a Simulink application, the HDL simulator functions as the server, as shown in the following figure.

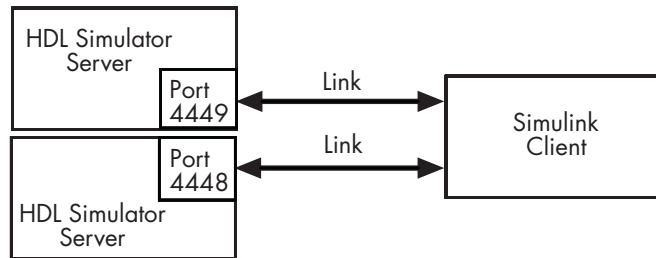


In this case, the HDL simulator responds to simulation requests it receives from cosimulation blocks in a Simulink model. You begin a cosimulation



session from Simulink. After a session is started, you can use Simulink and the HDL simulator to monitor simulation progress and results. For example, you might add signals to a wave window to monitor simulation timing diagrams.

As the following figure shows, multiple cosimulation blocks in a Simulink model can request the service of multiple instances of the HDL simulator, using unique TCP/IP socket ports.



## Preparing for Cosimulation

### In this section...

- “Overview” on page 3-14
- “How Simulink Drives Cosimulation Signals” on page 3-15
- “Representation of Simulation Time” on page 3-15
- “Handling Multirate Signals” on page 3-24
- “Handling Frame-Based Signals” on page 3-24
- “Avoiding Race Conditions in HDL Simulation” on page 3-32
- “Block Simulation Latency” on page 3-32
- “Interfacing with Continuous Time Signals” on page 3-37
- “Setting Simulink Software Configuration Parameters” on page 3-38
- “Simulink and HDL Simulator Communication Options” on page 3-39
- “Starting the HDL Simulator” on page 3-39

### Overview

The EDA Simulator Link™ MQ HDL Cosimulation block serves as a bridge between the Simulink® and the HDL simulator domains. The block represents an HDL component model within the Simulink software. Using the block, Simulink software writes (drives) signals to and reads signals from the HDL model under simulation in the ModelSim® simulator. Signal exchange between the two domains occurs at regularly scheduled time steps defined by the Simulink sample time.

As you develop an EDA Simulator Link MQ cosimulation application, you should be familiar with how signal values are handled across the simulation domains with respect to the following cases:

- How Simulink drives cosimulation signals
- Representation of simulation time
- Handling multirate signals
- Handling Frame-based signals

- Avoiding race conditions ()
- Block simulation latency
- Interfacing with continuous time signals
- Setting Simulink configuration parameters
- Setting the communication link
- Starting the HDL simulator

## How Simulink Drives Cosimulation Signals

Although you can bind the output ports of an HDL Cosimulation block to any signal in an HDL model hierarchy, you must use some caution when connecting signals to input ports. Ensure that the signal you are binding to does not have other drivers. If it does, use resolved logic types; otherwise you may get unpredictable results.

If you need to use a signal that has multiple drivers and it is resolved (for example, it is of VHDL type `STD_LOGIC`), Simulink applies the resolution function at each time step defined by the signal's Simulink sample rate. Depending on the other drivers, the Simulink value may or may not get applied. Furthermore, Simulink has no control over signal changes that occur between its sample times.

---

**Note** Make sure that signals being used in cosimulation have read/write access (this is done through the HDL simulator—see product documentation for details). This rule applies to all signals on the **Ports**, **Clocks**, and **Tcl** panes.

---

## Representation of Simulation Time

The representation of simulation time differs significantly between the HDL simulator and Simulink.

In ModelSim SE/PE, the unit of simulation time is referred to as a *tick*. The duration of a tick is defined by the HDL simulator *resolution limit*. The default resolution limit is 1 ns.

To determine current ModelSim resolution limit, enter `echo $resolution` or `report simulator state` at the ModelSim prompt. You can override the default resolution limit by specifying the `-t` option on the ModelSim command line, or by selecting a different **Simulator Resolution** in the ModelSim **Simulate** dialog box. Available resolutions in ModelSim are 1x, 10x, or 100x in units of fs, ps, ns, us, ms, or sec. See the ModelSim documentation for further information.

Simulink maintains simulation time as a double-precision value scaled to seconds. This representation accommodates modeling of both continuous and discrete systems.

The relationship between Simulink and the HDL simulator timing affects the following aspects of simulation:

- Total simulation time
- Input port sample times
- Output port sample times
- Clock periods

During a simulation run, Simulink communicates the current simulation time to the HDL simulator at each intermediate step. (An intermediate step corresponds to a Simulink sample time hit. Upon each intermediate step, new values are applied at input ports, or output ports are modified.) To bring the HDL simulator up-to-date with Simulink during cosimulation, sampled Simulink time must be converted to HDL simulator time (ticks) and the HDL simulator must run for the computed number of ticks.

---

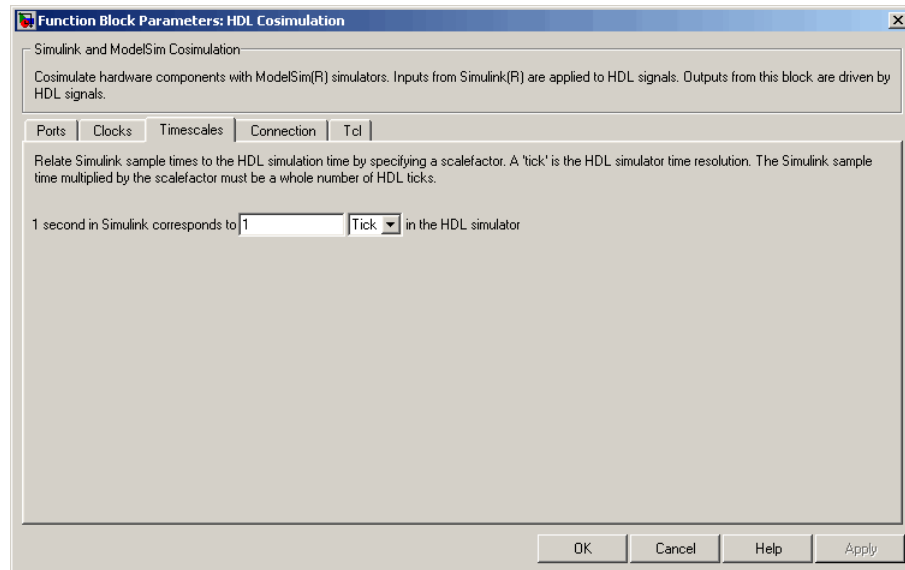
**Caution** If you specify a Simulink sample time that cannot be expressed as a whole number of HDL ticks, you will get an error.

---

The EDA Simulator Link MQ cosimulation interface provides controls that let you configure the timing relationship between the HDL simulator and Simulink and avoid timing errors caused by differences in timing representation.

## Defining the Simulink and HDL Simulator Timing Relationship

The **Timescales** pane of the HDL Cosimulation block parameters dialog lets you choose an optimal timing relationship between Simulink and the HDL simulator. The figure below shows the default settings of the **Timescales** pane.



The **Timescales** pane defines 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:

- In *relative* terms (i.e., as some number of HDL simulator ticks). In this case, the cosimulation is said to operate in *relative timing mode*. Relative timing mode is the default.
- In *absolute* units (such as milliseconds or nanoseconds). In this case, the cosimulation is said to operate in *absolute timing mode*.

The following sections discuss these two timing modes.

## Relative Timing Mode

Relative timing mode defines the following one-to-one correspondence between simulation time in Simulink and the HDL simulator:

- *One second* in Simulink corresponds to *N ticks* in the HDL simulator, where N is a scale factor.

This correspondence holds regardless of the HDL simulator timing resolution.

The following pseudocode shows how Simulink time units are quantized to HDL simulator ticks:

$$\text{InTicks} = N * \text{tInSecs}$$

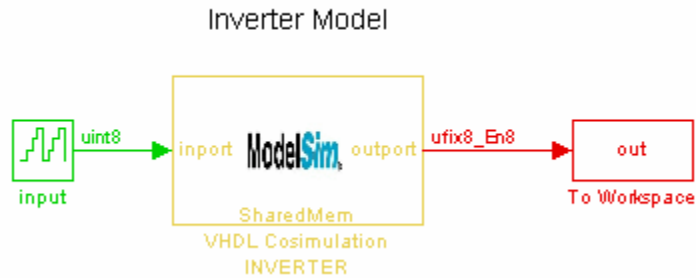
where *InTicks* is the HDL simulator time in ticks, *tInSecs* is the Simulink time in seconds, and N is a scale factor.

**Operation of Relative Timing Mode.** By default, the HDL Cosimulation block is configured for relative mode, with a scale factor of 1. Thus, 1 Simulink second corresponds to 1 tick in the HDL simulator. In the default case:

- If the total simulation time in Simulink is specified as N seconds, then the ModelSim SE/PE HDL simulation will run for exactly N ticks (i.e., N ns at the default resolution limit).
- Similarly, if Simulink computes the sample time of an HDL Cosimulation block input port as *T<sub>si</sub>* seconds, new values will be deposited on the HDL input port at exact multiples of *T<sub>si</sub>* ticks. If an output port has an explicitly specified sample time of *T<sub>so</sub>* seconds, values will be read from the HDL simulator at multiples of *T<sub>so</sub>* ticks.

## Relative Timing Mode Example

To understand how relative timing mode operates, review cosimulation results from the following example model.



The model contains an HDL Cosimulation block (labeled VHDL Cosimulation INVERTER) simulating an 8-bit inverter that is enabled by an explicit clock. The inverter has a single input and a single output. The VHDL code for the inverter is listed below:

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY inverter IS PORT (

    inport : IN  std_logic_vector := "11111111";
    outport: OUT std_logic_vector := "00000000";
    clk:IN  std_logic
);
END inverter;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

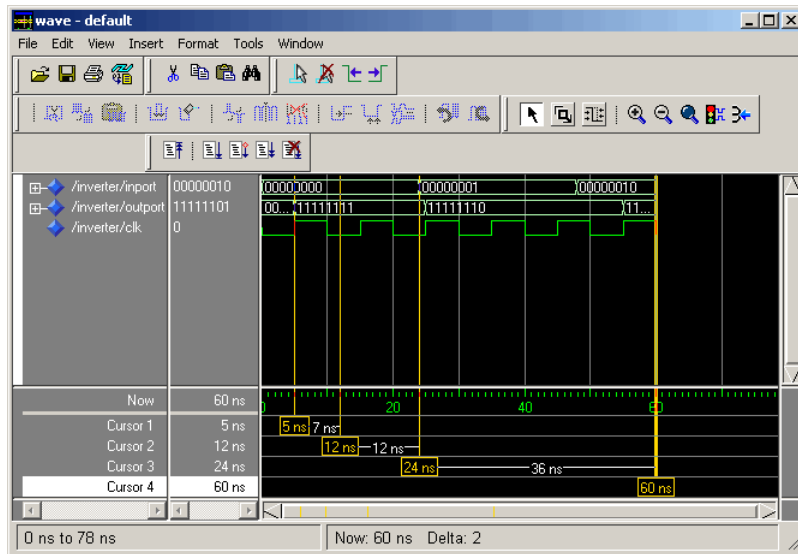
ARCHITECTURE behavioral OF inverter IS
BEGIN
    PROCESS(clk)
    BEGIN
        IF (clk'EVENT AND clk = '1') THEN
            output <= NOT inport;
        END IF;
    END PROCESS;
END behavioral;

```

A cosimulation of this model might have the following settings:

- Simulation parameters in Simulink:
  - **Timescales** parameters: default (relative timing with a scale factor of 1)
  - Total simulation time: 60 s
  - Input port (/inverter/inport) sample time: 24 s
  - Output port (/inverter/outport ) sample time: 12 s
  - Clock (inverter/clock) period: 10 s
- ModelSim resolution limit: 1 ns

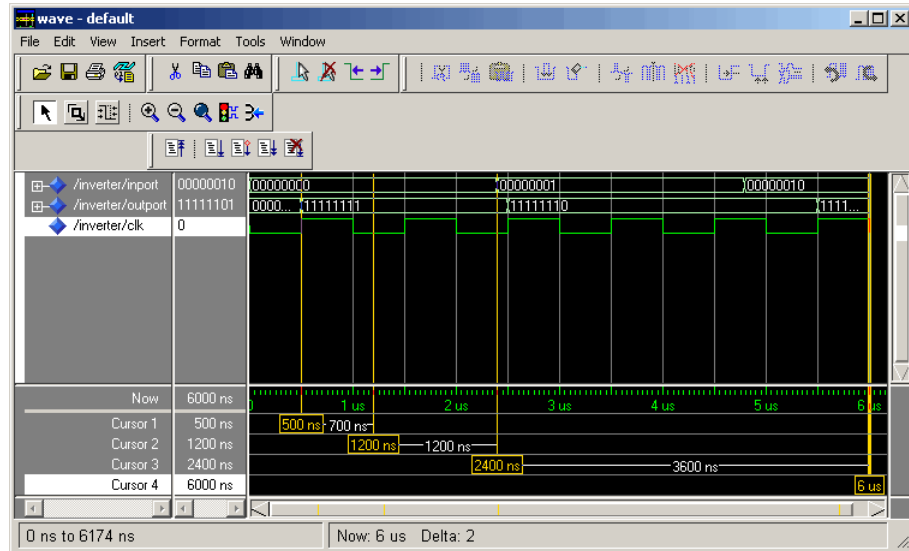
The figure below shows the ModelSim **wave** window after a cosimulation run of the example Simulink model for 60 ns. The **wave** window shows that ModelSim simulated for 60 ticks (60 ns). The inputs change at multiples of 24 ns and the outputs are read from ModelSim at multiples of 12 ns. The clock is driven low and high at intervals of 5 ns.



Now consider a cosimulation of the same model, this time configured with a scale factor of 100 in the **Timescales** pane.



The ModelSim **wave** window below shows that Simulink port and clock times were scaled by a factor of 100 during simulation. ModelSim simulated for 6 microseconds ( $60 * 100$  ns). The inputs change at multiples of  $24 * 100$  ns and outputs are read from ModelSim at multiples of  $12 * 100$  ns. The clock is driven low and high at intervals of 500 ns.



## Absolute Timing Mode

Absolute timing mode lets you define the timing relationship between Simulink and the HDL simulator in terms of absolute time units and a scale factor:

- *One second* in Simulink corresponds to  $(N * Tu)$  seconds in the HDL simulator, where  $Tu$  is an absolute time unit (e.g., ms, ns, etc.) and  $N$  is a scale factor.

To configure the **Timescales** parameters for absolute timing mode, you select a unit of absolute time, rather than Tick.

In absolute timing mode, all sample times and clock periods in Simulink are quantized to HDL simulator ticks. The following pseudocode illustrates the conversion:

$$tInTicks = tInSecs * (tScale / tRL)$$

where:

- `tInTicks` is the HDL simulator time in ticks.
- `tInSecs` is the Simulink time in seconds.
- `tScale` is the timescale setting (unit and scale factor) chosen in the **Timescales** pane of the HDL Cosimulation block.
- `tRL` is the HDL simulator resolution limit.

For example, given a **Timescales** pane setting of 1 s and an HDL simulator resolution limit of 1 ns, an output port sample time of 12 ns would be converted to ticks as follows:

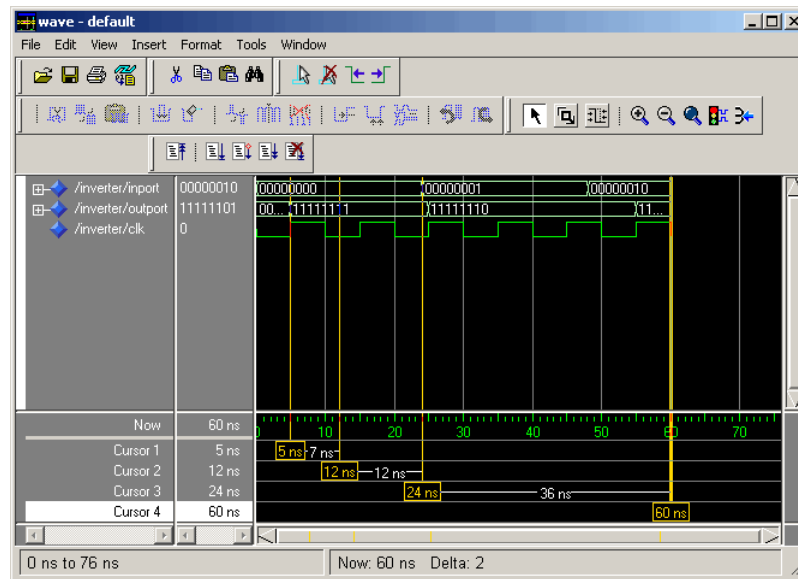
$$tInTicks = 12ns * (1s / 1ns) = 12$$

**Operation of Absolute Timing Mode.** To understand the operation of absolute timing mode, we will again consider the example model discussed in “Operation of Relative Timing Mode” on page 3-18. Suppose that the model is reconfigured as follows:

- Simulation parameters in Simulink:
  - **Timescale** parameters: 1 s of Simulink time corresponds to 1 s of HDL simulator time.
  - Total simulation time: 60e-9 s (60ns)
  - Input port (/inverter/inport) sample time: 24e-9 s (24 ns)
  - Output port (/inverter/outport) sample time: 12e-9 s (12 ns)
  - Clock (inverter/clock) period: 10e-9 s (10 ns)
- HDL simulator resolution limit: 1 ns

Given these simulation parameters, Simulink will cosimulate with the HDL simulator for 60 ns. Inputs are sampled at intervals of 24 ns and outputs are updated at intervals of 12 ns. Clocks are driven at intervals of 10 ns.

The figure below shows the ModelSim **wave** window after a cosimulation run.



### Timing Mode Usage Restrictions

The following restrictions apply to the use of absolute and relative timing modes:

- When multiple HDL Cosimulation blocks in a model are communicating with a single instance of the HDL simulator, all HDL Cosimulation blocks must have the same **Timescales** pane settings.
- If you change the **Timescales** pane settings in an HDL Cosimulation block between consecutive cosimulation runs, you must restart the simulation in the HDL simulator.

### Setting HDL Cosimulation Port Sample Times

In general, Simulink handles the sample time for the ports of an HDL Cosimulation block as follows:

- If an input port is connected to a signal that has an explicit sample time, based on forward propagation, Simulink applies that rate to that input port.

- If an input port is connected to a signal that *does not have* an explicit sample time, Simulink assigns a sample time that is equal to the least common multiple (LCM) of all identified input port sample times for the model.
- After Simulink sets the input port sample periods, it applies user-specified output sample times to all output ports. Sample times must be explicitly defined for all output ports.

If you are developing a model for cosimulation in *relative* timing mode, consider the following sample time guideline:

- Specify the output sample time for an HDL Cosimulation block as an integer multiple of the resolution limit defined in the HDL simulator. Use the HDL simulator command `report simulator state` to check the resolution limit of the loaded model. If the HDL simulator resolution limit is 1 ns and you specify a block's output sample time as 20, Simulink interacts with the HDL simulator every 20 ns.

### Handling Multirate Signals

EDA Simulator Link MQ software supports the use of multirate signals, signals that are sampled or updated at different rates, in a single HDL Cosimulation block. An HDL Cosimulation block exchanges data for each signal at the Simulink sample rate for that signal. For input signals, an HDL Cosimulation block accepts and honors all signal rates.

The HDL Cosimulation block also lets you specify an independent sample time for each output port. You must explicitly set the sample time for each output port, or accept the default. This lets you control the rate at which Simulink updates an output port by reading the corresponding signal from the HDL simulator.

### Handling Frame-Based Signals

This section discusses how to improve the performance of your cosimulation by using frame-based signals. An example is provided.

- “Overview” on page 3-25
- “Using Frame-Based Processing” on page 3-25
- “Frame-Based Cosimulation Example” on page 3-26

## Overview

The HDL Cosimulation block supports processing of single-channel frame-based signals.

A *frame* of data is a collection of sequential samples from a single channel or multiple channels. One frame of a single-channel signal is represented by a M-by-1 column vector. A signal is *frame-based* if it is propagated through a model one frame at a time.

Frame-based processing requires the Signal Processing Blockset software. Source blocks from the Signal Processing Sources library let you specify a frame-based signal by setting the **Samples per frame** block parameter. Most other signal processing blocks preserve the frame status of an input signal. You can use the Buffer block to buffer a sequence of samples into frames.

Frame-based processing can improve the computational time of your Simulink models, because multiple samples can be processed at once. Use of frame-based signals also lets you simulate the behavior of frame-based systems more accurately.

See “Working with Signals” in the Signal Processing Blockset documentation for detailed information about frame-based processing.

## Using Frame-Based Processing

You do not need to configure the HDL Cosimulation block in any special way for frame-based processing. To use frame-based processing in a cosimulation, connect one or more single-channel frame-based signals to the input port(s) of the HDL Cosimulation block. All such signals must meet the requirements described in “Frame-Based Processing Requirements and Restrictions” on page 3-26. The HDL Cosimulation block automatically configures its output(s) for frame-based operation at the appropriate frame size.

Note that use of frame-based signals affects only the Simulink side of the cosimulation. The behavior of the HDL code under simulation in the HDL simulator does not change in any way. Simulink assumes that HDL simulator processing is sample-based. Samples acquired from the HDL simulator are assembled into frames as required by Simulink. Conversely, output data framed by Simulink is transmitted to the HDL simulator in frames, which are unpacked and processed by the HDL simulator one sample at a time.

**Frame-Based Processing Requirements and Restrictions.** Observe the following restrictions and requirements when connecting frame-based signals in to an HDL Cosimulation block:

- Connection of mixed frame-based and sample-based signals to the same HDL Cosimulation block is not supported.
- Only single-channel frame-based signals can be connected to the HDL Cosimulation block. Use of multichannel (matrix) frame-based signals is not supported in this release.
- All frame-based signals connected to the HDL Cosimulation block must have the same frame size.

Frame-based processing in the Simulink model is transparent to the operation of the HDL model under simulation in the HDL simulator. The HDL model is presumed to be sample-based. The following constraint also applies to the HDL model under simulation in the HDL simulator:

- VHDL signals should be specified as scalars, not vectors or arrays (with the exception of bit vectors, as VHDL and Verilog bit vectors are converted to the appropriately sized fixed-point scalar data type by the HDL Cosimulation block).

### Frame-Based Cosimulation Example

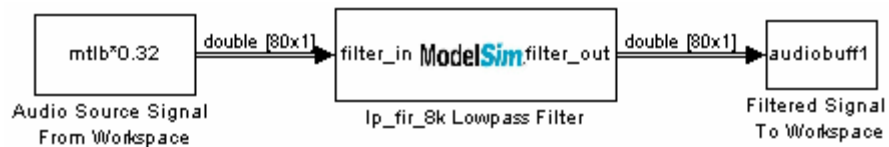
This example shows the use of the HDL Cosimulation block to cosimulate a VHDL implementation of a simple lowpass filter. In the example, you will compare the performance of the simulation using frame-based and sample-based signals.

The example files are:

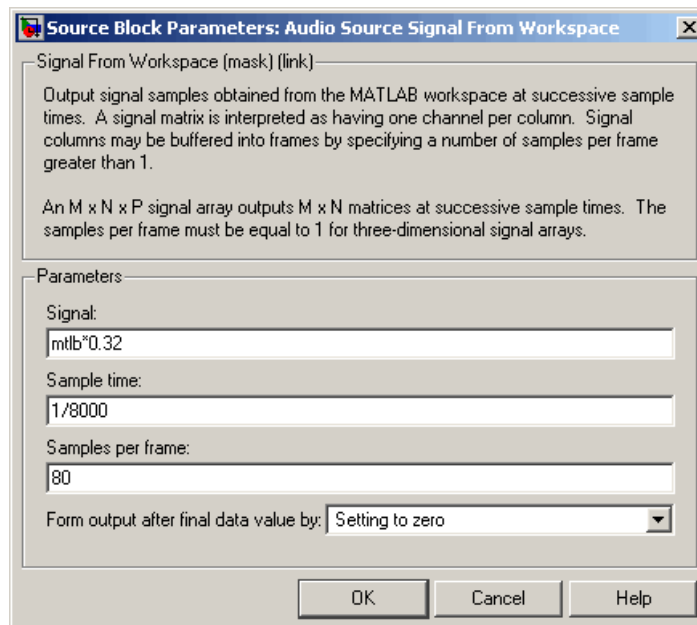
- `matlabroot\toolbox\modelsim\modelsimdemos\frame_filter_cosim.mdl`: the example model.
- `matlabroot\toolbox\modelsim\modelsimdemos\VHDL\frame_demos\lp_fir_8k.vhd`: VHDL code for the filter to be cosimulated. The filter was designed with FDATool and the code was generated by the Filter Design HDL Coder.

The example uses the data file `matlabroot\toolbox\signal\signal\mtlb.mat` as an input signal. This file contains a speech signal. The sample data is of data type double, sampled at a rate of 8 kHz.

The figure below shows the `frame_filter_cosim.mdl` model.



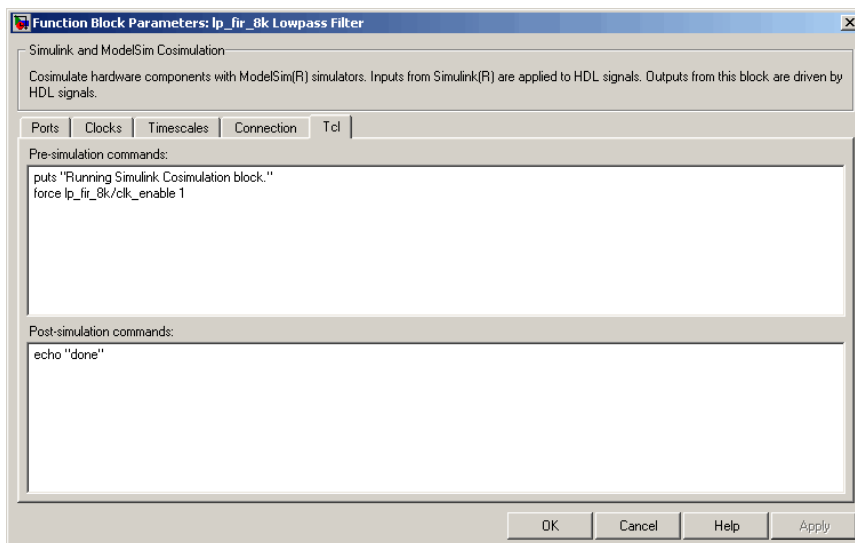
The Audio Source Signal From Workspace block provides an input signal from the workspace variable `mtlb`. The block is configured for an 8 kHz sample rate, with a frame size of 80, as shown in this figure.



The sample rate and frame size of the input signal propagate throughout the model.

The VHDL code file `lp_fir_8k.vhd` implements a simple lowpass FIR filter with a cutoff frequency of 1500 Hz. The HDL Cosimulation block simulates this HDL module. The HDL Cosimulation block ports and clock signal are configured to match the corresponding signals on the VHDL entity.

Note that for the ModelSim simulation to execute correctly, the `clk_enable` signal of the `lp_fir_8k` entity must be forced high. The signal is forced by a pre-simulation command transmitted by the HDL Cosimulation block. The command has been entered into the **Tcl** pane of the HDL Cosimulation block, as shown in the figure below.



Output data from the HDL Cosimulation block is returned the workspace variable `audiobuff1` via the Filtered Signal To Workspace block.

To run the cosimulation, perform the following steps:

- 1 Start MATLAB and make it your active window.
- 2 Set up and change to a writable working directory that is outside the context of your MATLAB installation directory.



**3** Add the demo directory  
(*matlabroot*\toolbox\modelsim\modelsimdemos\frame\_cosim) to  
the MATLAB path.

**4** Copy the demo VHDL file `lp_fir_8k.vhd` to your working directory.

**5** Open the example model.

```
open frame_filter_cosim.mdl
```

**6** Load the source speech signal, which will be filtered, into the MATLAB  
workspace.

```
load mtlb
```

If you have a compatible sound card, you can play back the source signal by  
typing the following commands at the MATLAB command prompt:

```
a = audioplayer(mtlb,8000);  
play(a);
```

**7** Start ModelSim by typing the following command at the MATLAB  
command prompt:

```
vsim
```

The ModelSim window should now be active. If not, start it.

**8** At the ModelSim prompt, create a design library, and compile the VHDL  
filter code from the source file `lp_fir_8k.vhd`, by typing the following  
commands:

```
vlib work  
vmap work work  
vcom lp_fir_8k.vhd
```

**9** The lowpass filter to be simulated is defined as the entity `lp_fir_8k`. At the  
ModelSim prompt, load the instantiated entity `lp_fir_8k` for cosimulation:

```
vsimulink lp_fir_8k
```

ModelSim is now set up for cosimulation.

- 10 Start MATLAB. Run a simulation and measure elapsed time as follows:

```
t = clock; sim(gcs); etime(clock,t)
```

```
ans =
```

```
2.7190
```

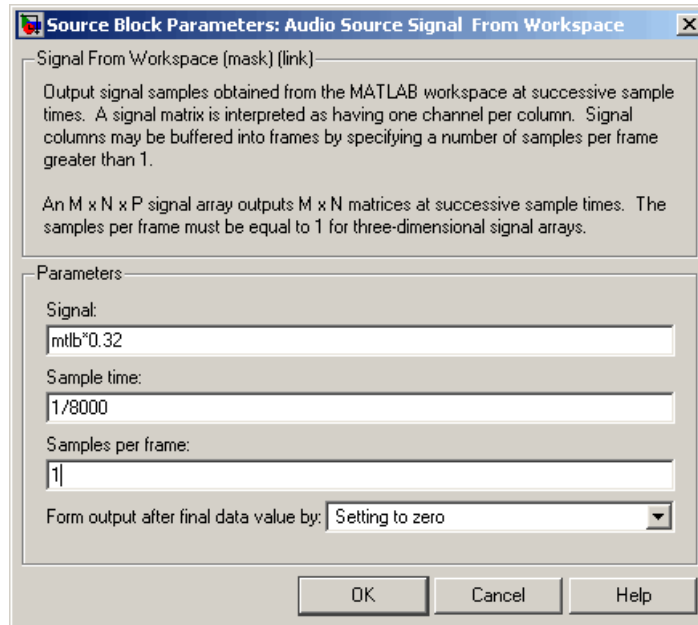
The timing above is typical for a run of this model given a simulation **Stop time** of 1 second and a frame size of 80 samples. Timings are system-dependent and will vary slightly from one simulation run to the next.

Take note of the timing you obtained. For the next simulation run, you will change the model to sample-based operation and obtain a comparative timing.

- 11 The filtered audio signal returned from ModelSim is stored in the workspace variable `audiobuff1`. If you have a compatible sound card, you can play back the filtered signal to hear the effect of the lowpass filter. Play the signal by typing the following commands at the MATLAB command prompt:

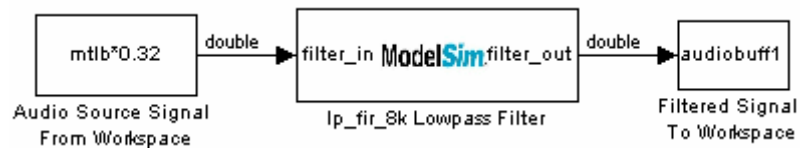
```
b = audioplayer(audiobuff1,8000);  
play(b);
```

- 12 Open the block parameters dialog of the Audio Source Signal From Workspace block and set the **Samples per frame** property to 1, as shown in this figure.



- 13** Close the dialog and select the Simulink window. Select **Update diagram** from the **Edit** menu.

The block diagram now indicates that the source signal (and all signals inheriting from it) is a scalar, as shown in the following figure.



- 14** Start ModelSim. At the ModelSim prompt, type

```
restart
```

- 15** Start MATLAB. Run a simulation and measure elapsed time as follows:

```
t = clock; sim(gcs); etime(clock,t)
```

```
ans =
```

```
3.8440
```

Observe that the elapsed time has increased significantly with a sample-based input signal. The timing above is typical for a sample-based run of this model given a simulation **Stop time** of 1 second. Timings are system-dependent and will vary slightly from one simulation run to the next.

- 16 Close down the simulation in an orderly way. In ModelSim, stop the simulation by selecting **Simulate > End Simulation**, and quit ModelSim. Then close the Simulink model window.

## Avoiding Race Conditions in HDL Simulation

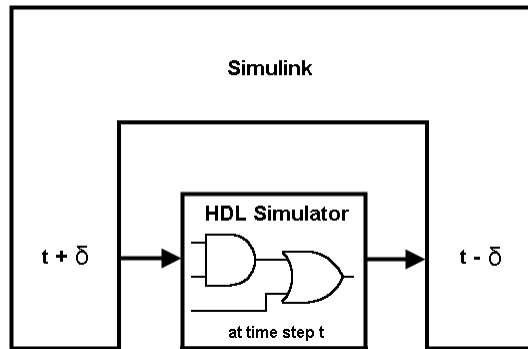
In ModelSim SE/PE, it is not possible to guarantee the order in which clock signals (rising-edge or falling-edge) defined in the HDL Cosimulation block are applied, relative to the data inputs driven by these clocks. Therefore, if care is not taken to ensure the relationship between the data and active edges of the clock, race conditions could create non-deterministic cosimulation results.

For more on race conditions in hardware simulators, see Appendix E, “Race Conditions in HDL Simulators”.

## Block Simulation Latency

Simulink and the EDA Simulator Link MQ Cosimulation blocks supplement the hardware simulator environment, rather than operate as part of it. During cosimulation, Simulink does not participate in the HDL simulator delta-time iteration. From the Simulink perspective, all signal drives (reads) occur during a single delta-time cycle. For this reason, and due to fundamental differences between the HDL simulator and Simulink with regard to use and treatment of simulation time, some degree of latency is introduced when you use EDA Simulator Link MQ Cosimulation blocks. The latency is a time lag that occurs between when Simulink begins the deposit of a signal and when the effect of the deposit is visible on cosimulation block output.

As the following figure shows, Simulink cosimulation block input affects signal values just after the current HDL simulator time step ( $t+\delta$ ) and block output reflects signal values just before the current HDL simulator step time ( $t-\delta$ ).



Regardless of whether your HDL code is specified with latency, the cosimulation block has a minimum latency that is equivalent to the cosimulation block's output sample time. For large sample times, the delay can appear to be quite long, but this is an artifact of the cosimulation block, which exchanges data with the HDL simulator at the block's output sample time only. This may be reasonable for a cosimulation block that models a device that operates on a clock edge only, such as a register-based device.

For cosimulation blocks that model combinatorial circuits, you may want to experiment with a faster sample frequency for output ports. For cosimulation blocks that model combinatorial circuits, you may want to experiment with a faster sampling frequency for output ports in order to reduce this latency.

### Block Latency Example

To visualize cosimulation block latency, consider the following VHDL code and Simulink model. The VHDL code represents an XOR gate:

```
-- edgedet.vhd

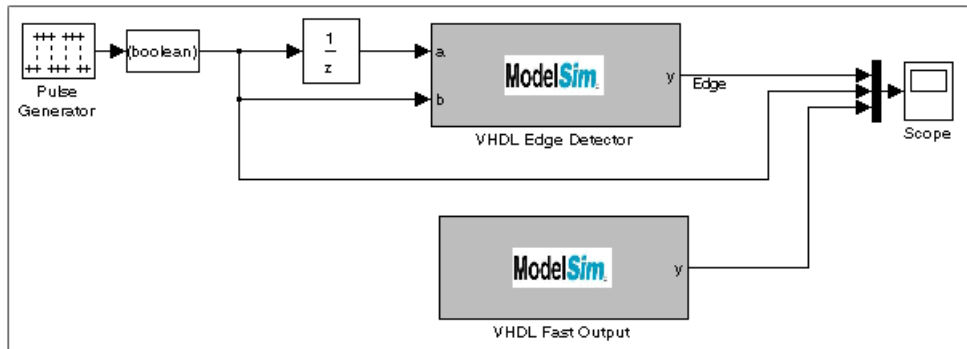
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
ENTITY edgedet IS
END edgedet;

ARCHITECTURE behavioral OF edgedet IS
SIGNAL a : std_logic;
SIGNAL b : std_logic;
```

```

SIGNAL y : std_logic;
BEGIN
  y <= a XOR b;
END behavioral;

```



In the Simulink model, the cosimulation block VHDL Edge Detector contains an XOR circuit. The second cosimulation block, VHDL Fast Output, simply reads the same XOR output. The first block is driven by a signal generated by the Pulse Generator block. The Data Type Conversion block converts the signal to a boolean value. The signal is then treated three different ways:

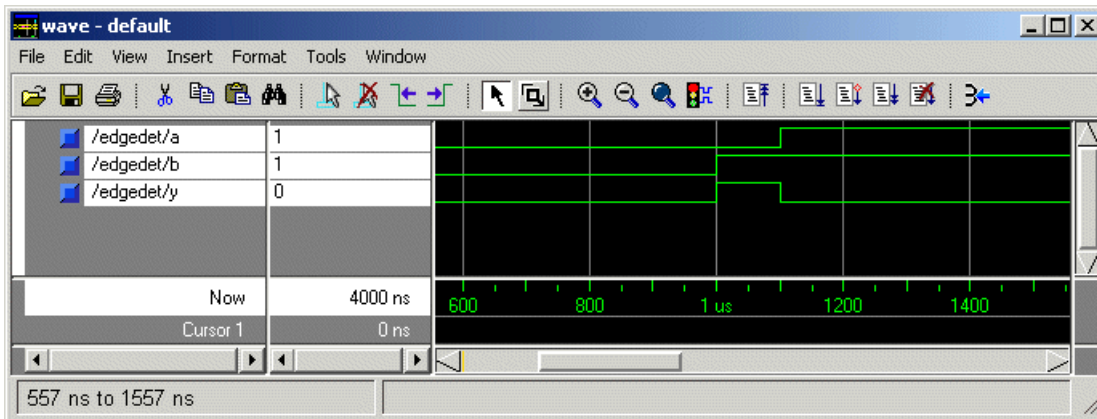
- A Unit Delay block applies a sample and hold to the signal and drives block input port a. The delay is equal to one period of the signal's Simulink sample time. When the delay is applied to the XOR, the pulse equals the period specified by the delay block after any edges.
- The signal without a delay drives block input port b.
- The third signal bypasses the cosimulation block and goes directly to the Scope block for display.

The second cosimulation block, VHDL Fast Output, is a source block that reads the output of the XOR circuit and passes it on to the Scope block for display.

Now, assume that ModelSim is set up with a resolution limit of 100 ns and an iteration limit of 5000, and that the sample times for the blocks in the Simulink model are as follows:

Block	Sample Time	Value
Pulse Generator	Sample time	100
Data Type Conversion block	Sample time	Inherited from Pulse Generator block
Unit Delay block	Sample time	Inherited from Data Type Conversion block
HDL Cosimulation block—Edge Detector	Input sample time	Inherited from Unit Delay block
	Output sample time	100
HDL Cosimulation block—Fast Output (source)	Output sample time	100

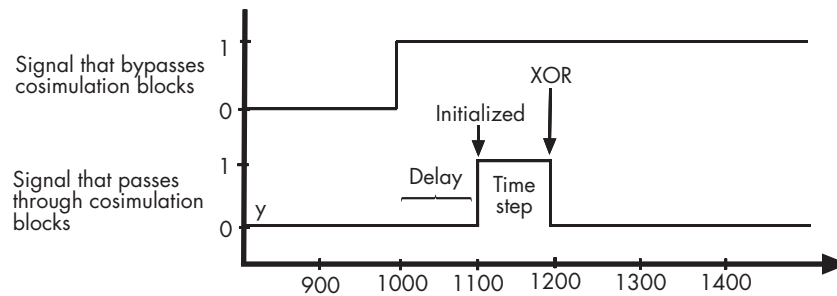
After the simulation runs, the ModelSim **wave** window appears as follows.



Note the following:

- Signal a gets asserted high after a 100 ns delay. This is due to the unit delay applied by the Simulink model.
- Signal b gets asserted high immediately.
- Signal y experiences a falling edge as a result of the XOR computation.

The following figure highlights the individual signal paths that get appear in the Simulink Scope window.

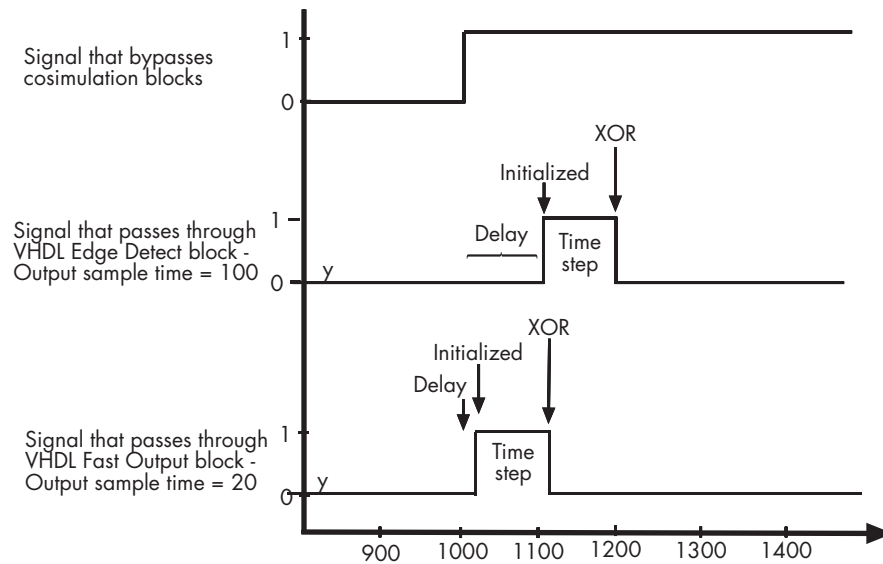


The signal that bypasses the cosimulation blocks rises at  $t=1000$ . That signal stays high for the duration of the sample period. However, the signals that are read from output port  $y$  of the two cosimulation blocks, display in the Scope window as follows:

- After a one time step delay, the signals rise in response to step generator. The delay occurs because the values that the step generator deposit on the cosimulation block's signal paths do not propagate to the block's output until the next Simulink cycle.
- After the next time step, the signal value falls due to the VHDL XOR operation.

For cosimulation blocks that model combinatorial circuits, such as the one in the preceding example, you may want to experiment with a faster sample frequency for output ports. For example, suppose you change the **Output sample time** for the VHDL Fast Output cosimulation block from 100 to 20. The following figure highlights the individual signal paths that appear in the Scope window for this scenario.





In this case, the signal that bypasses the cosimulation blocks and the output signal read from the VHDL Edge Detect block remain the same. However, the delay for the signal read from the VHDL Fast Output block is 20 ticks instead of 100. Although the size of the time step is still tied to the ModelSim resolution limit, the delay that occurs before the VHDL code is processed is significantly reduced and the time of execution more closely reflects simulation time in ModelSim.

---

**Note** Although this type of parameter tuning can increase simulation performance, it can make a model more difficult to debug. For example, it might be necessary to adjust the output sample time for each cosimulation block.

---

## Interfacing with Continuous Time Signals

Use the Simulink Zero-Order Hold block to apply a zero-order hold (ZOH) on continuous signals that are driven into an HDL Cosimulation block.

## Setting Simulink Software Configuration Parameters

When you create a Simulink model that includes one or more EDA Simulator Link MQ Cosimulation blocks, you might want to adjust certain Simulink parameter settings to best meet the needs of HDL modeling. For example, you might want to adjust the value of the **Stop time** parameter in the **Solver** pane of the Configuration Parameters dialog box.

You can adjust the parameters individually or you can use the M-file `dspstartup`, which lets you automate the configuration process so that every new model that you create is preconfigured with the following relevant parameter settings:

<b>Parameter</b>	<b>Default Setting</b>
'SingleTaskRateTransMsg'	'error'
'Solver'	'fixedstepdiscrete'
'SolverMode'	'singletasking'
'StartTime'	'0.0'
'StopTime'	'inf'
'FixedStep'	'auto'
'SaveTime'	'off'
'SaveOutput'	'off'
'AlgebraicLoopMsg'	'error'

The default settings for 'SaveTime' and 'SaveOutput' improve simulation performance.

You can use `dspstartup` by entering it at the MATLAB command line or by adding it to the Simulink `startup.m` file. You also have the option of customizing `dspstartup` settings. For example, you might want to adjust the 'StopTime' to a value that is optimal for your simulations, or set 'SaveTime' to 'on' to record simulation sample times.

For more information on using and customizing `dspstartup`, see the Signal Processing Blockset documentation. For more information about automating tasks at startup, see the description of the startup command in the MATLAB documentation.

### **Running and Testing a Hardware Model in Simulink**

If you take the approach of designing a Simulink model first, run and test your model thoroughly before replacing or adding hardware model components as EDA Simulator Link MQ Cosimulation blocks.

### **Simulink and HDL Simulator Communication Options**

Select shared memory or socket communication. See “Communicating with MATLAB or Simulink and the HDL Simulator” on page 1-8.

### **Starting the HDL Simulator**

See “Starting the HDL Simulator” on page 1-23.

## Incorporating Hardware Designs into a Simulink® Model

### In this section...

“Overview” on page 3-40

“Specifying HDL Signal/Port and Module Paths for Cosimulation” on page 3-41

“Driving Clocks, Resets, and Enables” on page 3-43

“Defining the Block Interface” on page 3-45

“Specifying the Signal Datatypes” on page 3-56

“Configuring the Simulink and ModelSim SE/PE Timing Relationship” on page 3-58

“Configuring the Communication Link in the HDL Cosimulation Block” on page 3-59

“Specifying Pre- and Post-Simulation Tcl Commands with HDL Cosimulation Block Parameters Dialog Box” on page 3-62

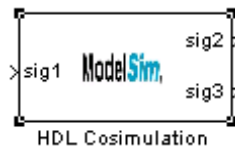
“Programmatically Controlling the Block Parameters” on page 3-64

“Adding a Value Change Dump (VCD) File” on page 3-66

### Overview

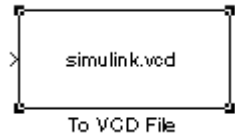
After you code one of your model’s components in VHDL or Verilog and simulate it in the HDL simulator environment, integrate the HDL representation into your Simulink® model as an HDL Cosimulation block by performing the following steps:

- 1** Open your Simulink model, if it is not already open.
- 2** Delete the model component that the HDL Cosimulation block is to replace.
- 3** In the Simulink Library Browser, click the EDA Simulator Link™ MQ block library. The browser displays the block icons shown below.



HDL  
Cosimulation

Block that has at least one input port and one output port.



To VCD File

Generates a Value Change Dump (VCD) file. For information on using this block, see “Adding a Value Change Dump (VCD) File” on page 3-66.

- 4 Copy the HDL Cosimulation block icon from the Library Browser to your model. Simulink creates a link to the block at the point where you drop the block icon.
- 5 Connect any HDL Cosimulation block ports to appropriate blocks in your Simulink model. To model a sink device, configure the block with inputs only. To model a source device, configure the block with outputs only.

## Specifying HDL Signal/Port and Module Paths for Cosimulation

These rules are for signal/port and module path specifications in Simulink. Other specifications may work but are not guaranteed to work in this or future releases.

HDL designs generally do have hierarchy; that is the reason for this syntax. This is not a file name hierarchy.

### Path Specifications for Simulink Cosimulation Sessions with Verilog Top Level

Path specifications must adhere to the following rules:

- Path specification must start with a top-level module name.
- Path specification can include "." or "/" path delimiters, but cannot include a mixture.
- The leaf module or signal must match the HDL language of the top-level module.

The following are valid signal and module path specification examples:

```
top.port_or_sig
/top/sub/port_or_sig
top
top/sub
top.sub1.sub2
```

The following are invalid signal and module path specification examples:

```
top.sub/port_or_sig
:sub:port_or_sig
:
:sub
```

### **Path Specifications for Simulink Cosimulation Sessions with VHDL Top Level**

Path specifications must adhere to the following rules:

- Path specification may include the top-level module name but it is not required.
- Path specification can include "." or "/" path delimiters, but cannot include a mixture.
- The leaf module or signal must match the HDL language of the top-level module.

The following are valid signal and module path specification examples:

```
top.port_or_sig
/sub/port_or_sig
top
top/sub
top.sub1.sub2
```

The following are invalid signal and module path specification examples:

```
top.sub/port_or_sig
:sub:port_or_sig
:
```

:sub

## Driving Clocks, Resets, and Enables

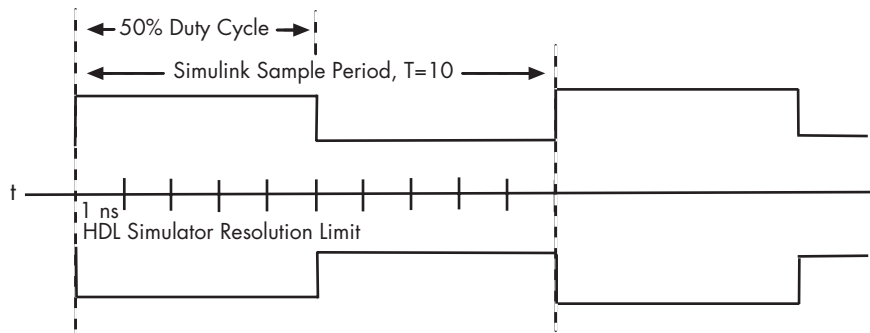
### Creating Optional Clocks

You can create rising-edge or falling-edge clocks that apply internal stimuli to your cosimulation model. When you specify a clock in your block definition, Simulink creates a rising-edge or falling-edge clock that drives the specified HDL signals by depositing them.

Simulink attempts to create a clock that has a 50% duty cycle and a predefined phase that is inverted for the falling edge case. If necessary, Simulink degrades the duty cycle to accommodate odd Simulink sample times, with a worst case duty cycle of 66% for a sample time of  $T=3$ .

The following figure shows a timing diagram that includes rising and falling edge clocks with a Simulink sample time of  $T=10$  and an HDL simulator resolution limit of 1 ns. The figure also shows that given those timing parameters, the clock duty cycle is 50%.

Rising Edge Clock



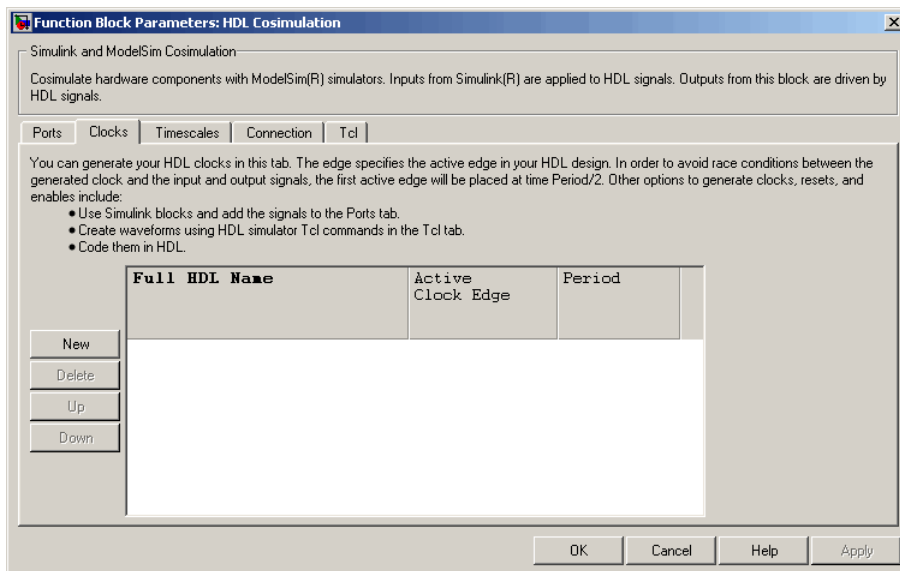
Falling Edge Clock

To create clocks, perform the following steps:

- 1 In the HDL simulator, determine the clock signal path names you plan to define in your block. To do this, you can use the same method explained for

determining the signal path names for ports in step 1 of “Mapping HDL Signals to Block Ports” on page 3-46.

- 2 Select the **Clocks** tab of the Block Parameters dialog. Simulink displays the dialog as shown below.



- 3 Click the **New** button to add a new clock signal.
- 4 Edit the clock signal path name directly in the table under the **Full HDL Name** column by double-clicking on the default clock signal name (/top/clk). Specify your new clock using HDL simulator path name syntax. See “Specifying HDL Signal/Port and Module Paths for Cosimulation” on page 3-41.  
  
Note that vectored signals in the **Clocks** pane are not supported. Signals must be logic types with '1' and '0' values.
- 5 To specify whether the clock generates a rising-edge or falling edge signal, select Rising or Falling from the **Active Clock Edge** list.

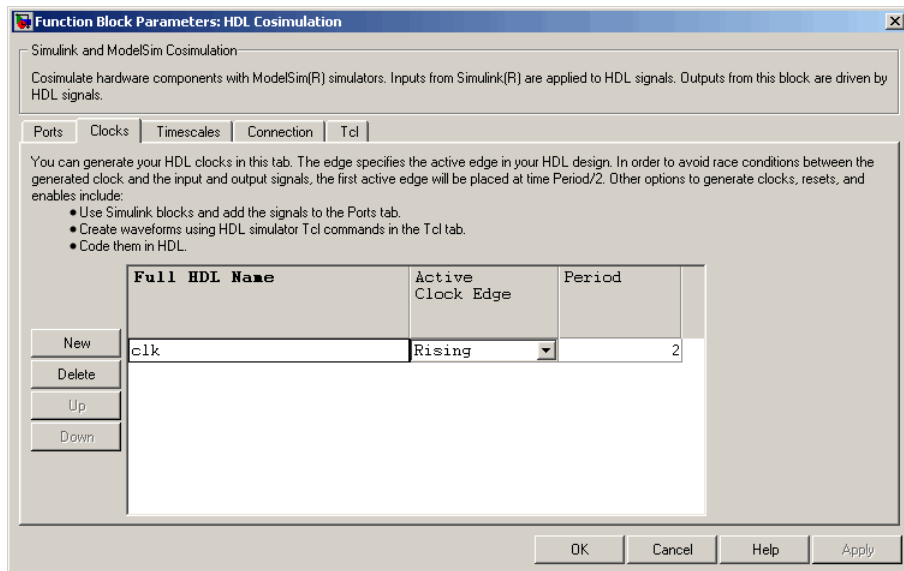


- 6** The **Period** field specifies the clock period. Accept the default (2), or override it by entering the desired clock period explicitly by double-clicking in the **Period** field.

Specify the **Period** field as an even integer, with a minimum value of 2.

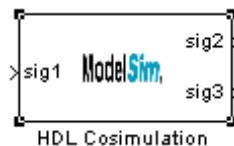
- 7** When you have finished editing clock signals, click **Apply** to register your changes with Simulink.

The following dialog defines the rising-edge clock `clk` for the HDL Cosimulation block, with a default period of 2.

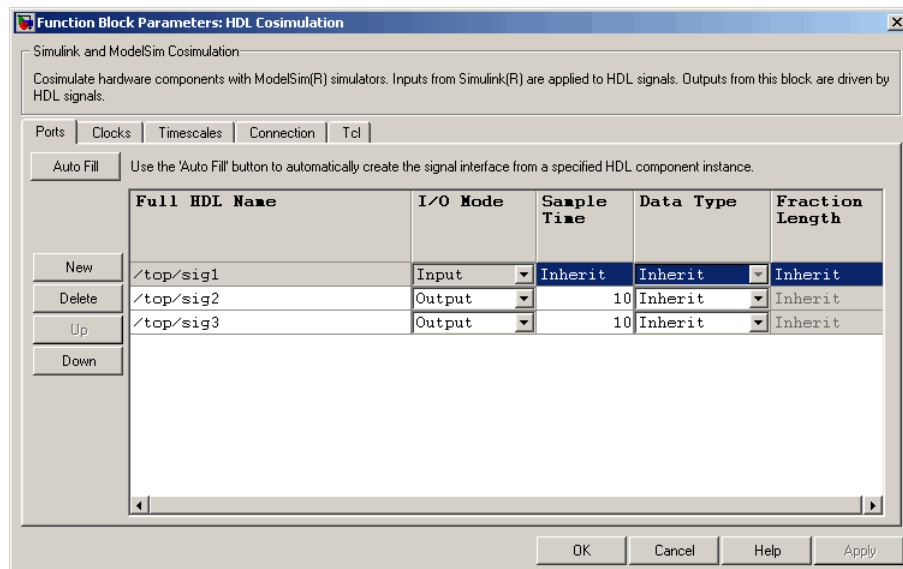


## Defining the Block Interface

To open the block parameters dialog for the HDL Cosimulation block, double-click the block icon.



Simulink displays the following Block Parameters dialog.



### Mapping HDL Signals to Block Ports

The first step to configuring your EDA Simulator Link MQ Cosimulation block is to map signals and signal instances of your HDL design to port definitions in your HDL Cosimulation block. In addition to identifying input and output ports, you can specify a sample time for each output port. You can also specify a fixed-point data type for each output port.

The signals that you map can be at any level of the HDL design hierarchy.

To map the signals, you can perform either of the following actions:

- Enter signal information manually into the **Ports** pane of the HDL Cosimulation Block Parameters dialog (see “Entering Signal Information Manually” on page 3-52). This approach can be more efficient when you want to connect a small number of signals from your HDL model to Simulink.
- Use the **Auto Fill** button to obtain signal information automatically by transmitting a query to the HDL simulator. This approach can save

significant effort when you want to cosimulate an HDL model that has many signals that you want to connect to your Simulink model. Note, however, that in some cases you will need to edit the signal data returned by the query. See “Obtaining Signal Information Automatically from the HDL Simulator” on page 3-47 for details.

---

**Note** Make sure that signals being used in cosimulation have read/write access (this is done through the HDL simulator—see product documentation for details). This rule applies to all signals on the **Ports**, **Clocks**, and **Tcl** panes.

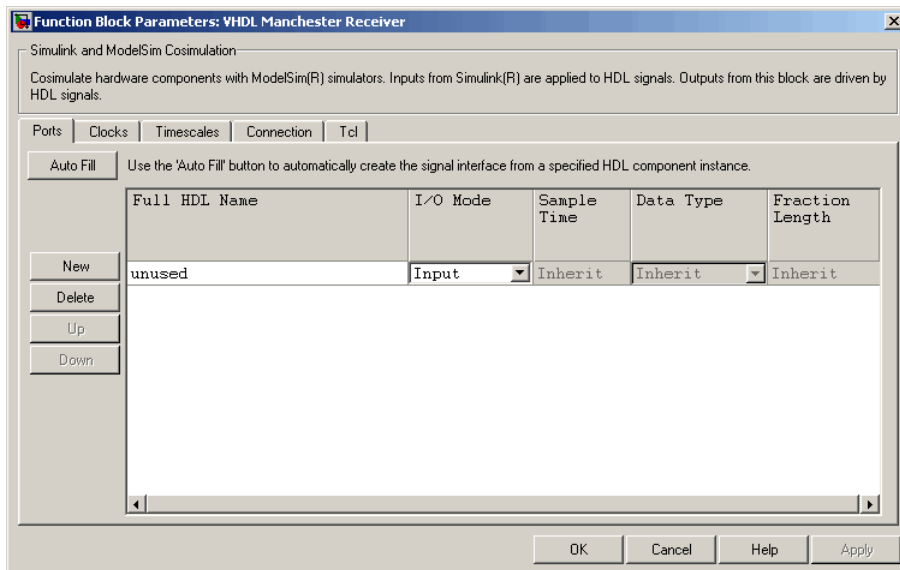
---

### **Obtaining Signal Information Automatically from the HDL Simulator.**

The **Auto Fill** button lets you begin an HDL simulator query and supply a path to a component or module in an HDL model under simulation in the HDL simulator. Usually, some change of the port information is required after the query completes. The required steps are outlined in the example below.

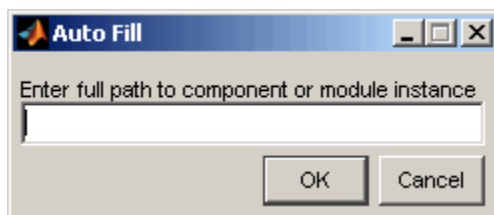
The example is based on a modified copy of the Manchester Receiver model (see “Creating a Hardware Model Design for Use in Simulink® Applications” on page 3-5), in which all signals were first deleted from the **Ports** and **Clocks** panes.

- 1 Open the block parameters dialog for the HDL Cosimulation block. Click the **Ports** tab. The **Ports** pane opens.



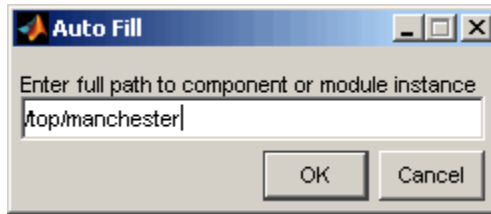
**Tip** Delete all ports before performing **Auto Fill**. This ensures that no unused signal is present in the Ports list at any time.

- 2 Click the **Auto Fill** button. The **Auto Fill** dialog opens.

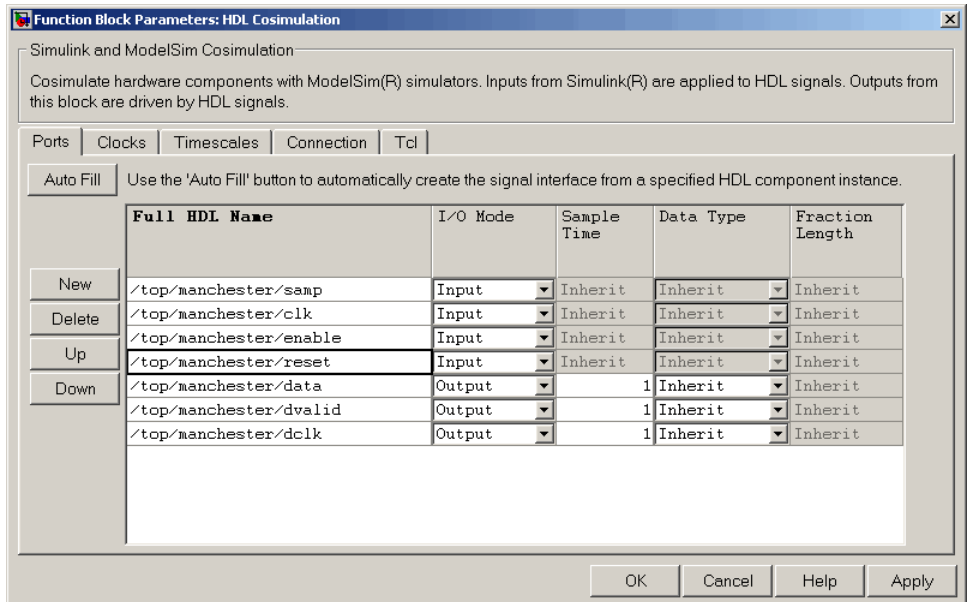


This modal dialog requests an instance path to a component or module in your HDL model; here you enter an explicit HDL path into the edit field. Note this is not a file path and has nothing to do with the source files.

- 3 In this example, we will obtain port data for a VHDL component called manchester. The HDL path is specified as /top/manchester.

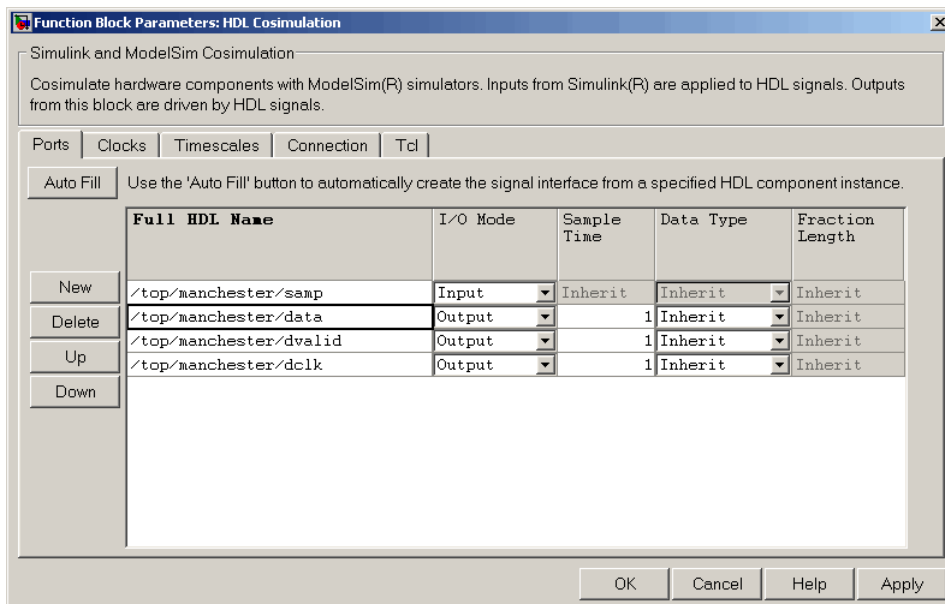


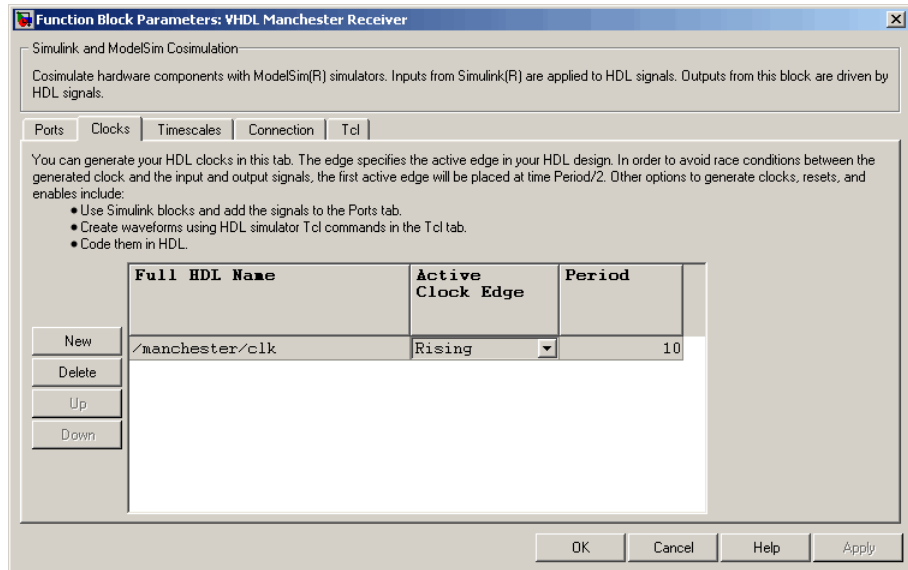
- 4 Click **OK**. The dialog is dismissed and the query is transmitted.
- 5 Port data is returned and entered into the **Ports** pane, as shown in the figure below.



- 6 Click **Apply** to commit the port additions.
- 7 Observe that **Auto Fill** has returned information about *all* inputs and outputs for the targeted component. In many cases, this will include signals that function in the HDL simulator but cannot be connected in the Simulink model. You may delete any such entries from the list in the **Ports** pane if they are unwanted. You *can* drive the signals from Simulink; you just have to define their values by laying down Simulink blocks.

The figure above shows that the query entered clock, clock enable, and reset ports (labeled clk, enable, and reset respectively) into the ports list. In this example, the clk signal is entered in the **Clocks** pane, and the enable and reset signals are deleted from the **Ports** pane, as shown in the figures below.





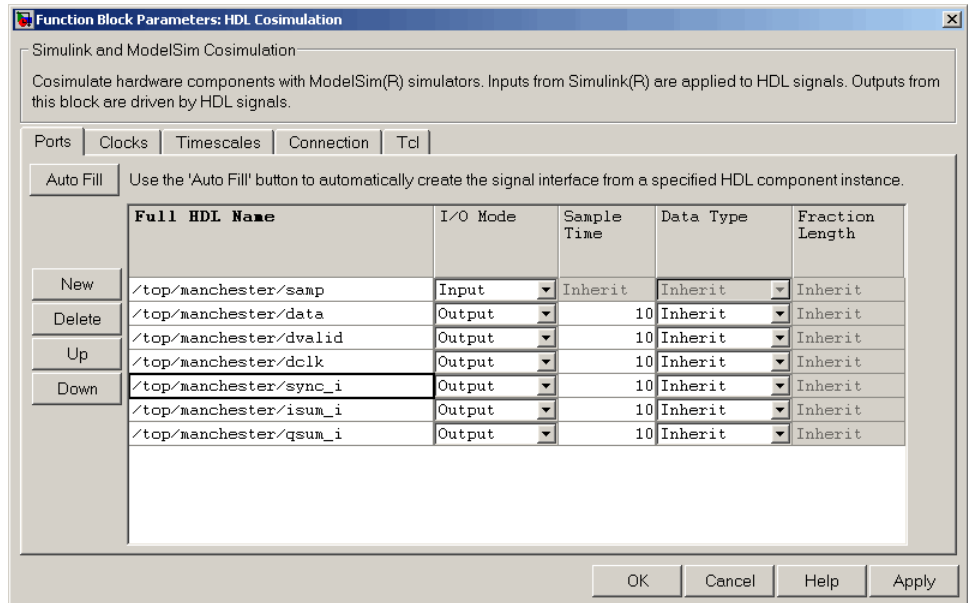
**8** **Auto Fill** returns default values for output ports:

- **Sample time:** 1
- **Data type:** Inherit
- **Fraction length:** Inherit

You may need to change these values as required by your model. In this example, the **Sample time** should be set to 10 for all outputs. See also “Specifying the Signal Datatypes” on page 3-56.

**9** Note that **Auto Fill** does not return information for internal signals. If your Simulink model needs to access such signals, you must enter them into the **Ports** pane manually. For example, in the case of the Manchester Receiver model, you would need to add output port entries for top/manchester/sync\_i, top/manchester/isum\_i, and top/manchester/qsum\_i, as shown below.

**10** Before closing the HDL Cosimulation block parameters dialog, click **Apply** to commit any edits you have made.




---

**Note** When importing VHDL signals, signal names are returned in all capitals.

---



---

**Note** Enter force commands in the **Tcl** pane to drive the reset and enable signals; for example:

```
force design/reset value time
```

where *value* is '1' or '0' and *time* is in nanoseconds.

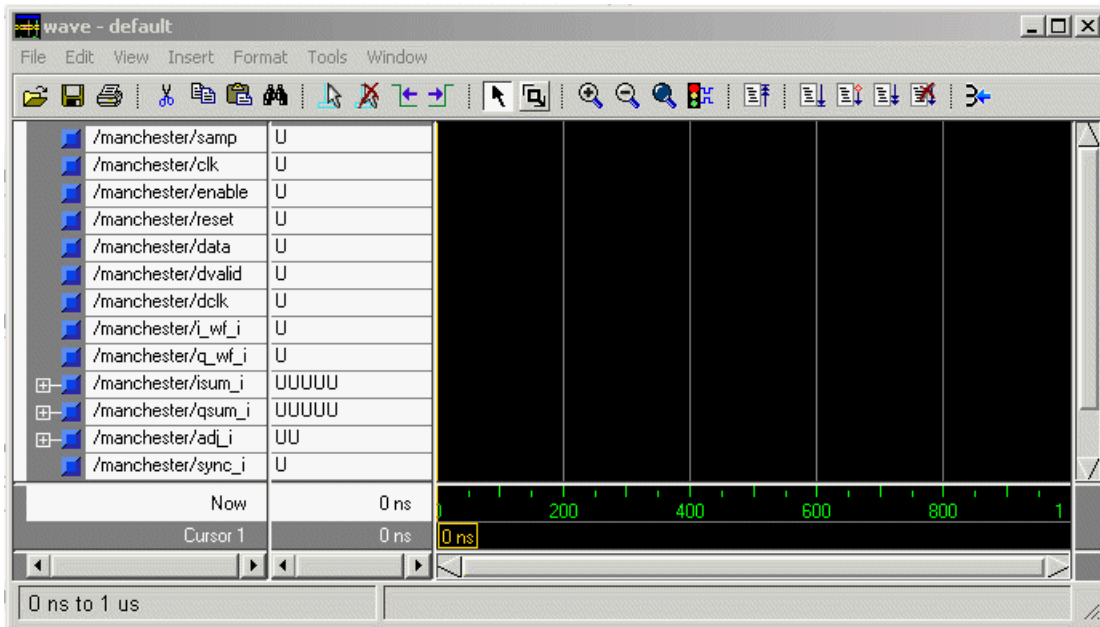
---

**Entering Signal Information Manually.** To enter signal information directly in the **Ports** pane, perform the following steps:

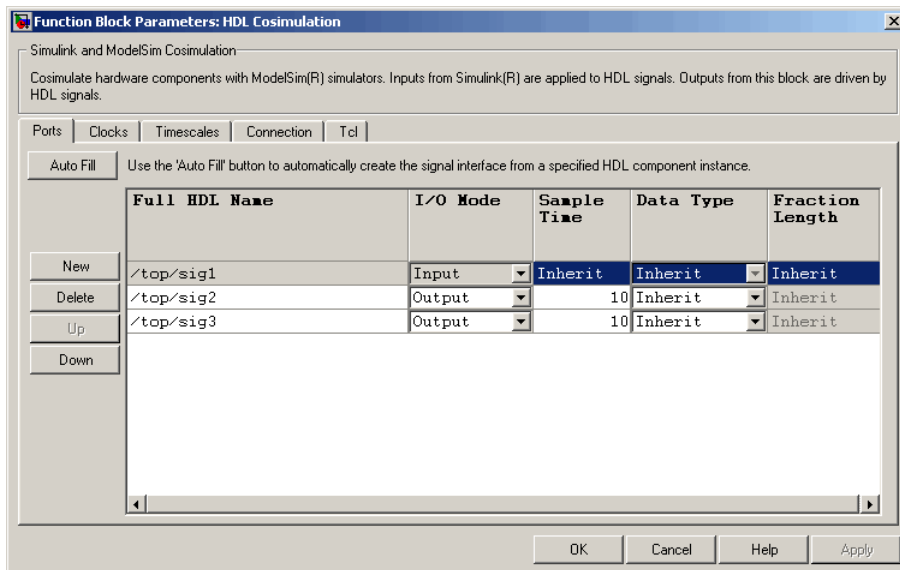
- 1 In the HDL simulator, determine the signal path names for the HDL signals you plan to define in your block.



For example, the following shows all signals are subordinate to the top-level module manchester.



- 2 In Simulink, open the block parameters dialog for your HDL Cosimulation block, if it is not already open.
- 3 Select the **Ports** tab of the Block Parameters dialog. Simulink displays the dialog as shown below.

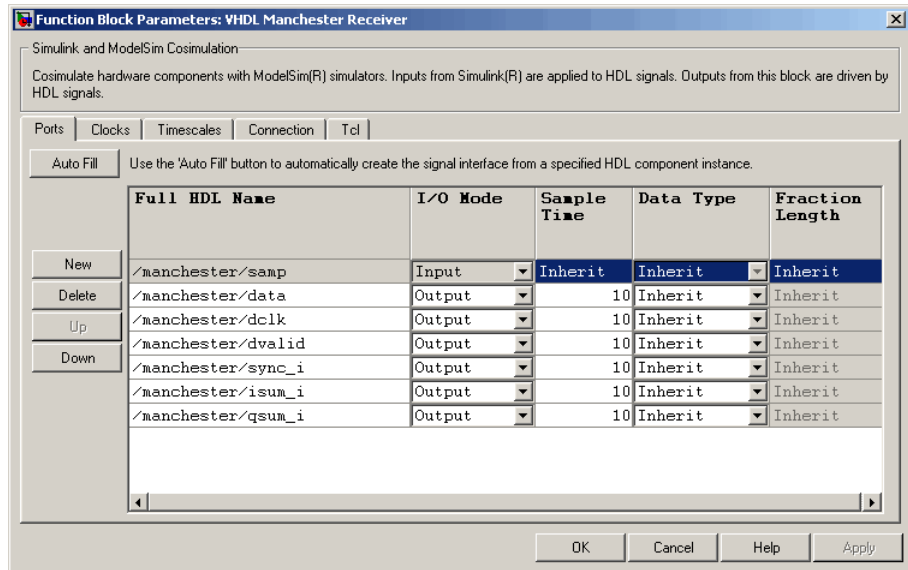


In this pane, you define the HDL signals of your design that you want to include in your Simulink block and set a sample time and data type for output ports. The parameters that you should specify on the **Ports** pane depend on the type of device the block is modeling as follows:

- For a device having both inputs and outputs: specify block input ports, block output ports, output sample times and output data types. For output ports, accept the default or enter an explicit sample time. Data types can be specified explicitly, or set to **Inherit** (the default). In the default case, the output port data type is inherited either from the signal connected to the port, or derived from the HDL model.
  - For a sink device: specify block output ports
  - For a source device: specify block input ports
- 4** Enter signal path names in the **Full HDL name** column by double-clicking on the existing default signal. Use HDL simulator path name syntax (see “Specifying HDL Signal/Port and Module Paths for Cosimulation” on page 3-41). If you are adding signals, click **New** and then edit the default values. Select either **Input** or **Output** from the **I/O Mode** column. If desired, set the **Sample Time**, **Data Type**, and **Fraction Length** parameters for signals explicitly, as discussed below.

When you have finished editing clock signals, click **Apply** to register your changes with Simulink.

The following dialog shows port definitions for an HDL Cosimulation block. Note the signal path names match path names that appear in the HDL simulator **wave** window.



**Note** When you define an input port, make sure that only one source is set up to force input to that port. If multiple sources drive a signal, your Simulink model may produce unpredictable results.

- 5 You must specify a sample time for the output ports. Simulink uses the value that you specify, and the current settings of the **Timescales** pane, to calculate an actual simulation sample time.

For more information on sample times in the EDA Simulator Link MQ cosimulation environment, see “Representation of Simulation Time” on page 3-15.

- 6 You can configure the fixed-point data type of each output port explicitly if desired, or use a default (Inherited) . In the default case, Simulink determines the data type for an output port as follows:

If Simulink can determine the data type of the signal connected to the output port, it applies that data type to the output port. For example, the data type of a connected Signal Specification block is known by back-propagation. Otherwise, Simulink queries the HDL simulator to determine the data type of the signal from the HDL module.

To assign an explicit fixed-point data type to a signal, perform the following steps:

- a Select either Signed or Unsigned from the **Data Type** column.
- b If the signal has a fractional part, enter the **Fraction Length**.

For example, an 8-bit signal with Signed data type and a **Fraction Length** of 5 is assigned the data type `sfix8_En5`. An Unsigned 16-bit signal with no fractional part (a **Fraction Length** of 0) is assigned the data type `ufix16`.

- 7 Before closing the dialog, click **Apply** to register your edits.

## Specifying the Signal Datatypes

The **Data Type** and **Fraction Length** parameters apply only to output signals, as follows:

- 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. The **Fraction Length** property is enabled when the signal **Data Type** property is not set to Inherit.

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

- VHDL signals of `STD_LOGIC` or `STD_LOGIC_VECTOR` type

- Verilog signals of wire or reg type

Output port data types are determined by the signal width and by the **Data Type** and **Fraction Length** properties of the signal. 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.

**Inherit** is the default setting. When **Inherit** is selected, the **Fraction Length** edit field is disabled.

Simulink attempts to compute the data type of the signal connected to the output port by backward propagation. For example, if a Signal Specification block is connected 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 data type `STD_LOGIC_VECTOR` for a VHDL 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 **Signed** is selected, the **Fraction Length** edit field is enabled. The port is assigned 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 **Unsigned** is selected, the **Fraction Length** edit field is enabled. The port is assigned a fixed point type `ufixN_EnF`, where  $N$  is the signal width and  $F$  is the **Fraction Length** value.

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`.

## Configuring the Simulink and ModelSim SE/PE Timing Relationship

You configure the timing relationship between Simulink and the HDL simulator by using the **Timescales** pane of the block parameters dialog. Before setting the **Timescales** parameters, you should read “Representation of Simulation Time” on page 3-15 to understand the supported timing modes and the issues that will determine your choice of timing mode.

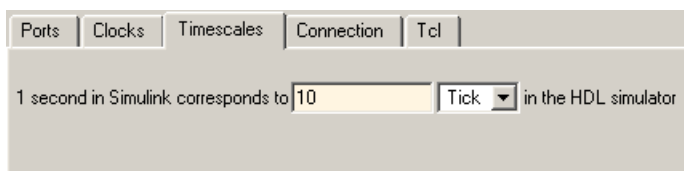
You can specify either a relative or an absolute timing relationship between Simulink and the HDL simulator, as described in the sections below.

### Specifying a Relative Timing Relationship

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

- 1 Select the **Timescales** tab of the HDL Cosimulation block parameters dialog.
- 2 Select Tick from the list on the right. (This is the default.)
- 3 Enter a scale factor in the text box on the left. The default scale factor is 1.

For example, in the figure below, the **Timescales** pane is configured for a relative timing correspondence of 10 HDL simulator ticks to 1 Simulink second.



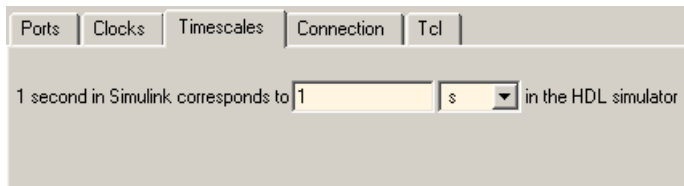
- 4 Click **Apply** to commit your changes.

### Specifying an Absolute Timing Relationship

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

- 1 Select the **Timescales** tab of the HDL Cosimulation block parameters dialog.
- 2 Select a unit of absolute time from the list on the right. Available units are fs, ps, ns, us, ms, and s.
- 3 Enter a scale factor in the text box on the left. The default scale factor is 1.

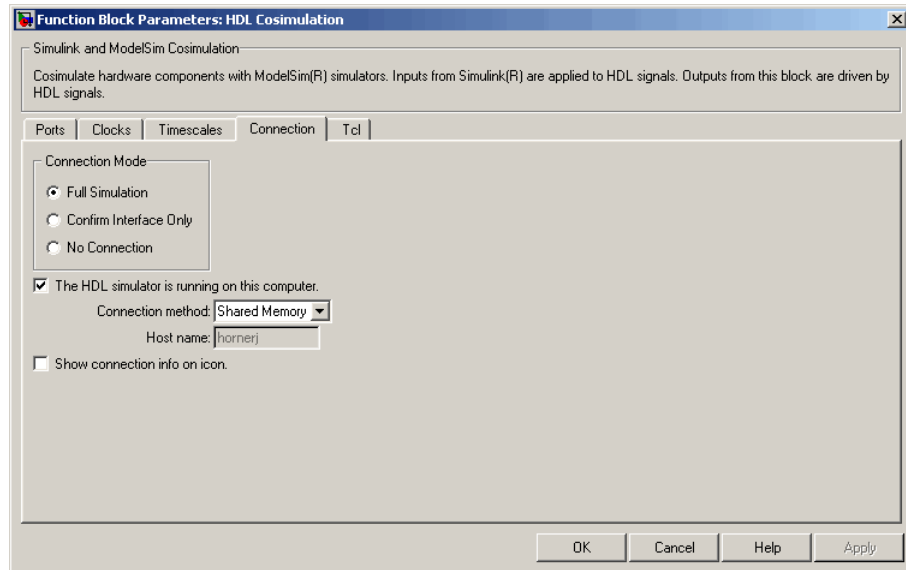
For example, in the figure below, the **Timescales** pane is configured for an absolute timing correspondence of 1 HDL simulator second to 1 Simulink second.



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

## Configuring the Communication Link in the HDL Cosimulation Block

Configure a block's communication link with the **Connection** pane of the block parameters dialog.



The following steps guide you through the communication configuration:

- 1 Determine whether Simulink and the HDL simulator are running on the same computer. If they are, skip to step 4.
- 2 Clear the **The HDL simulator is running on this computer** check box. (This check box is selected by default.) Note that since Simulink and the HDL simulator are running on different computers, **Connection method** is automatically set to Socket.
- 3 Enter the hostname of the computer that is running your HDL simulation (in the HDL simulator) in the **Host name** text field. In the **Port number or service** text field, specify a valid port number or service for your computer system. For information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports” on page D-2. Skip to step 5.
- 4 If the HDL simulator and Simulink are running on the same computer, decide whether you are going to use shared memory or TCP/IP sockets for the communication channel. For information on the different modes of communication, see “Communicating with MATLAB or Simulink and the HDL Simulator” on page 1-8.



If you choose TCP/IP socket communication, specify a valid port number or service for your computer system in the **Port number or service** text field. For information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports” on page D-2.

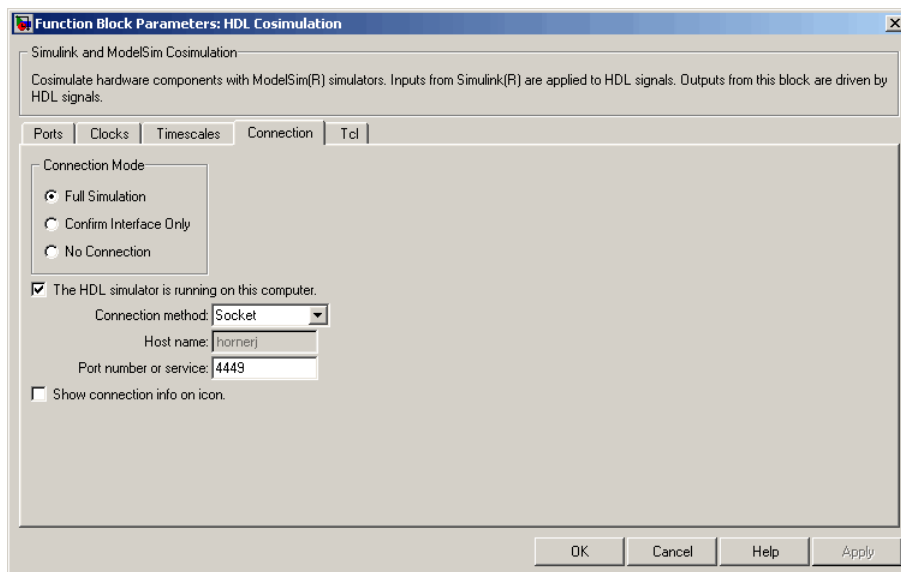
If you choose shared memory communication, select the **Shared memory** check box.

- 5** If you want to bypass the HDL simulator when you run a Simulink simulation, use the **Connection Mode** options to specify what type of simulation connection you want. Select one of the following:
- **Full Simulation:** Confirm interface and run HDL simulation (default).
  - **Confirm Interface Only:** Check HDL simulator 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 2nd and 3rd options, EDA Simulator Link MQ software does not communicate with the HDL simulator during Simulink simulation.

**6** Click **Apply**.

The following example dialog shows communication definitions for an HDL Cosimulation block. The block is configured for Simulink and the HDL simulator running on the same computer, communicating in TCP/IP socket mode over TCP/IP port 4449.



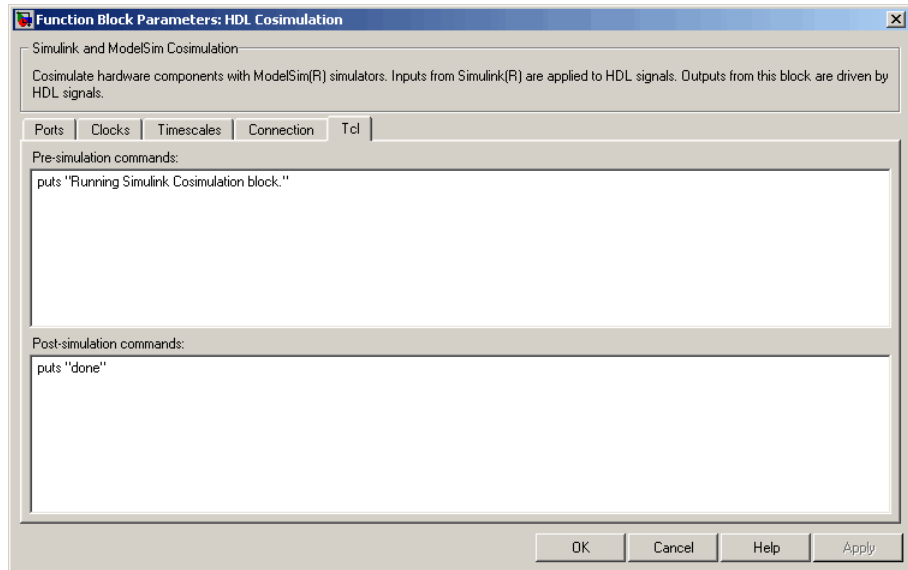
## Specifying Pre- and Post-Simulation Tcl Commands with HDL Cosimulation Block Parameters Dialog Box

You have the option of specifying Tcl commands to execute before and after the HDL simulator simulates the HDL component of your Simulink model. Tcl is a programmable scripting language supported by most HDL simulation environments. Use of Tcl can range from something as simple as a one-line puts command to confirm that a simulation is running or as complete as a complex script that performs an extensive simulation initialization and startup sequence. For example, the **Post- simulation command** field on the Tcl Pane is particularly useful for instructing the HDL simulator to restart at the end of a simulation run.

You can specify the pre- and post-simulation Tcl commands by entering Tcl commands in the Pre-simulation commands or Post-simulation commands text fields of the HDL Cosimulation block.

To specify Tcl commands, perform the following steps:

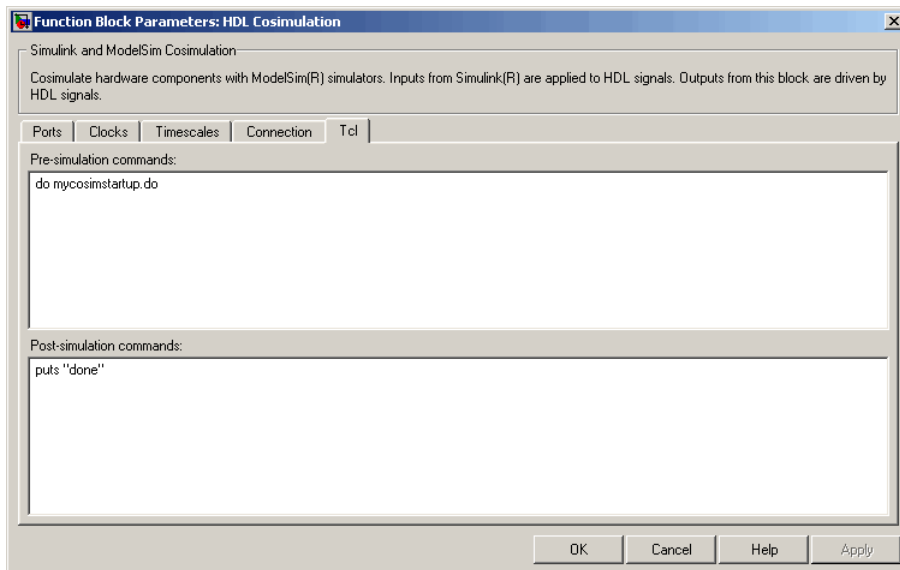
- 1 Select the **Tcl** tab of the Block Parameters dialog box. The dialog box appears as follows.



The **Pre-simulation commands** text box includes an puts command for reference purposes.

- 2 Enter one or more commands in the **Pre-simulation command** and **Post-simulation command** text boxes. 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 (;), which is the standard Tcl concatenation operator.

Alternatively, you can create a ModelSim DO file that lists Tcl commands and then specify that file with the ModelSim do command as shown in the following figure.



3 Click **Apply**.

## Programmatically Controlling the Block Parameters

One way to control block parameters is through the HDL Cosimulation block graphical dialog box. However, you can also control blocks by programmatically controlling the mask parameter values and the running of simulations. Parameter values can be read using the Simulink `get_param` function and written using the Simulink `set_param` function. All block parameters have attributes that indicate whether they are:

- Tunable — the attributes can change during the simulation run
- Evaluated — the parameter string value is put through an evaluation to determine its actual value used by the S-Function

The HDL Cosimulation block does not have any tunable parameters; thus, you get an error if you try to change a value while the simulation is running, but it does have a few evaluated parameters.

You can see the list of parameters and their attributes by performing a right-mouse click on the block, selecting **View Mask**, and then the

**Parameters** tab. The **Variable** column shows the programmatic parameter names. Alternatively, you can get the names programmatically by selecting the HDL Cosimulation block and then typing at the MATLAB prompt:

```
>> get_param(gcf, 'DialogParameters')
```

Some examples of using MATLAB to control simulations and mask parameter values follow. Usually, the commands are put into an M-script or M-function file and automatically called by several callback hooks available to the model developer. You can place the code in any of these suggested locations, or anywhere you choose:

- In the model workspace, e.g., **View > Model Explorer > Simulink Root > *model\_name* > Model Workspace > Data Source is M-Code.**
- In a model callback, e.g., **File > Model Properties > Callbacks.**
- A subsystem callback (right-mouse click on an empty subsystem and then select **Block Properties > Callbacks**). Many of the EDA Simulator Link MQ demos use this technique to start the HDL simulator by placing M-code in the OpenFcn callback.
- The HDL Cosimulation block callback (right-mouse click on HDL Cosimulation block, and then select **Block Properties > Callbacks**)

## Examples

The following examples show the use of programmatically controlling the HDL Cosimulation block parameters.

- “Scripting the Value of the Socket Number for HDL Simulator Communication” on page 3-65
- 

### Scripting the Value of the Socket Number for HDL Simulator

**Communication.** In a regression environment, you may need to determine the socket number for the Simulink/HDL simulator connection during the simulation to avoid collisions with other simulation runs. This example shows code that could handle that task. The script is for a 32-bit Linux platform.

```
ttcp_exec = [matlabroot ' /toolbox/shared/hdlink/scripts/ttcp_glnx'];
[status, results] = system([ttcp_exec ' -a']);
```

```
if ~s
    parsed_result = strread(results,'%s');
    avail_port = parsed_result{2};
else
    error(results);
end

set_param('MyModel/HDL Cosimulation', 'CommPortNumber', avail_port);
```

## Adding a Value Change Dump (VCD) File

A value change dump (VCD) file logs changes to variable values, such as the values of signals, in a file during a simulation session. VCD files can be useful during design verification. Some examples of how you might apply VCD files include the following cases:

- 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

VCD files can provide data that you might not otherwise acquire unless you understood the details of a device's internal logic. In addition, they include data that can be graphically displayed or analyzed with postprocessing tools.

For example, the ModelSim `vcd2wlf` tool converts a VCD file to a Wave Log Format (WLF) file that you can view in a ModelSim **wave** window. Other examples of postprocessing include the extraction of data about a particular section of a design hierarchy or data generated during a specific time interval.

The To VCD File block provided in the EDA Simulator Link MQ block library serves as a VCD file generator during Simulink sessions. The block generates a VCD file that contains information about changes to signals connected to the block's input ports and names the file with a specified file name.

---

**Note** The To VCD File block logs changes to states '1' and '0' only. The block does *not* log changes to states 'X' and 'Z'.

---

To generate a VCD file, perform the following steps:

- 1 Open your Simulink model, if it is not already open.
- 2 Identify where you want to add the To VCD File block. For example, you might temporarily replace a scope with this block.
- 3 In the Simulink Library Browser, click the EDA Simulator Link MQ block library.



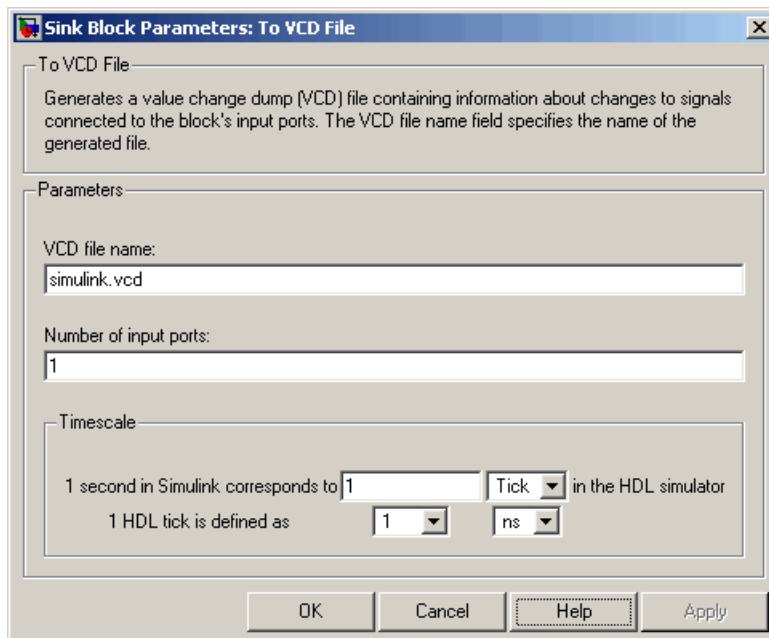
- 4 Copy the To VCD File block from the Library Browser to your model by clicking the block and dragging it from the browser to your model window.
- 5 Connect the block ports to appropriate blocks in your Simulink model.

---

**Note** Because multi-dimensional signals are not part of the VCD specification, they are flattened to a 1D vector in the file.

---

- 6 Configure the To VCD File block by specifying values for parameters in the Block Parameters dialog, as follows.
  - a Double-click the block icon. Simulink displays the following dialog.



- b** Specify a file name for the generated VCD file in the **VCD file name** text box. If you specify a file name only, Simulink places the file in your current MATLAB directory. Specify a complete path name to place the generated file in a different location.

---

**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.

---

- c** Specify an integer in the **Number of input ports** text box that indicates 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.
- d** Click **OK**.



- 7** Choose a timing relationship between Simulink and the HDL simulator. The time scale options specify a correspondence between one second of Simulink time and some quantity of HDL simulator time. Choose relative time or absolute time. For more on the To VCD File time scale, see the reference documentation for the To VCD File block.
- 8** Run the simulation. Simulink captures the simulation data in the VCD file as the simulation runs.

For a description of the VCD file format see “VCD File Format” on page 6-25.

For a sample application of a VCD file, see “toVCD Block Tutorial” on page 3-90.

## Running Cosimulation Sessions

In this section...
“Starting the HDL Simulator for Use with Simulink” on page 3-70
“Determining an Available Socket Port Number” on page 3-71
“Checking the Connection Status” on page 3-71
“Managing a Simulink Cosimulation Session” on page 3-71

### Starting the HDL Simulator for Use with Simulink

The options available for starting the HDL simulator for use with Simulink vary depending on whether you run the HDL simulator and Simulink on the same computer system.

If both tools are running on the same system, start the HDL simulator directly from MATLAB by calling the MATLAB function `vsim`. Alternatively, you can start the HDL simulator manually and load the EDA Simulator Link™ MQ libraries yourself. Either way, see “Starting the HDL Simulator” on page 1-23.

### Loading an HDL Module for Cosimulation

After you start the HDL simulator from MATLAB, load an instance of an HDL module for cosimulation with the HDL simulator command `vsimulink`. Issue the command for each instance of an HDL module in your model that you want to cosimulate. For example:

```
vsimulink work.manchester
```

This command opens a simulation workspace for `manchester` and displays a series of messages in the HDL simulator command window as the simulator loads the HDL module’s packages and architectures.

## Determining an Available Socket Port Number

### Checking the Connection Status

You can check the connection status by clicking the Update diagram button



or by selecting **Edit > Update Diagram**. If there is a connection error, Simulink will notify you.

The MATLAB command `pingHdlSim` can also be used to check the connection status. If a -1 is returned, then there is no connection with the HDL simulator.

### Managing a Simulink Cosimulation Session

To run and test a cosimulation model in Simulink, click **Simulation > Start**



or the Start Simulation button in your Simulink model window. Simulink runs the model and displays any errors that it detects.

If you need to reset a clock during a cosimulation, you can do so by entering HDL simulator force commands at the HDL simulator command prompt or by specifying HDL simulatorforce commands in the **Post- simulation command** text field on the **Tcl** pane of your EDA Simulator Link MQ Cosimulation block's parameters dialog.

If you change any part of the Simulink model, including the HDL Cosimulation block parameters, re-run the simulation or click the Update



diagram button or select **Edit > Update Diagram** so that the diagram reflects those changes.

## Simulink and ModelSim Tutorial

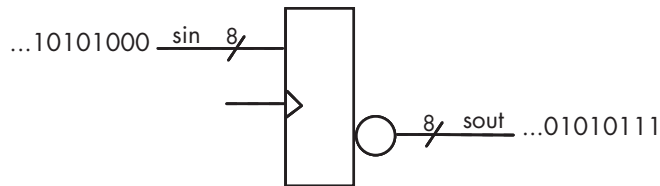
In this section...
“Tutorial Overview” on page 3-72
“Developing the VHDL Code” on page 3-72
“Compiling the VHDL File” on page 3-73
“Creating the Simulink Model” on page 3-75
“Setting Up ModelSim for Use with Simulink” on page 3-84
“Loading Instances of the VHDL Entity for Cosimulation with Simulink” on page 3-85
“Running the Simulation” on page 3-86
“Shutting Down the Simulation” on page 3-89

### Tutorial Overview

This chapter guides you through the basic steps for setting up an EDA Simulator Link™ MQ session that uses Simulink and the HDL Cosimulation block to verify an HDL model. The HDL Cosimulation block cosimulates a hardware component by applying input signals to and reading output signals from an HDL model under simulation in ModelSim. The HDL Cosimulation block supports simulation of either VHDL or Verilog models. In the tutorial in this section, we will cosimulate a simple VHDL model.

### Developing the VHDL Code

A typical Simulink and ModelSim scenario is to create a model for a specific hardware component in ModelSim that you later need to integrate into a larger Simulink model. The first step is to design and develop a VHDL model in ModelSim. In this tutorial, you use ModelSim and VHDL to develop a model that represents the following inverter:



The VHDL entity for this model will represent 8-bit streams of input and output signal values with an IN port and OUT port of type `STD_LOGIC_VECTOR`. An input clock signal of type `STD_LOGIC` will trigger the bit inversion process when set.

Perform the following steps:

**1** Start ModelSim

**2** Change to the writable directory `MyPlayArea`, which you may have created for another tutorial. If you have not created the directory, create it now. The directory must be writable.

```
ModelSim>cd C:/MyPlayArea
```

**3** Open a new VHDL source edit window.

**4** Add the following VHDL code:

**5** Save the file to `inverter.vhd`.

## Compiling the VHDL File

This section explains how to set up a design library and compile `inverter.vhd`, as follows:

**1** Verify that the file `inverter.vhd` is in the current directory by entering the `ls` command at the ModelSim command prompt.

**2** Create a design library to hold your compilation results. To create the library and required `_info` file, enter the `vlib` and `vmap` commands as follows:

```
ModelSim> vlib work
```

```
ModelSim> vmap work work
```

If the design library work already exists, ModelSim *does not* overwrite the current library, but displays the following warning:

```
# ** Warning: (vlib-34) Library already exists at "work".
```

---

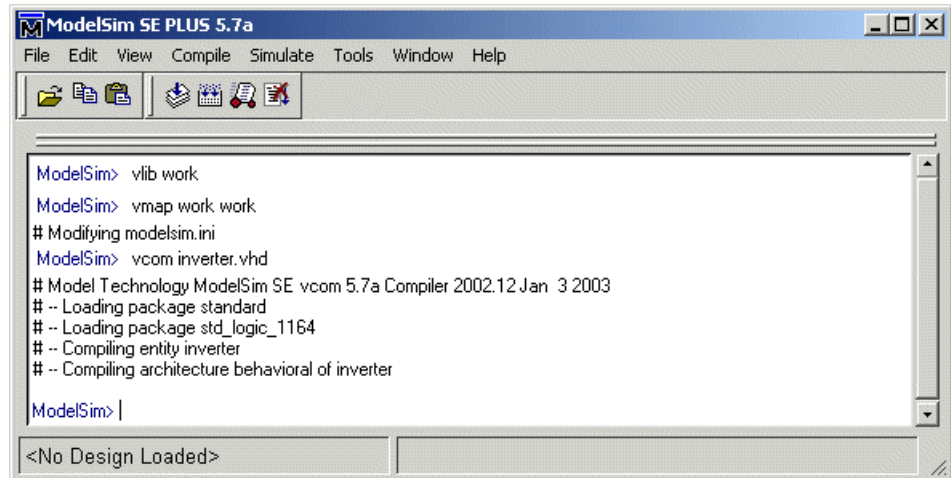
**Note** You must use the ModelSim **File** menu or `vlib` command to create the library directory to ensure that the required `_info` file is created. Do not create the library with operating system commands.

---

- 3** Compile the VHDL file. One way of compiling the file is to click the file name in the project workspace and select **Compile > Compile All**. Another alternative is to specify the name of the VHDL file with the `vcom` command, as follows:

```
ModelSim> vcom inverter.vhd
```

If the compilations succeed, informational messages appear in the command window and the compiler populates the work library with the compilation results.



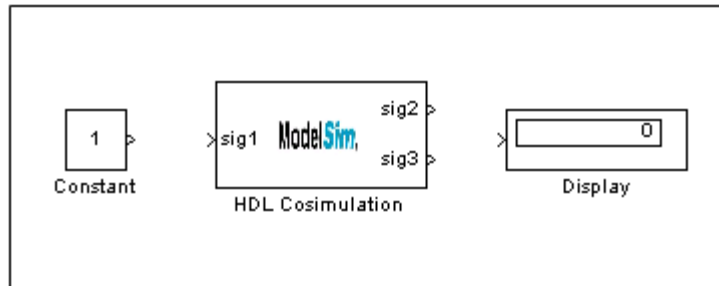
## Creating the Simulink Model

Now create your Simulink model. For this tutorial, you create a simple Simulink model that drives input into a block representing the VHDL inverter you coded in “Developing the VHDL Code” on page 3-72 and displays the inverted output.

Start by creating a model, as follows:

- 1 Start MATLAB, if it is not already running. Open a new model window. Then, open the Simulink Library Browser.
- 2 Drag the following blocks from the Simulink Library Browser to your model window:
  - Constant block from the Simulink Source library
  - HDL Cosimulation block from the EDA Simulator Link MQ block library
  - Display block from the Simulink Sink library

Arrange the three blocks as shown below.



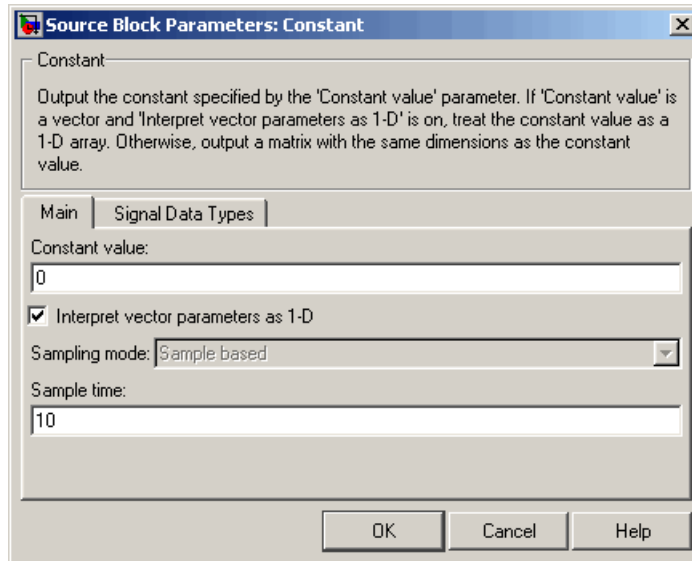
Next, configure the Constant block, which is the model's input source, by performing the following actions:

- 1 Double-click the Constant block icon to open the Constant block parameters dialog. Enter the following parameter values in the **Main** pane:
  - **Constant value:** 0
  - **Sample time:** 10

Later you can change these initial values to see the effect various sample times have on different simulation runs.

The dialog box should now appear as follows.

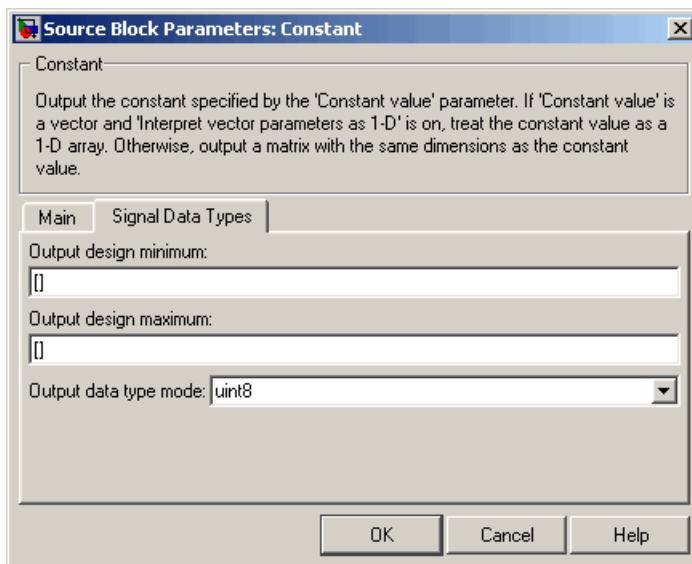




- 2 Click the **Signal data types** tab. The dialog box now displays the **Output data type mode** menu.

Select `uint8` from the **Output data type mode** menu. This data type specification is supported by EDA Simulator Link MQ software without the need for a type conversion. It maps directly to the VHDL type for the VHDL port `sin`, `STD_LOGIC_VECTOR(7 DOWNTO 0)`.

The dialog box should now appear as follows.



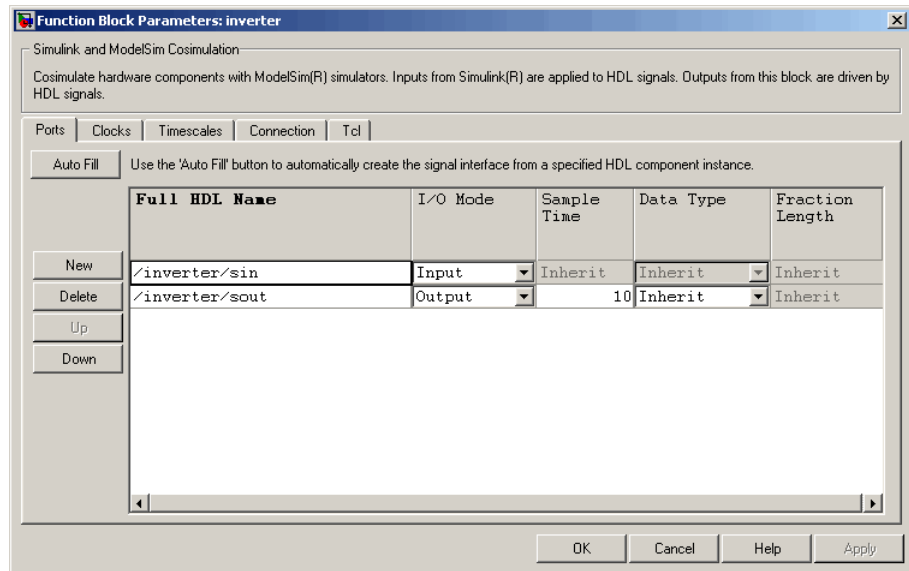
- 3 Click **OK**. The Constant block parameters dialog closes and the value in the Constant block icon changes to 0.

Next, configure the HDL Cosimulation block, which represents the inverter model written in VHDL. Start with the **Ports** pane, by performing the following actions:

- 1 Double-click the HDL Cosimulation block icon. The Block Parameters dialog for the HDL Cosimulation block appears. Click the **Ports** tab.
- 2 In the **Ports** pane, select the sample signal /top/sig1 from the signal list in the center of the pane by double-clicking on it.
- 3 Replace the sample signal path name /top/sig1 with /inverter/sin. Then click **Apply**. The signal name on the HDL Cosimulation block changes.
- 4 Similarly, select the sample signal /top/sig2. Change the **Full HDL Name** to /inverter/sout. Select Output from the **I/O Mode** list. Change the **Sample Time** parameter to 10. Then click **Apply** to update the list.

- 5 Select the sample signal `/top/sig3`. Click the **Delete** button. The signal is now removed from the list.

The **Ports** pane should appear as follows.



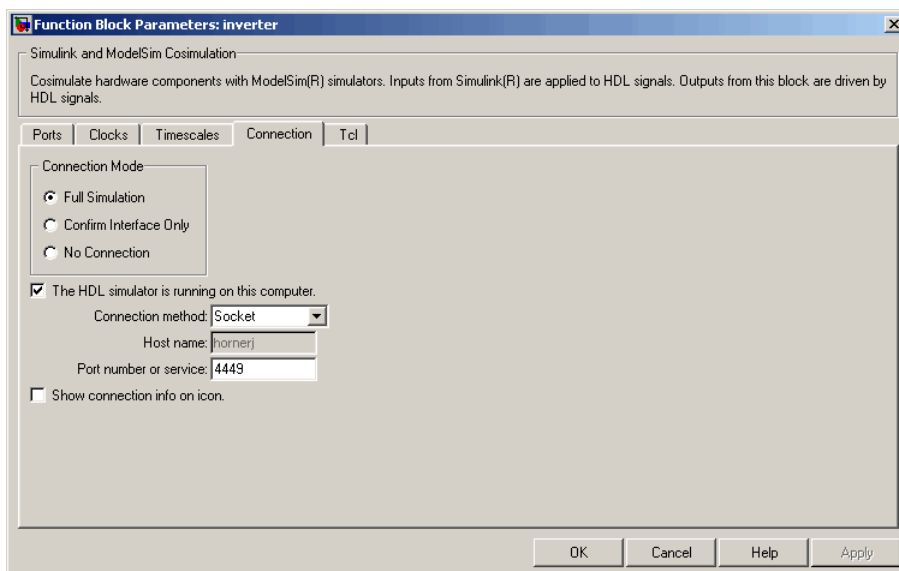
Now configure the parameters of the **Connection** pane by performing the following actions:

- 1 Click the **Connection** tab.
- 2 Leave **Connection Mode** as **Full Simulation**.
- 3 Select **socket** from the **Connection method** list. This option specifies that Simulink and ModelSim will communicate via a designated TCP/IP socket port. Observe that two additional fields, **Port number or service** and **Host name**, are now visible.

Note that, because the **The HDL simulator is running on this computer option** is selected by default, the **Host name** field is disabled. In this configuration, both Simulink and ModelSim execute on the same computer, so you do not need to enter a remote host system name.

- 4 In the **Port number or service** text box, enter socket port number 4449 or, if this port is not available on your system, another valid port number or service name. The model will use TCP/IP socket communication to link with ModelSim. Note what you enter for this parameter. You will specify the same socket port information when you set up ModelSim for linking with Simulink.

The **Connection** pane should appear as follows.

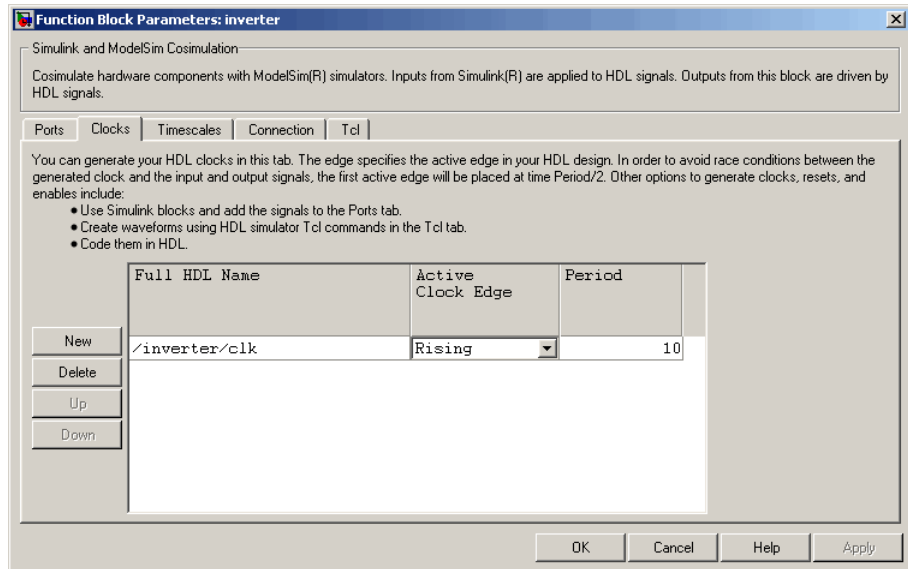


- 5 Click **Apply**.

Now configure the **Clocks** pane by performing the following actions:

- 1 Click the **Clocks** tab.
- 2 Click the **New** button. A new clock signal with an empty signal name is added to the signal list.
- 3 Double-click on the new signal name to edit. Enter the signal path `/inverter/clock`. Then select Rising from the **Edge** list. Set the **Period** parameter to 10.

**4** The **Clocks** pane should appear as follows.



**5** Click **Apply**.

Next, enter some simple Tcl commands to be executed before and after simulation, as follows:

**1** Click the **Tcl** tab.

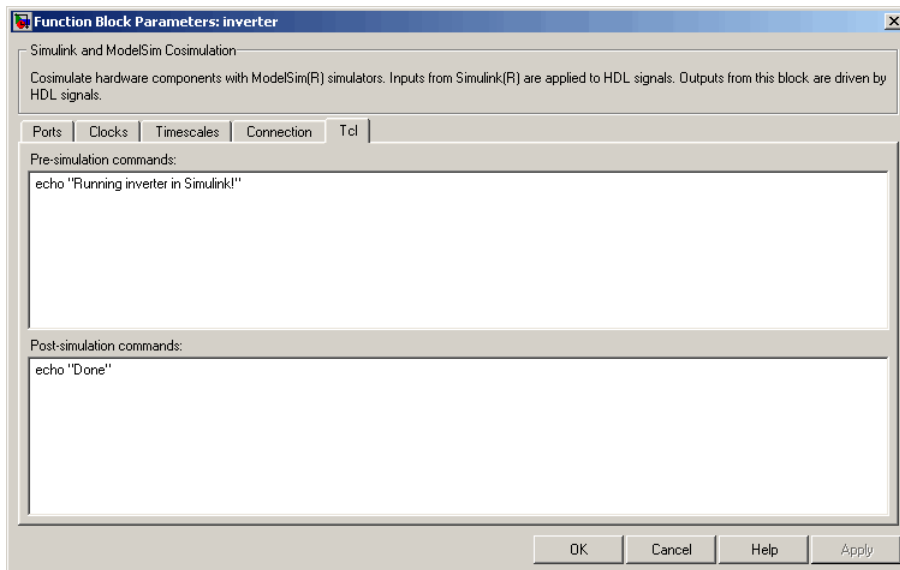
**2** In the **Pre-simulation commands** text box, enter the following Tcl command:

```
echo "Running inverter in Simulink!"
```

**3** In the **Post-simulation commands** text box, enter

```
echo "Done"
```

The **Tcl** pane should appear as follows.

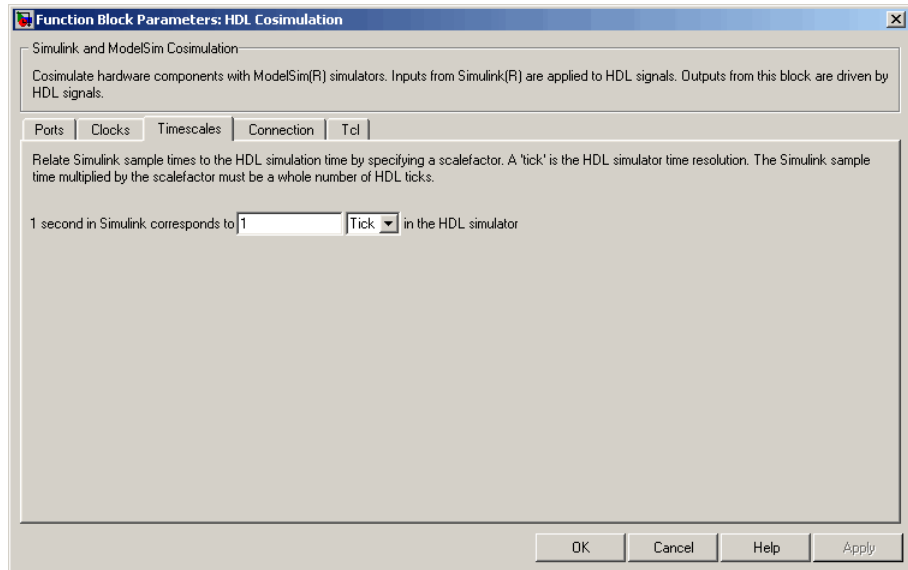


#### 4 Click **Apply**.

Next, view the **Timescales** pane to make sure it is set to its default parameters, as follows:

#### 1 Click the **Timescales** tab.

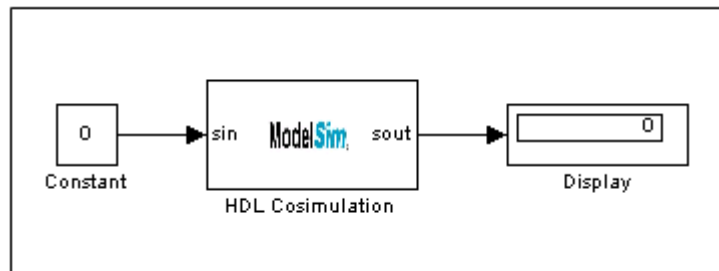
2 The default settings of the **Timescales** pane are shown below. These settings are required for correct operation of this example. See “Representation of Simulation Time” on page 3-15 for further information.



**3** Click **OK** to close the Function Block Parameters dialog box.

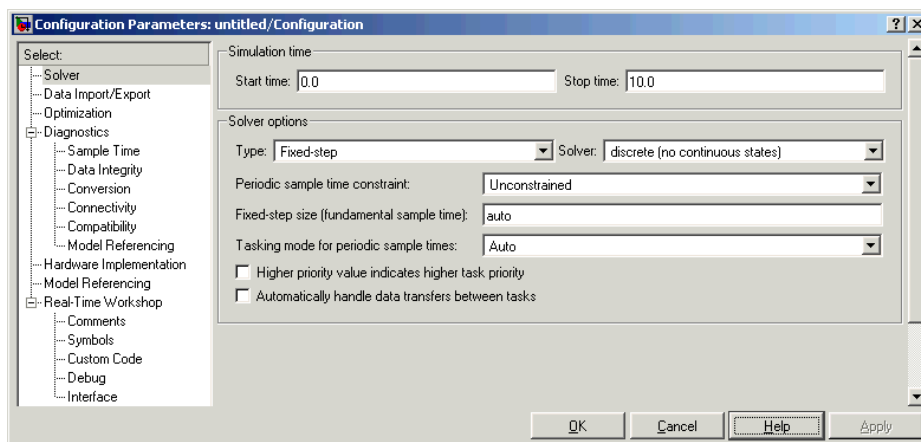
The final step is to connect the blocks, configure model-wide parameters, and save the model. Perform the following actions:

**1** Connect the blocks as shown below.



At this point, you might also want to consider adjusting block annotations.

- 2 Configure the Simulink solver options for a fixed-step, discrete simulation; this is required for correct cosimulation operation. Perform the following actions:
  - a Select **Configuration Parameters** from the **Simulation** menu in the model window. The Configuration Parameters dialog box opens, displaying the **Solver options** pane.
  - b Select Fixed-step from the **Type** menu.
  - c Select discrete (no continuous states) from the **Solver** menu.
  - d Click **Apply**. The **Solver options** pane should appear as shown below.



- e Click **OK** to close the Configuration Parameters dialog box.

See “Setting Simulink Software Configuration Parameters” on page 3-38 for further information on Simulink settings that are optimal for use with EDA Simulator Link MQ software.

- 3 Save the model.

## Setting Up ModelSim for Use with Simulink

You now have a VHDL representation of an inverter and a Simulink model that applies the inverter. To start ModelSim such that it is ready for use with Simulink, enter the following command line in the MATLAB Command Window:



```
vsim('socketsimulink', 4449)
```

---

**Note** If you entered a different socket port specification when you configured the HDL Cosimulation block in Simulink, replace the port number 4449 in the preceding command line with the correct socket port information for your model. The `vsim` function informs ModelSim of the TCP/IP socket to use for establishing a communication link with your Simulink model.

---

## Loading Instances of the VHDL Entity for Cosimulation with Simulink

This section explains how to use the `vsimulink` command to load an instance of your VHDL entity for cosimulation with Simulink. The `vsimulink` command is an EDA Simulator Link MQ variant of the ModelSim `vsim` command. It is made available as part of the ModelSim configuration.

To load an instance of the `inverter` entity, perform the following actions:

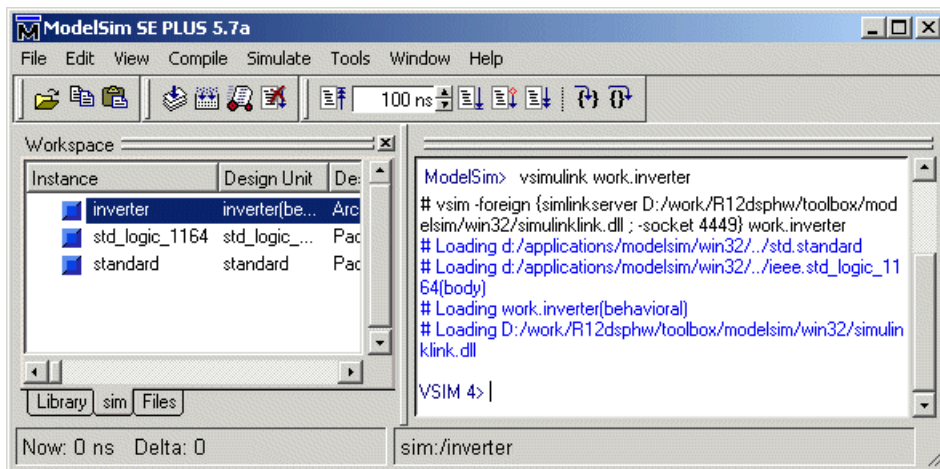
- 1 Change your input focus to the ModelSim window.
- 2 If necessary, change your directory to the location of your `inverter.vhd` file. For example:

```
ModelSim> cd C:/MyPlayArea
```

- 3 Enter the following `vsimulink` command:

```
ModelSim> vsimulink work.inverter
```

ModelSim starts the `vsim` simulator such that it is ready to simulate entity `inverter` in the context of your Simulink model. The ModelSim command window display should be similar to the following.



## Running the Simulation

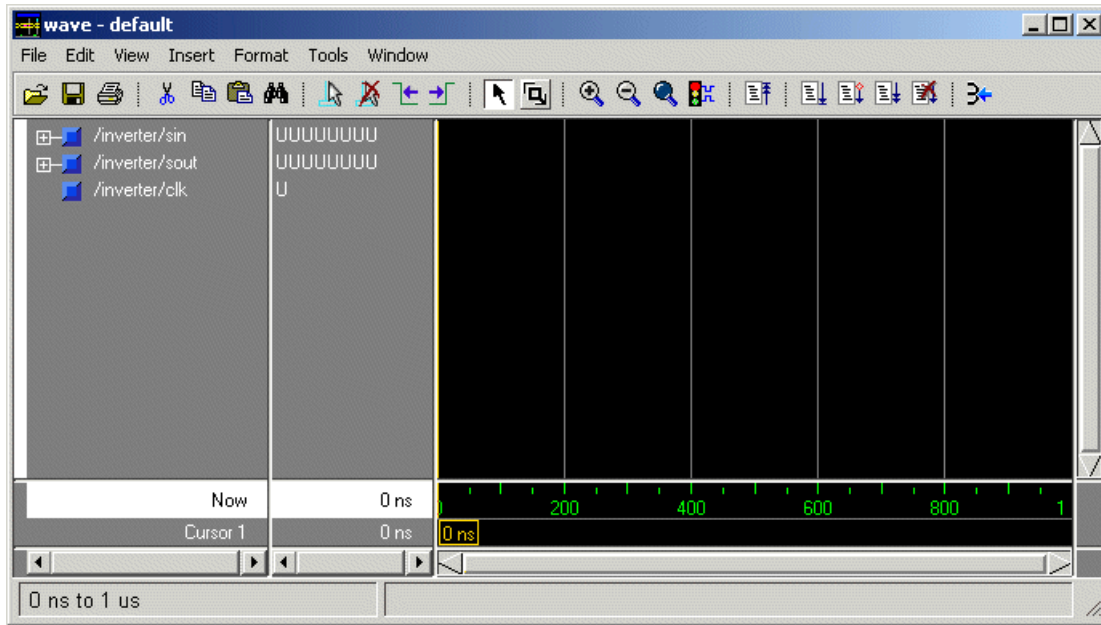
This section guides you through a scenario of running and monitoring a cosimulation session.

Perform the following actions:

- 1 Open and add the inverter signals to a **wave** window by entering the following ModelSim command:

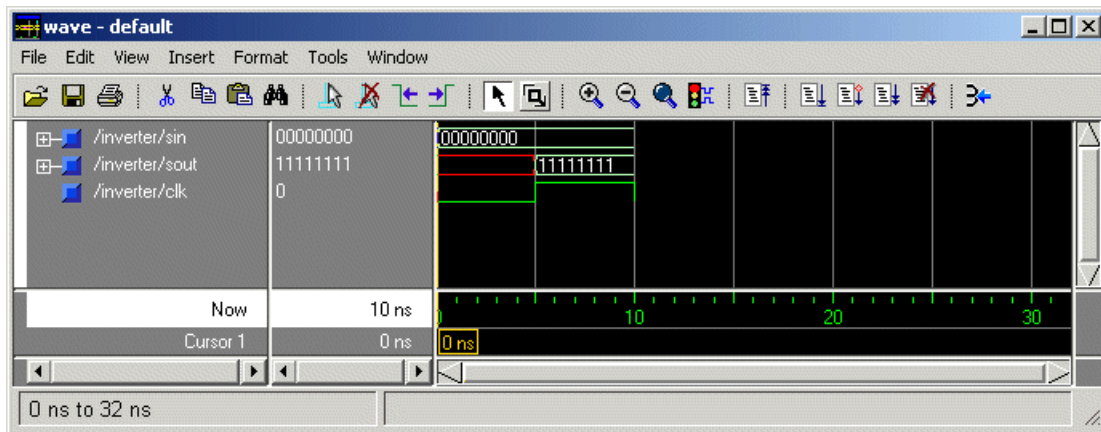
```
VSIM n> add wave /inverter/*
```

The following **wave** window appears.

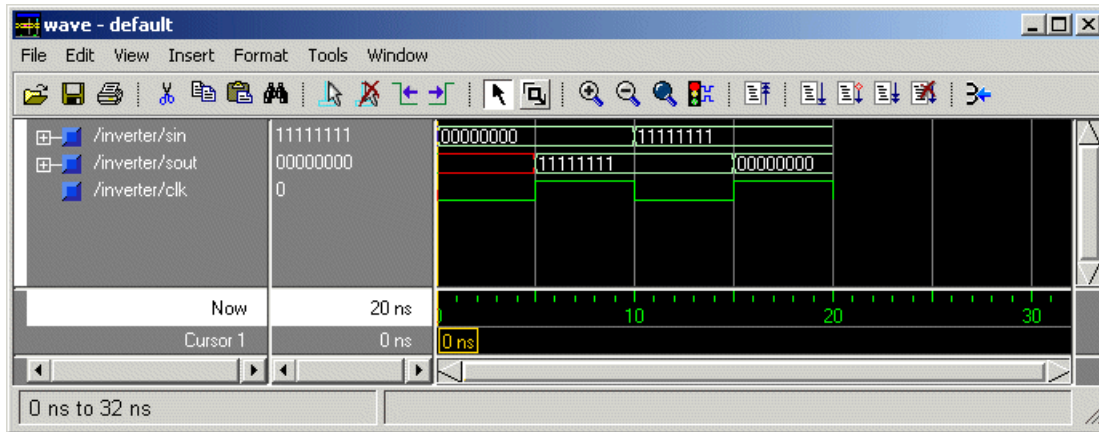


**2** Change your input focus to your Simulink model window.

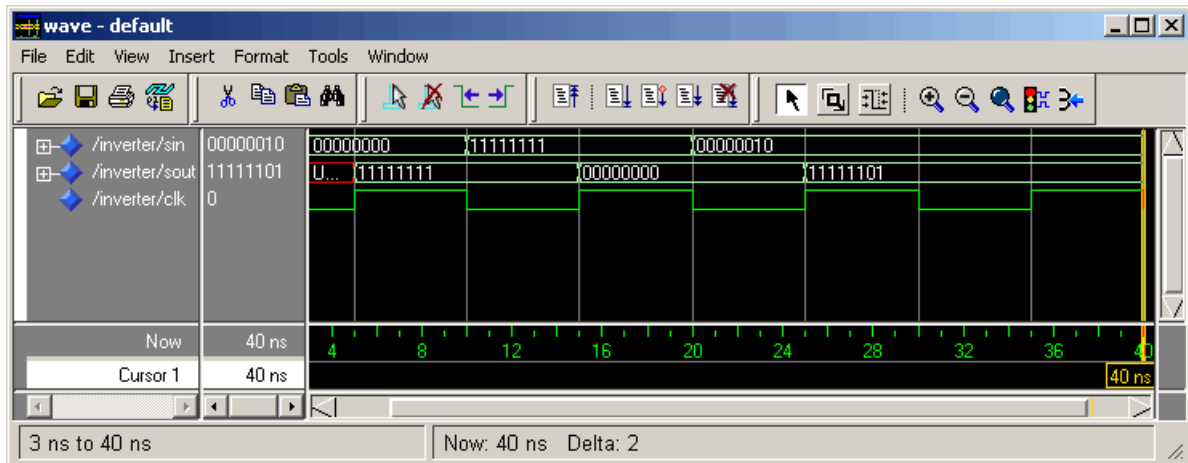
**3** Start a Simulink simulation. The value in the Display block changes to 255. Also note the changes that occur in the ModelSim **wave** window. You might need to zoom in to get a better view of the signal data.



- 4 In the Simulink model, change **Constant value** to 255, save the model, and start another simulation. The value in the Display block changes to 0 and the ModelSim **wave** window is updated as follows.



- 5 In the Simulink Model, change **Constant value** to 2 and **Simulation time** to 20 and start another simulation. This time, the value in the Display block changes to 253 and the ModelSim **wave** window appears as shown below.



Note the change in the sample time in the **wave** window.

## Shutting Down the Simulation

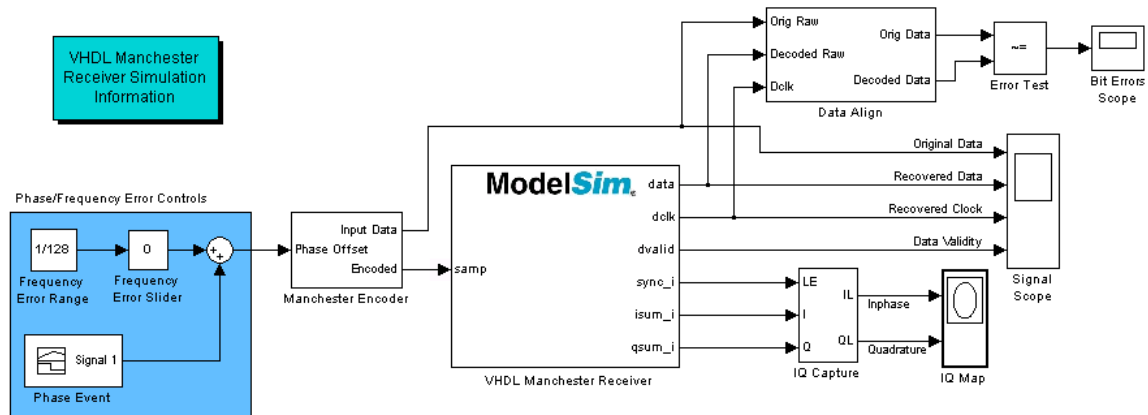
This section explains how to shut down a simulation in an orderly way, as follows:

- 1** In ModelSim, stop the simulation by selecting **Simulate > End Simulation**.
- 2** Quit ModelSim.
- 3** Close the Simulink model window.

### toVCD Block Tutorial

VCD files include data that can be graphically displayed or analyzed with postprocessing tools. An example of such a tool is the ModelSim `vcd2wlf` tool, which converts a VCD file to a WLF file that you can then view in a ModelSim **wave** window. This section shows how you might apply the `vcd2wlf` tool, as follows:

- 1 Place a copy of the Manchester Receiver Simulink demo `manchestermode1.mdl` in a writable directory.
- 2 Open your writable copy of the Manchester Receiver model. For example, select **File > Open**, select the file `manchestermode1.mdl` and click **Open**. The Simulink model should appear as follows.



Before running this model you must first launch ModelSim.  
You can launch ModelSim on this computer using either a  
shared memory link or a TCP/IP socket link.

#### Shared memory link:

- 1) Be sure that the 'Connection' tab of the Cosimulation block dialog is set as follows:  
     'ModelSim running on this computer' is checked  
     and 'Shared memory' is selected
- 2) Execute the following MATLAB command:  
     `vsim('tclstart',manchestercmds)`
- 3) Start the Simulink simulation.

```
vsim('tclstart',manchestercmds)
%Double-click here to launch a new ModelSim
```

ModelSim Startup Command(Shared Memory)

#### TCP/IP socket link:

- 1) Be sure that the 'Connection' tab of the Cosimulation block dialog is set as follows:  
     'ModelSim running on this computer' is checked  
     and 'Socket' is selected  
     'Port number or service' matches the port number used  
     in the command below.
- 2) Execute the following MATLAB command:  
     `vsim('tclstart',manchestercmds,'socketsimulink',4442)`
- 3) Start the Simulink simulation.

```
vsim('tclstart',manchestercmds,'socketsimulink',4442)
%Double-click here to launch a new ModelSim
```

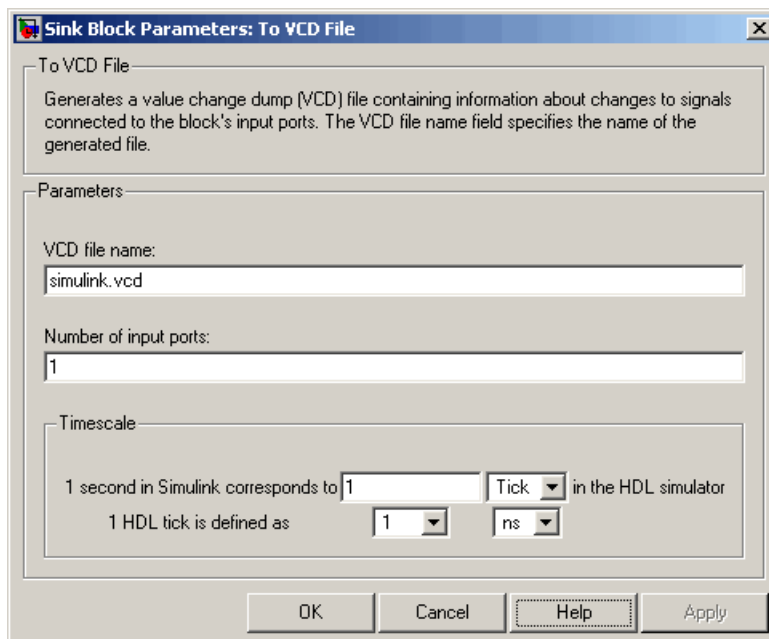
ModelSim Startup Command(TCP/IP Socket)

### 3 Open the Library Browser.

### 4 Replace the Signal Scope block with a To VCD File block, as follows:

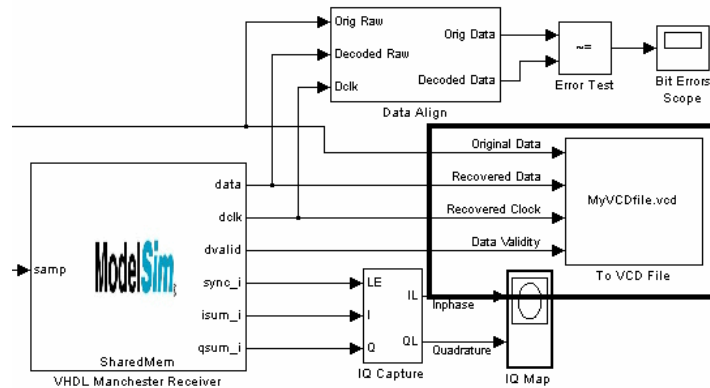
- a Delete the Signal Scope block. The lines representing the signal connections to that block change to red dashed lines, indicating the disconnection.
- b Find and open the EDA Simulator Link™ MQ block library.

- c Copy the To VCD File block from the Library Browser to the model by clicking the block and dragging it from the browser to the location in your model window previously occupied by the Signal Scope block.
- d Double-click the To VCD File block icon. The Block Parameters dialog appears.



- e Type MyVCDfile.vcd in the **VCD file name** text box.
  - f Type 4 in the **Number of input ports** text box.
  - g Click **OK**. Simulink applies the new parameters to the block.
- 5 Connect the signals Original Data, Recovered Data, Recovered Clock, and Data Validity to the block ports. The following display highlights the modified area of the model.





6 Save the model.

7 Select the following command line from the instructional text that appears in the demonstration model:

```
vsim('tclstart',manchestercmds,'socketsimulink',4442)
```

8 Paste the command in the MATLAB Command Window and execute the command line. This command starts ModelSim and configures it for a Simulink cosimulation session.

---

**Note** You might need to adjust the TCP/IP socket port. The port you specify in the `vsim` command must match the value specified for the HDL Cosimulation block. To check the port setting for that block, double-click the block icon and then select the **Connection** tab in the Block Parameters dialog.

---

9 Start the simulation from the Simulink model window.

10 When the simulation is complete, locate, open, and browse through the generated VCD file, `MyVCDfile.vcd`.

11 Close the VCD file.

12 Change your input focus to ModelSim and end the simulation.

- 13** Change the current directory to the directory containing the VCD file and enter the following command at the ModelSim command prompt:

```
vcd2wlf MyVCDfile.vcd MyVCDfile.wlf
```

The `vcd2wlf` utility converts the VCD file to a WLF file that you display with the command `vsim -view`.


- 14** In ModelSim, open the wave file `MyVCDfile.wlf` as data set `MyVCDwlf` by entering the following command:

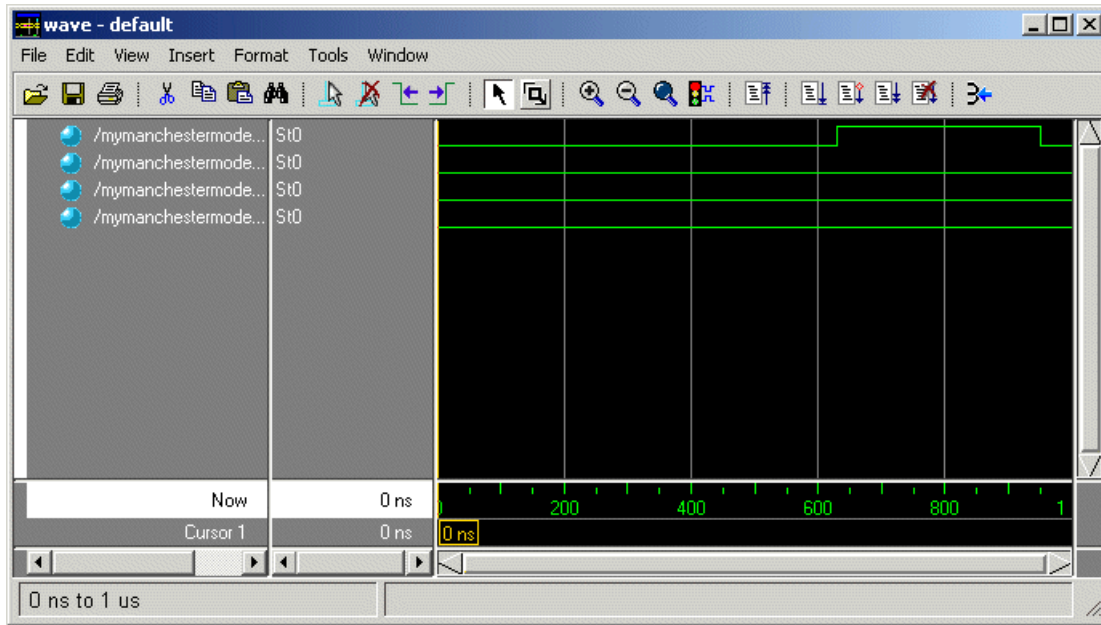
```
vsim -view MyVCDfile.wlf
```

- 15** Open the `MyVCDwlf` data set with the following command:

```
add wave MyVCDfile:/*
```

A **wave** window appears showing the signals logged in the VCD file.

- 16** Click the Zoom Full button  to view the signal data. The **wave** window should appear as follows.



**17** Exit the simulation. One way of exiting is to enter the following command:

```
dataset close MyVCDfile
```

ModelSim closes the data set, clears the **wave** window, and exits the simulation.

For more information on the `vcd2w1f` utility and working with data sets, see the ModelSim documentation.



# EDA Simulator Link™ MQ MATLAB® Function Reference

---

# configuremodelsim

---

**Purpose** Configure ModelSim for use with EDA Simulator Link™ MQ

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

## Description

---

**Note** `configuremodelsim` has been replaced by the guided setup script (`syscheckmq`) for configuring your simulator setup. Although `configuremodelsim` supported for backward compatibility, it is recommended that you use the setup script instead. See “Setting Up the HDL Simulator for Use with the Link Software” on page 1-12.

---

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

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

When it is used without any arguments, `configuremodelsim` prompts you to either let `configuremodelsim` find the installed ModelSim executable or provide the path to the ModelSim installation you want to use. If no previous configuration was performed (no Tcl DO file), `configuremodelsim` creates a new `ModelSimTclFunctionsForMATLAB.tcl` script in the `tcl` directory 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” on page 4-4.

After you call this function, ModelSim is ready to use EDA Simulator Link MQ when ModelSim is invoked from outside of MATLAB. You can use Chapter 5, “EDA Simulator Link™ MQ Command Extensions for the HDL Simulator Reference” from the ModelSim environment 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.

## Usage Notes

`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 is already on the system path, and, if it is 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`.

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 (this setup is done instead 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 desire.

# configuremodelsim

---

---

**Note** You need to run `configuremodelsim` only once to set the location of the Tcl DO file (you may run `configuremodelsim` multiple times to add additional Tcl commands to the Tcl startup DO file).

---

---

**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.

---

## 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 directory.

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.

'action', 'uninstall'

Removes the EDA Simulator Link MQ configuration from the ModelSim startup DO file. The contents of `ModelSimTclFunctionsForMATLAB.tcl` are replaced 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 directory.

'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 directory under this ModelSim installation.

'vsimdir', 'pathname'

Specifies where to put the Tcl script containing EDA Simulator Link MQ Tcl commands. This script is programmatic only; if no directory is specified 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 directory under this ModelSim installation.

## Examples

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

```
configuremodelsim
```

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

```
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\Modelstech_6.0e\win32
```

# configuremodelsim

---

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

When `configuremodelsim` is run on an existing configuration, the dialog looks like this:

```
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:

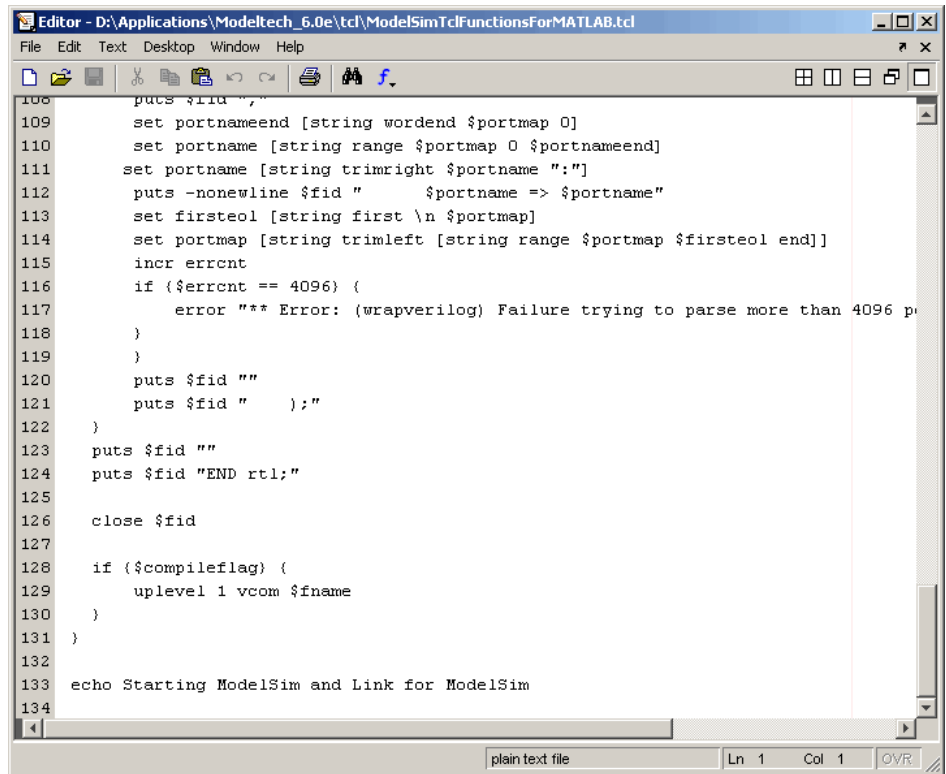
```
configuremodelsim('tclstart','echo Starting ModelSim and EDA Simulator Link MQ')

vsimoptions =

echo Starting ModelSim and EDA Simulator Link MQ

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, as shown in the following graphic.



```
Editor - D:\Applications\Modeltech_6.0e\tcl\ModelSimTclFunctionsForMATLAB.tcl
File Edit Text Desktop Window Help
puts $fid "
109 set portnameend [string wordend $portmap 0]
110 set portname [string range $portmap 0 $portnameend]
111 set portname [string trimright $portname ":"]
112 puts -nonewline $fid "      $portname => $portname"
113 set firsteol [string first \n $portmap]
114 set portmap [string trimleft [string range $portmap $firsteol end]]
115 incr errcnt
116 if {$errcnt == 4096} {
117     error "*** Error: (wrapverilog) Failure trying to parse more than 4096 p
118 }
119 }
120 puts $fid ""
121 puts $fid "    );"
122 }
123 puts $fid ""
124 puts $fid "END rtl;"
125
126 close $fid
127
128 if {$compileflag} {
129     uplevel 1 vcom $fname
130 }
131 }
132
133 echo Starting ModelSim and Link for ModelSim
134
```

The following example shows removing the EDA Simulator Link MQ 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
```

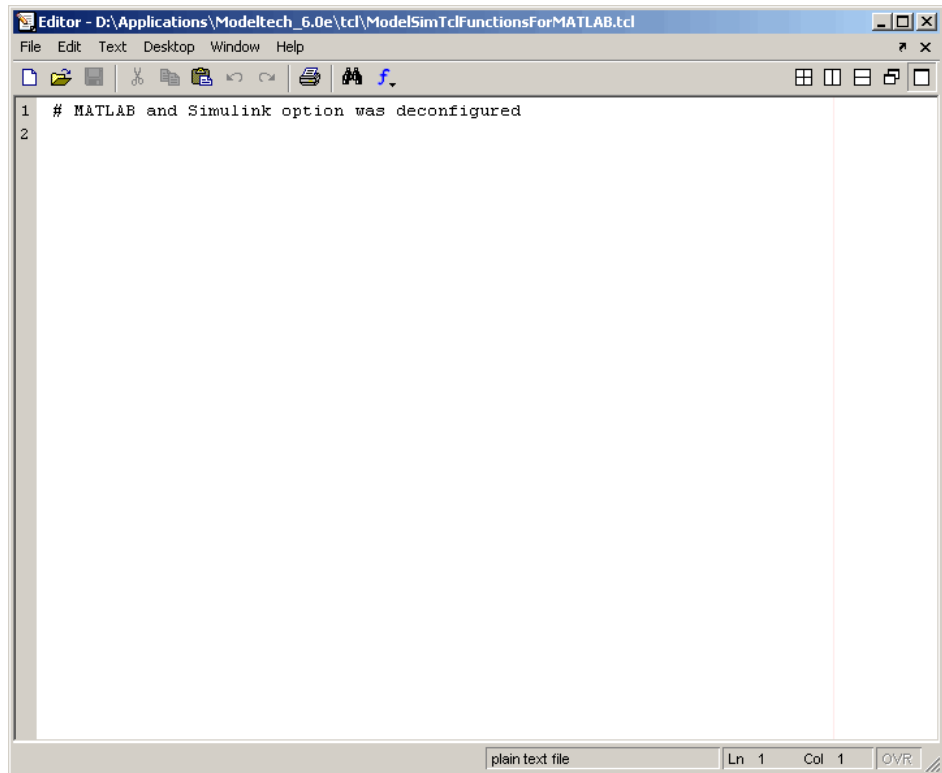
```
Please enter the path to your Modelsim executable file (modelsim.exe or vsim.exe):
```

```
D:\Applications\Modeltech_6.0e\win32
```

# configuremodelsim

```
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`

# hdldaemon

---

**Purpose** Start MATLAB server component of EDA Simulator Link™ MQ interface

**Syntax**

```
hdldaemon
hdldaemon('PropertyName', 'PropertyValue'...)
hdldaemon('status')
hdldaemon('kill')
```

**Description** **Server Activation**

hdldaemon starts the MATLAB server component of the EDA Simulator Link MQ software with the following default settings:

- Shared memory communication enabled
- Time resolution for the MATLAB simulation function output ports set to scaled (type double)

Although you can use TCP/IP on a single system (one that is running both MATLAB and the HDL simulator), using shared memory communication when your application configuration consists of a single system can result in increased performance.

Only one hdldaemon per MATLAB session can be running at any given time.

### Matching Communication Modes and Socket Ports

The communication mode that you specify (shared memory or TCP/IP sockets) must match what you specify for the communication mode when you issue the matlabcp, matlabb, or matlabbbeval command in the HDL simulator.

In addition, if you specify TCP/IP socket mode, you must also identify a socket port to be used for establishing links. You can choose and specify a socket port yourself, or you can use an option that instructs the operating system to identify an available socket port for you. Regardless of how the socket port is identified, the

socket you specify with the HDL simulator must match the socket being used by the server.

For more information on modes of communication, see “Communicating with MATLAB or Simulink and the HDL Simulator” on page 1-8. For more information on establishing the HDL simulator end of the communication link, see “Associating a MATLAB® Link Function with an HDL Module” on page 2-37.

`hdldaemon('PropertyName', 'PropertyValue'...)` starts the EDA Simulator Link MQ MATLAB server component with property-value pair settings that specify the communication mode for the link between MATLAB and the HDL simulator, the resolution of `now` (the current time argument passed by the associated m-function), and, optionally, a Tcl command to be executed immediately in the HDL simulator. See `hdldaemon` “Property Name/Property Value Pairs” on page 4-12 for details.

## Link Status

`hdldaemon('status')` returns the following message indicating that a link (connection) exists between MATLAB and the HDL simulator:

```
HDLDaemon socket server is running on port 4449 with 0 connections
```

You can also use this function to check on the communication mode being used, the number of existing connections, and the interprocess communication identifier (`ipc_id`) being used for a link by assigning the return value of `hdldaemon` to a variable. The `ipc_id` identifies a port number for TCP/IP socket links or the file system name for a shared memory communication channel. For example:

```
x=hdldaemon('status')
x =
      comm: 'sockets'
  connections: 0
      ipc_id: '4449'
```

This function call indicates that the server is using TCP/IP socket communication with socket port 4449 and is running with no active HDL simulator clients. If a shared memory link is in use, the value of `comm` is 'shared memory' and the value of `ipc_id` is a file system name for the shared memory communication channel. For example:

```
x=hdldaemon('status')
HDLDaemon shared memory server is running with 0 connections
x =
    comm: 'shared memory'
 connections: 0
 ipc_id: [1x45 char]
```

## Server Shutdown

`hdldaemon('kill')` shuts down the MATLAB server without shutting down MATLAB.

## Property Name/ Property Value Pairs

The following property name/property value pairs are valid for `hdldaemon`:

'socket', `tcp_spec`

Specifies the TCP/IP socket mode of communication for the link between MATLAB and the HDL simulator. If you omit this argument, the server uses the shared memory mode of communication.

---

**Note** You *must* use TCP/IP socket communication when your application configuration consists of multiple computing systems.

---

The `tcp_spec` can be a TCP/IP port number, TCP/IP port alias or service name, or the value zero, indicating that the port is to be assigned by the operating system. See “Specifying TCP/IP Values” on page D-5 for some valid `tcp_spec` examples.



For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports” on page D-2.

---

**Note** If you specify the operating system option ('0' or 0), use `hdldaemon('status')` to acquire the assigned socket port number. You must specify this port number when you issue a link request with the `matlabtb` or `matlabtbeval` command in the HDL simulator.

---

`'time', 'sec' | 'time', 'int64'`

Specifies the time resolution for MATLAB function ports and simulation times (`tnow`).

**Specify...**

`'time' 'sec'`  
(default)

`'time' 'int64'`

**For...**

A double value that is scaled to seconds based on the current HDL simulation resolution

64-bit integer representing the number of simulation steps

If you omit this argument, the server uses scaled resolution time.

`'tclcmd', 'command'`

Passes a TCL command string, to be executed in the HDL simulator, from MATLAB to the HDL simulator. You may use a compound command and separate the commands with semicolons.

---

**Note** 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 `run`, `stop`, or `reset`.

---

'quiet', 'true'

Suppresses printing messages to the standard queue. Errors are still shown.

The following table provides guidelines on when and how to specify property name/property value pairs.

**If Your Application Is to... Do the Following...**

Operate in shared memory mode

Omit the 'socket', *tcp\_spec* property name/property value pair. The interface operates in shared memory mode by default. You should use shared memory mode if your application configuration consists of a single system and uses a single communication channel.

Operate in TCP/IP socket mode, using a specific TCP/IP socket port

Specify the 'socket', *tcp\_spec* property name and value pair. The *tcp\_spec* can be a socket port number or service name. Examples of valid port specifications include '4449', 4449, and MATLAB Service. For information on choosing a TCP/IP socket port, see “Choosing TCP/IP Socket Ports” on page D-2.

Operate in TCP/IP socket mode, using a TCP/IP socket that the operating system identifies as available

Specify 'socket', 0 or 'socket', '0'.

Return time values in seconds (type double)

Specify 'time', 'sec' or omit the parameter. This is the default time value resolution.

Return 64-bit time values (type int64)

Specify 'time', 'int64'.

## If Your Application Is to... Do the Following...

Execute Tcl command immediately upon simulator connection

Specify the 'tclcmd', 'command' property name and value pair. Command must be a valid Tcl command but cannot include commands that load an HDL simulator project or modify the simulator state.

Suppress server shutdown message when using hdldaemon to get an unused socket number (message can appear confusing)

Specify 'quiet', 'true'.

## Examples

The following function call starts the MATLAB server with shared memory communication enabled and a 64-bit time resolution format for the MATLAB function's tnow parameter:

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

The following function call starts the MATLAB server with TCP/IP socket communication enabled on socket port 4449. Although it is not necessary to use TCP/IP socket communication on a single-computer application, you can use that mode of communication locally. A time resolution is not specified. Thus, the default, scaled simulation time resolution is applied to the MATLAB function's output ports:

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

The following function call starts the MATLAB server with TCP/IP socket communication enabled on port 4449. A 64-bit time resolution format is also specified:

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

You also can start the server from a script. Consider the following function call sequence:

```
dstat = hdldaemon('socket', 0)
portnum = dstat.ipc_id
```

The first call to `hdldaemon` specifies that the server use TCP/IP communication with a port number that the operating system identifies and returns connection status information, including the assigned port number, to `dstat`. The statement on the second line assigns the socket port number to `portnum` for future reference.

The following function call causes the string `This is a test` to be displayed at the HDL simulator prompt:

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

The following is an example of a compound Tcl command used with `hdldaemon`:

```
hdldaemon('tclcmd',{force filter2d_v.clk_enable 1
-after 0ns;
force filter2d_v.reset 1 -after 0 ns 0 -after 1 ns;
puts {Running Simulink Cosimulation block};
puts [clock format [clock seconds]]'})
```

---

<b>Purpose</b>	Convert multivalued logic to decimal
<b>Syntax</b>	<pre>mvl2dec('multivalued_logic_string') mvl2dec('multivalued_logic_string', signed)</pre>
<b>Description</b>	<p><code>mvl2dec('multivalued_logic_string')</code> converts a multivalued logic string <i>multivalued_logic_string</i> to a positive decimal. If <i>multivalued_logic_string</i> contains any character other than '0' or '1', NaN is returned. <i>multivalued_logic_string</i> must be a vector.</p> <p><code>mvl2dec('multivalued_logic_string', signed)</code> converts a multivalued logic string <i>multivalued_logic_string</i> to a positive or a negative decimal. If <i>signed</i> is true, this function assumes the first character <i>multivalued_logic_string</i>(1) to be a signed bit of a 2s complement number. If <i>signed</i> is missing or false, the multivalued logic string is converted to a positive decimal.</p>
<b>Examples</b>	<p>The following function call returns the decimal value 23:</p> <pre>mvl2dec('010111')</pre> <p>The following function call returns NaN:</p> <pre>mvl2dec('xxxxxx')</pre> <p>The following function call returns the decimal value -9:</p> <pre>mvl2dec('10111', true)</pre>
<b>See Also</b>	<code>dec2mvl</code>

# pingHdlSim

---

**Purpose** Block cosimulation until HDL simulator is ready

**Syntax**

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

**Description** pingHdlSim(timeout) blocks cosimulation by not returning until the Simulink server is loaded or until the specified timeout occurs. pingHdlSim returns the process ID of the HDL simulator or -1 if a timeout occurs. You must enter a timeout value.

This function is useful if you are trying to automate a cosimulation and you need to know that the Simulink 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 Simulink server is loaded 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 Simulink server is loaded or until 20 seconds have passed:

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

The following function call blocks further cosimulation on port 5678 on hostname msuser until the Simulink server is loaded or until 20 seconds have passed:

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

# tclHdlSim

---

**Purpose** Execute Tcl command in HDL simulator

**Syntax**

```
tclHdlSim(tclCmd)
tclHdlSim(tclCmd, 'portNumber')
tclHdlSim(tclCmd, 'portnumber', 'hostname')
```

**Description**

tclHdlSim(tclCmd) executes a Tcl command on the HDL simulator using a shared connection.

tclHdlSim(tclCmd, 'portNumber') executes a Tcl command on the HDL simulator by connecting to the local host on port *portNumber*.

tclHdlSim(tclCmd, 'portnumber', 'hostname') executes a Tcl command on the HDL simulator by connecting to the host *hostname* on port *portname*.

The HDL simulator must be connected to MATLAB using the EDA Simulator Link™ MQ software for this function to work (see “Starting the HDL Simulator” on page 1-23. If you start from within MATLAB, you must use vsimmatlab.).

**Examples**

The following function call displays a message in the HDL simulator command window using port 5678 on hostname msuser:

```
tclHdlSim('puts "Done"', '5678', 'msuser')
```



**Purpose** Start and configure ModelSim for use with EDA Simulator Link™ MQ

**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 MQ. The first directory in ModelSim matches your MATLAB current directory.

After you call this function, you can use EDA Simulator Link MQ command extensions 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** '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.

'startupfile', 'pathname'

Specifies a DO macro file that defines the behavior of the ModelSim commands `vsimmatlab` and `vsimulink`. The DO file consists of some general-purpose Tcl commands for launching ModelSim and any commands you specify with the `'tclstart'` property. If you omit this property, the function creates a temporary file that is overwritten each time ModelSim starts. If you specify a name for the DO file, later you can use the file to start ModelSim from the command line as shown in the following syntax:

```
vsim -gui -do do_file
```

'rundir', 'dirname'

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

The following conditions apply to this name/value pair:

- If `dirname` is specified and the directory exists, ModelSim is run in the specified directory.
- If no `rundir` property/value pair is specified or if `dirname` is empty, ModelSim is run in the current working directory.
- If the value of `dirname` is “TEMPDIR”, the function creates a temporary directory in which it runs ModelSim.
- If `dirname` is specified and the directory does *not* exist, you will get an error.

'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” on page D-5 for some valid `tcp_spec` examples.

For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports” on page D-2

If ModelSim and Simulink are running 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 omit `-socketsimulink tcp_spec` from the function call.

---

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

---

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

Determines whether ModelSim is launched from vsim. The default is yes, which launches ModelSim and creates a startup Tcl file. If `startms` is set to no, ModelSim is not launched, but a startup Tcl file is still created.

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™ MQ Libraries” on page 1-18.

`'libdir', 'directory'`

Specifies the directory containing MATLAB library files. This property creates an entry in the startup Tcl file that points to the directory with the libraries needed for ModelSim to communicate with MATLAB when ModelSim is running on a machine that does not have MATLAB.

## Examples

The following function call sequence changes the directory 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 directory. 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 directory to ModelSim project directory
vsim('tclstart','vsimmatlab work.parse; matlabtb parse 10 ns -socket 4449')
```

The following function call sequence changes the directory 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 directory. 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 directory to ModelSim project directory
vsim('tclstart','vsimulink work.parse','socketsimulink','4449')
```

EDA Simulator Link™  
MQ Command Extensions  
for the HDL Simulator  
Reference

---

# matlabcp

---

**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>]
```

**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. Alternatively, you can specify a different MATLAB function with -mfunc.

---

**Note** Note Do not specify an instance of an HDL module that has already been associated with a MATLAB test bench function (via matlabcp or matlabb). 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:

---

<code>&lt;timen&gt;,...</code>	Specifies one or more discrete time values at which the specified MATLAB function is called. Each time value is relative to the current simulation time. The MATLAB function is always called once at the start of the simulation, even if you do not specify a time. Multiple time values are separated by a space, for example: <code>matlabcp vlogtestbench_top 1e6 fs 3 2e3 ps -repeat 3 ns -cancel 7ns</code>
<code>-repeat &lt;time&gt;</code>	Specifies that the MATLAB function be called repeatedly based on the specified <code>&lt;timen&gt;,...</code> pattern. The time values are relative to the value of <code>tnow</code> at the time the MATLAB function is first called.
<code>-cancel &lt;time&gt;</code>	Specifies a time at which the specified MATLAB function stops executing. The time value is relative to the value of <code>tnow</code> at the time the MATLAB function is first called. If you do not specify a cancel time, the command calls the MATLAB function.

---

**Note** Time specifications must be placed after the `matlabcp` instance and before any additional command arguments; otherwise the time specifications are ignored.

---

All time specifications for the `matlabcp` functions are represented by 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. For TCP/IP socket communication on a single computer, the <tcp\_spec> argument 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 that is running the MATLAB server (hdldaemon). See “Specifying TCP/IP Values” on page D-5 for some valid tcp\_spec examples.

For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports” on page D-2.

If the HDL simulator and MATLAB are running 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 omit -socket <tcp\_spec> from 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 “Communicating with MATLAB or Simulink and the HDL Simulator” on page 1-8. For more information on establishing the MATLAB end of the communication link, see “Starting the MATLAB Server” on page 2-49.



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

Indicates that the specified MATLAB function is called on the rising edge (transition from '0' to '1') of any of the specified signals. For determining signal transition in:

- Verilog: Z and X are read as 0
- VHDL: Z and X will not create a rate transition

Rate transitions are only from 0 -> 1 and 1-> 0. Specify -rising with the path names of one or more signals defined as a logic type.

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

Indicates that the specified MATLAB function is called when 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.

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

Indicates that the specified MATLAB function is called when any of the specified signals changes state. Specify -sensitivity with the path names of one or more signals. Signals in the sensitivity list can be any type and can be at any level in the hierarchy of the HDL model.

-mfunc <name>

The name of the MATLAB function that is associated with the HDL module instance you specify for instance. If you omit this argument, matlabcp associates the HDL module instance to a MATLAB function that has the same name as the HDL instance. For example, if the HDL module instance is myfirfilter, matlabcp associates the HDL module instance with the MATLAB function myfirfilter. If you omit this argument and matlabcp does not find a MATLAB function with the same name, the command generates an error message.

## Description

The `matlabcp` command has the following characteristics:

- Starts the HDL simulator client component of the EDA Simulator Link™ MQ 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.

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 “Coding an EDA Simulator Link™ MQ MATLAB® Application” on page 2-4.

---

**Note** For the HDL simulator to establish a communication link with MATLAB, the MATLAB server, `hdldaemon`, must be running with the same communication mode and, if appropriate, the same TCP/IP socket port as you specify with the `matlabcp` command.

---

## Examples

This example associates the Verilog module `vlogtestbench_top.u_matlab_component` with the MATLAB function `vlogmatlabcp` using socket communication on port 4449. The `'-mfunc'` option specifies the m-function to connect to and `'-socket'` option specifies the port number for socket connection mode.

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

This example also associates the component with the MATLAB function but includes explicit times and uses the `-cancel` option.

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

This example also associates the component with the MATLAB function and also uses rising and falling edges.

```
matlabcp vlogtestbench_top 1 2 3 4 5 6 7 -rising outclk3 -falling
```

# 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>]
```

**Arguments**

<instance>  
Specifies the instance of an HDL module that is to be associated 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`. Alternatively, you can specify a different MATLAB function with `-mfunc`.

---

**Note** Note Do not specify an instance of an HDL module that has already been associated with a MATLAB component 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:

---

<code>&lt;timen&gt;,...</code>	Specifies one or more discrete time values at which the specified MATLAB function is called. Each time value is relative to the current simulation time. Even if you do not specify a time, the command calls the MATLAB function once at the start of the simulation. Multiple time values are separated by a space, for example: <code>matlabtb vlogtestbench_top 1e6 fs 3 2e3 ps -repeat 3 ns -cancel 7ns</code>
<code>-repeat &lt;time&gt;</code>	Specifies that the MATLAB function be called repeatedly based on the specified <code>&lt;timen&gt;,...</code> pattern. The time values are relative to the value of <code>tnow</code> at the time the MATLAB function is first called.
<code>-cancel &lt;time&gt;</code>	Specifies a time at which the specified MATLAB function stops executing. The time value is relative to the value of <code>tnow</code> at the time the MATLAB function is first called. If you do not specify a cancel time, the command calls the MATLAB function.

---

**Note** Time specifications must be placed after the `matlabtb` instance and before any additional command arguments; otherwise the time specifications are ignored.

---

All time specifications for the `matlabtb` functions are represented by 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. 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 that is running the MATLAB server (hdldaemon). See “Specifying TCP/IP Values” on page D-5 for some valid tcp\_spec examples.

For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports” on page D-2.

If the HDL simulator and MATLAB are running 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 omit -socket <tcp\_spec> from 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 “Communicating with MATLAB or Simulink and the HDL Simulator” on page 1-8. For more information on establishing the MATLAB end of the communication link, see “Starting the MATLAB Server” on page 2-49.

---

- rising <signal>[, <signal>...]  
Indicates that the specified MATLAB function is called 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.
- falling <signal>[, <signal>...]  
Indicates that the specified MATLAB function is called when 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.
- sensitivity <signal>[, <signal>...]  
Indicates that the specified MATLAB function is called when any of the specified signals changes state. Specify -sensitivity with the path names of one or more signals. Signals in the sensitivity list can be any type and can be at any level of the HDL design.
- 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. For example, if the HDL module instance is myfirfilter, matlabtb associates the HDL module instance with the MATLAB function myfirfilter. If you omit this argument and matlabtb does not find a MATLAB function with the same name, the command generates an error message.

## Description

The matlabtb command has the following characteristics:

- Starts the HDL simulator client component of the EDA Simulator Link™ MQ 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.

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

---

**Note** For the HDL simulator to establish a communication link with MATLAB, the MATLAB server, `hdldaemon`, must be running with the same communication mode and, if appropriate, the same TCP/IP socket port as you specify with the `matlabtb` command.

---

## Examples

The following command starts the ModelSim client component of EDA Simulator Link MQ, 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:

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

The following command starts the ModelSim client component of EDA Simulator Link MQ, 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 signal `\work\fclk` experiences a rising edge, and each time signal `\work\din` changes state.

```
vsim> matlabtb myfirfilter 10 ns -rising \work\fclk  
-sensitivity \work\din -socket 4449@compa
```



<b>Purpose</b>	Call specified MATLAB function once and immediately on behalf of instantiated HDL module
<b>Syntax</b>	<code>matlabtbeval &lt;instance&gt; [-socket &lt;tcp_spec&gt;] [-mfunc &lt;name&gt;]</code>
<b>Arguments</b>	<p><code>&lt;instance&gt;</code> Specifies the instance of an HDL module that is associated with a MATLAB function. By default, <code>matlabtbeval</code> 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 <code>myfirfilter</code>, <code>matlabtbeval</code> associates the HDL module instance with the MATLAB function <code>myfirfilter</code>. Alternatively, you can specify a different MATLAB function with the <code>-mfunc</code> property.</p> <p><code>-socket &lt;tcp_spec&gt;</code> Specifies TCP/IP socket communication for the link between the HDL simulator and MATLAB. For TCP/IP socket communication on a single computer, the <code>&lt;tcp_spec&gt;</code> 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” on page D-5 for some valid <code>tcp_spec</code> examples.</p> <p>For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports” on page D-2.</p> <p>If the HDL simulator and MATLAB are running 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 omit <code>-socket &lt;tcp_spec&gt;</code> from the command line.</p>

---

**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 “Communicating with MATLAB or Simulink and the HDL Simulator” on page 1-8. For more information on establishing the MATLAB end of the communication link, see “Starting the MATLAB Server” on page 2-49.

---

`-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. For example, if the HDL module instance is `myfirfilter`, `matlabtbeval` associates the HDL module instance with the MATLAB function `myfirfilter`. If you omit this argument and `matlabtbeval` does not find a MATLAB function with the same name, the command displays an error message.

## Description

The `matlabtbeval` command has the following characteristics:

- Starts the HDL simulator client component of the EDA Simulator Link™ MQ 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.

---

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

---

---

**Note** For the HDL simulator to establish a communication link with MATLAB, the MATLAB `hdldaemon` must be running with the same communication mode and, if appropriate, the same TCP/IP socket port as you specify with the `matlabtbeval` command.

---

## 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:
```

# 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.

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

```
> nomatlabtb
```

**See Also** `matlabtb`, `matlabcp`

<b>Purpose</b>	Load instantiated HDL module for verification with MATLAB
<b>Syntax</b>	<code>vsimmatlab &lt;instance&gt; [&lt;vsim_args&gt;]</code>
<b>Arguments</b>	<code>&lt;instance&gt;</code> Specifies the instance of an HDL module to load for verification. <code>&lt;vsim_args&gt;</code> Specifies one or more ModelSim vsim command arguments. For details, see the description of vsim in the ModelSim documentation.
<b>Description</b>	<p>The vsimmatlab 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 may be run from the HDL simulator prompt or from a Tcl script shell (tclsh).</p>
<b>Examples</b>	<p>The following command loads the HDL module instance parse from library work for verification and sets up ModelSim so it can establish a communication link with MATLAB:</p> <pre>ModelSim&gt; vsimmatlab work.parse</pre>

**Purpose** Load instantiated HDL module for cosimulation with Simulink

**Syntax** vsimulink <instance>  
[-socket <tcp\_spec>] [<vsim\_args>]

**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 Appendix D, “TCP/IP Socket Communication”.

If ModelSim and MATLAB are running 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 omit -socket <tcp-spec> from 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™ MQ blocks in your Simulink model. For more information on modes of communication, see “Communicating with MATLAB or Simulink and the HDL Simulator” on page 1-8. For more information on establishing the Simulink end of the communication link, see “Configuring the Communication Link in the HDL Cosimulation Block” on page 3-59.

---

<vsim\_args>

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

**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.

**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
```

# wrapverilog

---

**Purpose** Apply VHDL wrapper to Verilog module

**Syntax** wrapverilog [-nocompile] <verilog\_module>

**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*.

**Description** 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™ MQ.

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
```

---

**Note** EDA Simulator Link MQ now supports Verilog models directly, without requiring a VHDL wrapper. All EDA Simulator Link MQ MATLAB functions, and the HDL Cosimulation block, offer the same language-transparent feature set for both Verilog and VHDL models. The wrapverilog function is supported for backward compatibility, and is still in use by many EDA Simulator Link MQ demos.

---



## 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
```



# EDA Simulator Link™ MQ Simulink® Block Reference

---

# HDL Cosimulation

---

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

**Library** EDA Simulator Link MQ

**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. You can specify the period for each clock signal.
- Tcl commands to run before and after the simulation.

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 a *relative* timing relationship (Simulink seconds correspond to an HDL simulator-defined tick interval) or an *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. The **Pre-simulation commands** field on this pane is particularly useful for simulation initialization and startup operations, but it cannot be used to change simulation state.

---

**Note** You must make sure that signals being used in cosimulation have read/write access (this is done through the HDL simulator—see product documentation for details). This rule applies to all signals on the **Ports**, **Clocks**, and **Tcl** panes.

---

## Dialog Box

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

- “Ports Pane” on page 6-3
- “Connection Pane” on page 6-10
- “Timescales Pane” on page 6-13
- “Clocks Pane” on page 6-15
- “Tcl Pane” on page 6-18

### 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 Cosimulation

---

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 an input port is connected to a signal that has an explicit sample period, based on forward propagation, Simulink applies that rate to the port.
- If an input port is connected 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. An explicit sample time must be specified 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_E5`.

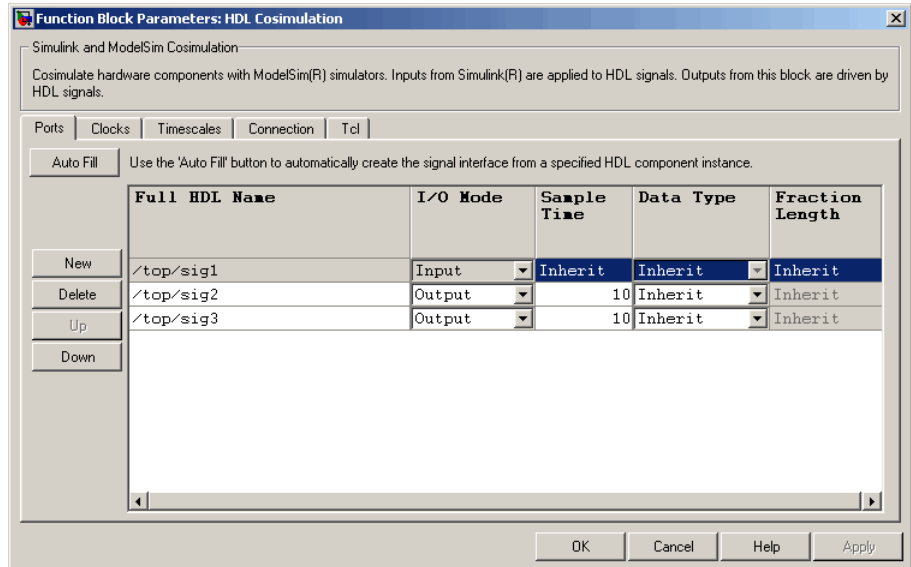
(Note, however, that can not force width; the width is always inherited from the HDL simulator. )

---

**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
-

Input/output ports can be set here as well; specify port as both input and output.



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” on page 3-47 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.

# HDL Cosimulation

---

- **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**.

---

**Note** When importing VHDL signals, signal names are returned in all capitals.

---

To edit the 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, 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” on page 3-41.

---

**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 (as long as you 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.

Input designates signals of your HDL module that are to be driven by Simulink. 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 are to be read by Simulink. For output signals, you must specify an explicit sample time. You can also specify a data type (except width), if desired (see Date Type and Fraction Length in a following section).

Since Simulink signals do not have the semantic of tri-states (there is no 'Z' value), it is not meaningful to connect to a bi-directional HDL signal directly. In order to interface with bi-directional signals, you can interface to the input of the output driver, the enable of the output driver, and the output of the input driver. This leaves the actual tri-state buffer in HDL where resolution functions can handle interfacing with other tri-state buffers.

## Sample Time

This property is enabled only for output signals. 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 “Timescales Pane” on page 6-13). See also “Representation of Simulation Time” on page 3-15.

## **Data Type** **Fraction Length**

These two related parameters apply only to output signals.

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** is enabled when the **Data Type** property is not set to Inherit.

Output port data types are determined by 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.

Inherit is the default setting. When Inherit is selected, the **Fraction Length** edit field is disabled.

Simulink always double checks that the word-length back propagated by Simulink matches the word length queried from the HDL simulator. If they don't match an error is generated. For example, if a Signal Specification block is connected 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 **Signed** is selected, the **Fraction Length** edit field is enabled. The port is assigned 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`.

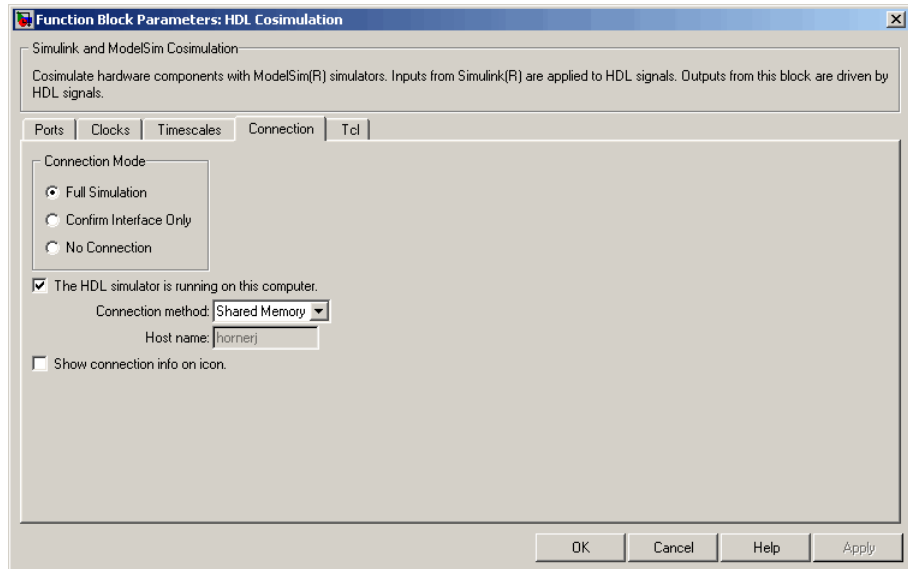
- Select **Unsigned** from the **Data Type** list if you want to explicitly assign an unsigned fixed point data type. When **Unsigned** is selected, the **Fraction Length** edit field is enabled. The port is assigned a fixed point type `ufixN_EnF`, where `N` is the signal width and `F` is the **Fraction 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`.

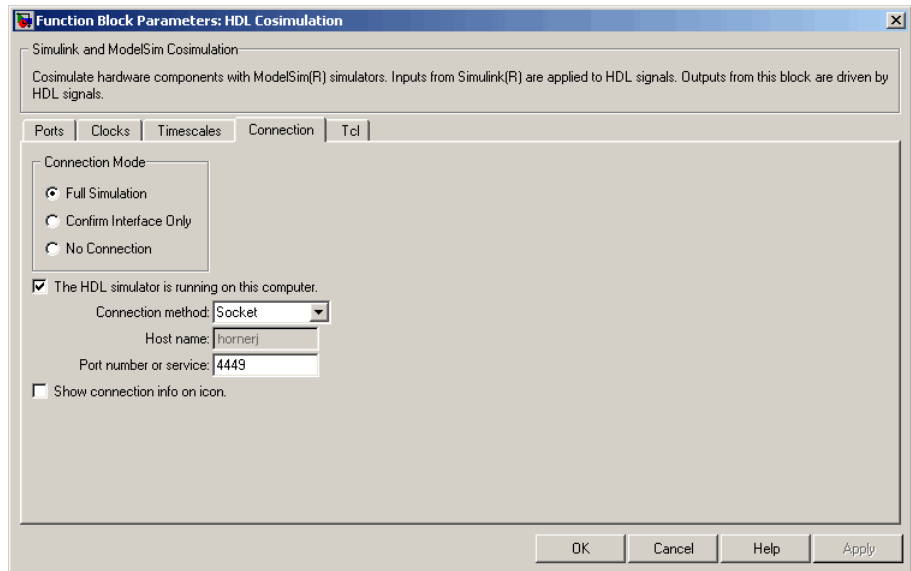
# HDL Cosimulation

## Connection Pane

This figure shows the default configuration of the **Connection** pane. By default, the block is configured for shared memory communication between Simulink and the HDL simulator, running 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 running a Simulink simulation, use these options to specify what type of simulation connection you want. Select one of the following:

- **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 2nd and 3rd options, the EDA Simulator Link™ MQ 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 this option is deselected, only TCP/IP socket mode is available, and the **Connection method** list is disabled.

## **Connection method**

This list is enabled when **The HDL Simulator is running on this computer** is selected. 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 “Communicating with MATLAB or Simulink and the HDL Simulator” on page 1-8.

## **Host name**

If Simulink and the HDL simulator are running on different computers, this text field is enabled. 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” on page D-2.

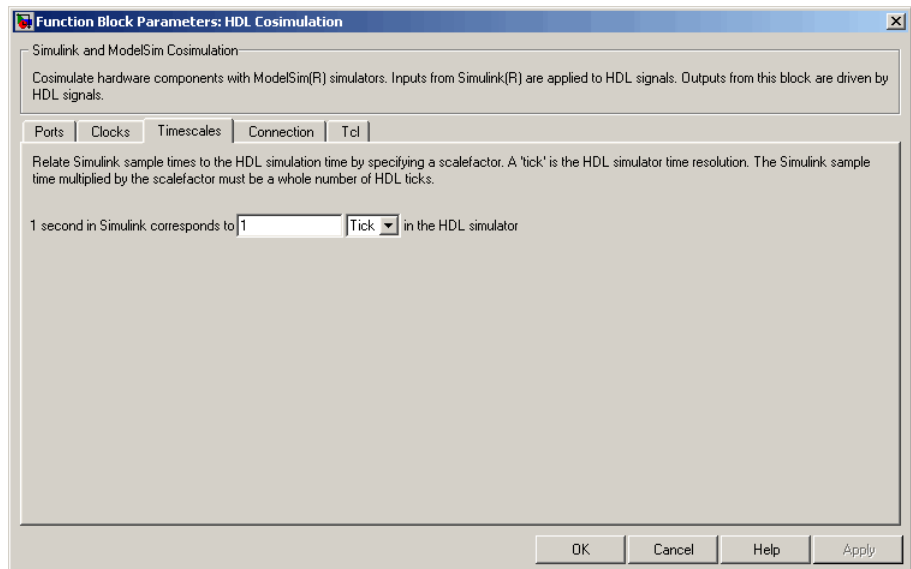
## **Show connection info on icon**

When this option is selected, Simulink indicates information about the selected communication method and (if applicable) communication options information on the HDL Cosimulation block icon. If shared memory is selected, the icon displays the string SharedMem. If TCP socket communication is selected, the icon 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 lets you choose a timing relationship between Simulink and the HDL simulator. The following figure shows the default settings of the **Timescales** pane.



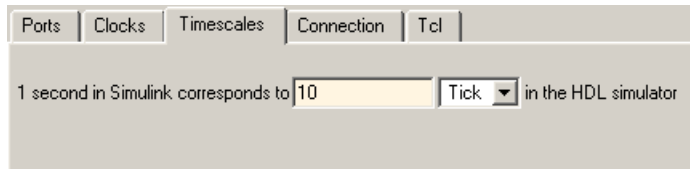
The **Timescales** pane specifies a correspondence between one second of Simulink time and some quantity of HDL simulator time. This quantity of HDL simulator time can be expressed in one of the following ways:

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

# HDL Cosimulation

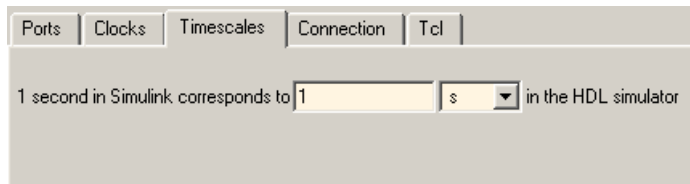
---

To use relative mode, select **Tick** from the list on the right, and enter the desired number of ticks in the edit box. For example, in the figure below the **Timescales** pane is configured for a relative timing correspondence of 10 HDL simulator ticks to 1 Simulink second.



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

To use absolute mode, select a unit of absolute time (available units are fs, ps, ns, us, ms, s ) from the list on the right. Then enter a scale factor in the left-side edit box. For example, in the figure below the **Timescales** pane is configured for an absolute timing correspondence of 1 HDL simulator second to 1 Simulink second.



For more information on calculating relative and absolute timing modes, see “Defining the Simulink and HDL Simulator Timing Relationship” on page 3-17.

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 “Representation of Simulation Time” on page 3-15.



## Clocks Pane

Create optional rising-edge and falling-edge clocks that apply stimuli to your cosimulation model using the Clocks pane of the HDL Cosimulation block. You can either specify an explicit period for each clock, or accept a default period of 2. Simulink attempts to create a clock that has a 50% duty cycle and a predefined phase that is inverted for the falling edge case.

Whether you have configured the **Timescales** pane for relative timing mode or absolute timing mode, the following restrictions apply to clock periods:

- 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, and therefore the EDA Simulator Link MQ software creates the falling edge at

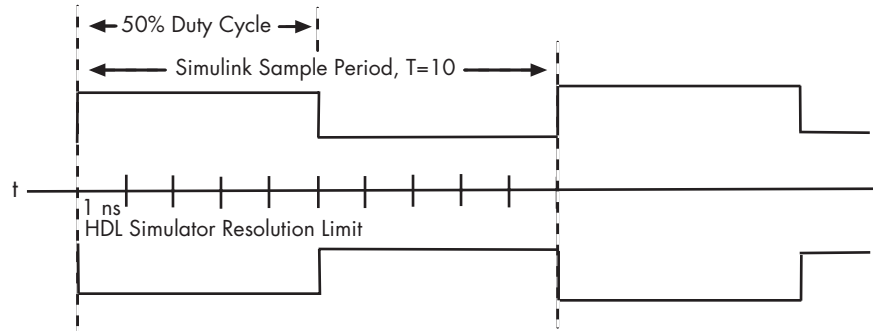
$$\text{clockperiod} / 2$$

(rounded down to the nearest integer).

The following figure shows a timing diagram that includes rising-edge and falling-edge clocks with a Simulink sample period of  $T=10$  and an HDL simulator resolution limit of 1 ns. The figure also shows that given those timing parameters, the clock duty cycle is 50%.

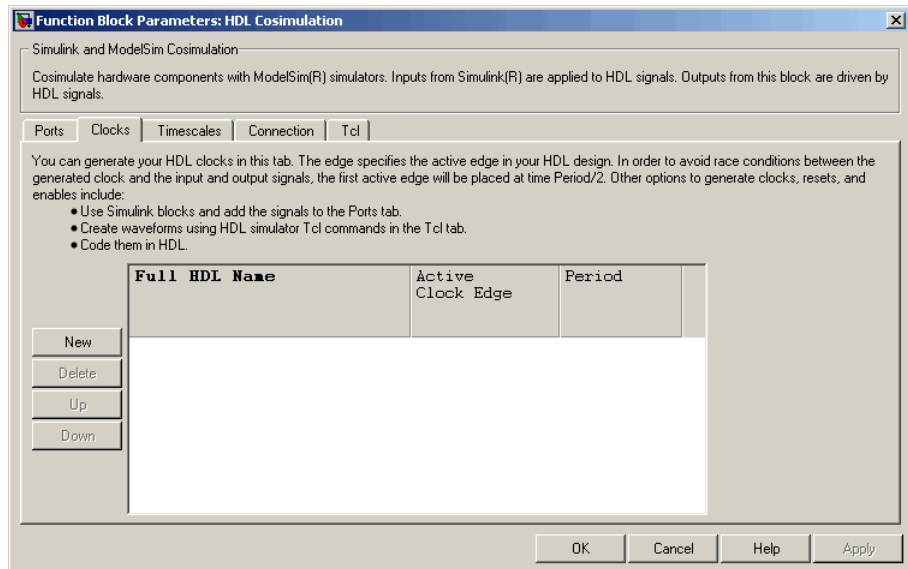
# HDL Cosimulation

Rising Edge Clock



Falling Edge Clock

For more information on calculating relative and absolute timing modes, see “Defining the Simulink and HDL Simulator Timing Relationship” on page 3-17 .



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**.

To edit the name of a clock signal, double-click it and enter the correct name. To edit the properties of a clock signal, select the appropriate property in that signal row. The properties of a clock signal are

### **Full HDL Name**

Specify each clock as a signal path name, using the HDL simulator path name syntax. A sample path name for a clock might be `/manchester/clk`.

For information about and requirements for path specifications in Simulink, see “Specifying HDL Signal/Port and Module Paths for Cosimulation” on page 3-41.

---

**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 (as long as you 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.

---

# HDL Cosimulation

---

## 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, and therefore the EDA Simulator Link MQ software creates the falling edge at

$$\text{clockperiod} / 2$$

(rounded down to the nearest integer).

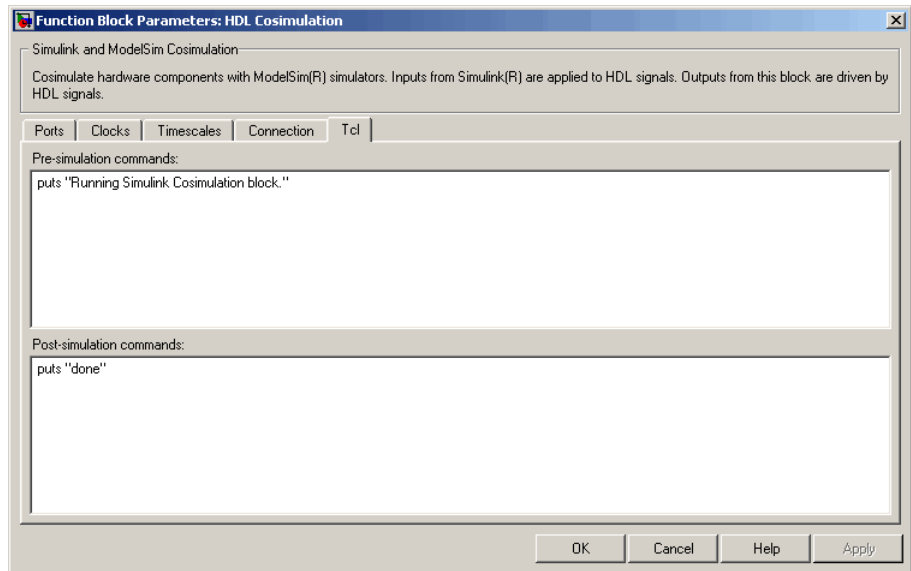
---

**Note** Vectored signals in the **Clocks** pane are not supported. Signals must be logic types with '1' and '0' values.

---

## Tcl Pane

Specify tools command language (Tcl) commands to be executed before and after the HDL simulator simulates the HDL component of your Simulink model



## 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.

Alternatively, 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
```

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.

---

**Note** The command string or DO file that you specify for this parameter cannot include commands that load a ModelSim project or modify simulator state. For example, they cannot include commands such as start, stop, or restart.

---

## 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.

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

```
do mycosimcleanup.do
```

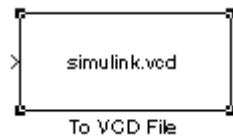
---

## Notes

- 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 have an opportunity to 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.
-

**Purpose** Generate value change dump (VCD) file

**Library** EDA Simulator Link MQ



## Description

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. VCD files can be useful during design verification. Some examples of how you might apply VCD files include the following cases:

- 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

In addition, VCD files include data that can be graphically displayed or analyzed 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.

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

- 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

## To VCD File

---

VCD files can grow very large for larger designs or smaller designs with longer simulation runs. However, the size of a VCD file generated by the To VCD File block is limited only by 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 6-25.

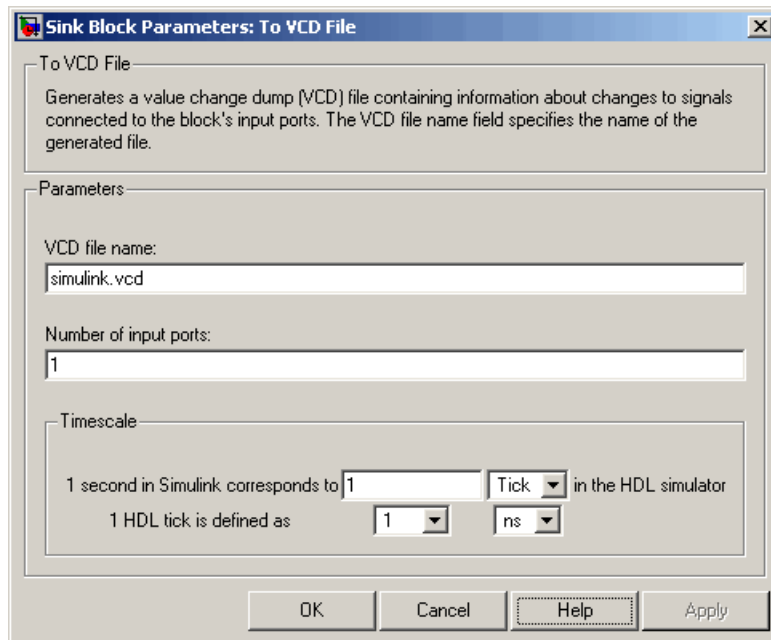
---

**Note** The toVCD block is integrated into the Simulink Signal & Scope Manager. See the Simulink User’s Guide for more information on using the Signal & Scope Manager.

---



## Dialog Box



### 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 directory. 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 occurs when the input port receives a multi-dimensional signal.

Because multi-dimensional signals are not part of the VCD specification, they are flattened 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. This quantity of HDL simulator time can be expressed in one of the following ways:

- In *relative* terms (i.e., as some number of HDL simulator ticks). In this case, the cosimulation is said to operate in *relative timing mode*. Relative timing mode is the 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 is said to operate 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 MQ version 1.0 \$ end</pre>	Version of the VCD block that generated the file.
<pre>\$timescale 1 ns \$ end</pre>	The time scale that was used during the simulation.
<pre>\$scope module manchestermodel \$end</pre>	The scope of the module being dumped.

## Generated VCD File Format (Continued)

File Content	Description
<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>	Variable definitions. Each definition associates a signal with character identification code (symbol). The symbols are derived from printable characters in the ASCII character set from ! to ~. Variable definitions also include the variable type (wire) and size in bits.
<pre>\$upscope \$end</pre>	Marks a change to the next higher level in the HDL design hierarchy.
<pre>\$enddefinitions \$end</pre>	Marks the end of the header and definitions section.
<pre>#0</pre>	Simulation start time.
<pre>\$dumpvars 0! 0" 0# 0\$ \$end</pre>	Lists the values of all defined variables at time equals 0.

## Generated VCD File Format (Continued)

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

## To VCD File

---

# Examples

---

Use this list to find examples in the documentation.

## **Invoking ModelSim Using the MATLAB Function vsim**

“vsim Examples” on page 1-23

## **MATLAB and ModelSim Random Number Generator Tutorial**

“Setting Up Tutorial Files” on page 2-56

“Starting the MATLAB Server” on page 2-57

“Setting Up ModelSim” on page 2-58

“Developing the VHDL Code” on page 2-60

“Compiling the VHDL File” on page 2-63

“Loading the Simulation” on page 2-63

“Developing the MATLAB Function” on page 2-66

“Running the Simulation” on page 2-68

“Shutting Down the Simulation” on page 2-72

## **Frame-Based Processing**

“Frame-Based Cosimulation Example” on page 3-26

## **Simulink and ModelSim Inverter Tutorial**

“Developing the VHDL Code” on page 3-72

“Compiling the VHDL File” on page 3-73

“Creating the Simulink Model” on page 3-75

“Setting Up ModelSim for Use with Simulink” on page 3-84

“Loading Instances of the VHDL Entity for Cosimulation with Simulink”  
on page 3-85

“Running the Simulation” on page 3-86

“Shutting Down the Simulation” on page 3-89



## **Generating a VCD File**

“toVCD Block Tutorial” on page 3-90



# ADMS Support

---

Adding Libraries for ADMS Support  
(p. B-2)

Contains instructions for  
adding system libraries to  
LD\_LIBRARY\_PATH

Linking MATLAB® or Simulink®  
Software to ModelSim in ADMS  
(p. B-3)

Contains instructions and  
constraints for using EDA Simulator  
Link™ MQ with ADMS

## Adding Libraries for ADMS Support

No special library installation is needed for ADMS support.

If you need to add system libraries to the `LD_LIBRARY_PATH` you can add them in a `.vams_setup` file. Doing it this way (rather than specifying the path before calling `vasim`) will prevent `vasim` from overwriting the path addition each time it starts.

This example appends the system shared libraries to `LD_LIBRARY_PATH`:

```
proc fixldpath {args} {
  set pvpair [split [join $args]]
  set pval   [lindex $pvpair 1]
  append newpval /directory/of/system/dlls ":" $pval
  append setcmd { array set env [list LD_LIBRARY_PATH ] " " $newpval " " }
  uplevel 1 $setcmd
}

fixldpath [array get env LD_LIBRARY_PATH]
```

## Linking MATLAB® or Simulink® Software to ModelSim in ADMS

### In this section...

“Starting ADMS for Use with EDA Simulator Link™ MQ” on page B-3

“Using Tcl Test Bench Commands with ADMS” on page B-4

“Constraints” on page B-4

### Starting ADMS for Use with EDA Simulator Link™ MQ

You must call `vasim` with all parameters manually; the configuration script available for ModelSim® SE/PE is not available for ADMS.

When you call `vasim`, you must provide the `-ms` and `-foreign` parameters. For example,

```
vasim -lib ADC12_ELDO_MS -cmd
      /devel/user/work/ams/adc12test.cmd TEST -ms -foreign matlabclient path/matlablibrary
```

where:

`-lib ADC12_ELDO_MS` is the model library

`/devel/user/work/ams/adc12test.cmd` is the command file

`TEST` is the design

`path/matlablibrary` is the path to and the name of the MATLAB shared library (see “Using the EDA Simulator Link™ MQ Libraries” on page 1-18)

A similar example for the Simulink link looks like this:

```
vasim -lib ADC12_ELDO_MS -cmd
```

```
/devel/user/work/ams/adc12test.cmd TEST -ms -foreign simlinkserver path/simlinklibrary
```

All line arguments after 'ms' are sent to the ModelSim process.

See your ModelSim documentation for more about the -foreign option.

## Using Tcl Test Bench Commands with ADMS

When you use any of the EDA Simulator Link™ MQ ModelSim link commands (e.g., matlabcp, matlabtb, matlabtbeval, nomatlabtb), you must precede each command with `ms` in the ADMS Tcl interpreter. For example:

```
ms matlabtb myfirfilter 5 ns -repeat 10 ns -socket 4449
```

All line arguments after 'ms' are sent to the ModelSim process.

## Constraints

### Setting Simulation Running Time

When running cosimulation sessions in Simulink, make sure that the runtime of the ADMS simulation is greater than or equal to the Simulink runtime.

### Solaris Support

- ADMS is not supported on Solaris 64-bit, but is supported on Solaris 8, 9, 10 32-bits
- glibc errors on glnxa64 when running a cosimulation session with ADMS

If you get the following:

```
*** glibc detected *** free(): invalid pointer: 0x000000001eeb490 ***
```

The you must do this:

```
setenv MALLOC_CHECK_ 0
```

Which says, effectively, "Do not generate an error message, and do not kill the program".

# EDA Simulator Link™ MQ Machine Configuration Requirements

---

Valid Configurations For Using the  
EDA Simulator Link™ MQ Software  
with MATLAB® Applications (p. C-2)

Describes how you choose the  
number of clients and servers  
and how they communicate when  
using the EDA Simulator Link™  
MQ cosimulation interface with  
MATLAB® software

Valid Configurations For Using the  
EDA Simulator Link™ MQ Software  
with Simulink® Software (p. C-4)

Describes how you choose the  
number of clients and servers  
and how they communicate when  
using the EDA Simulator Link  
MQ cosimulation interface with  
Simulink® software

## Valid Configurations For Using the EDA Simulator Link™ MQ Software with MATLAB® Applications

The following list provides samples of valid configurations for using the ModelSim® HDL simulator and the EDA Simulator Link™ MQ software with MATLAB® software. The scenarios apply whether the HDL simulator is running on the same or different computing system as the MATLAB software. In a network configuration, you use an Internet address in addition to a TCP/IP socket port to identify the servers in an application environment.

- An HDL simulator session linked to a MATLAB function foo through a single instance of the MATLAB server
- An HDL simulator session linked to multiple MATLAB functions (for example, foo and bar) through a single instance of the MATLAB server
- An HDL simulator session linked to a MATLAB function foo through multiple instances of the MATLAB server (each running within the scope of a unique MATLAB session)
- Multiple HDL simulator sessions each linked to a MATLAB function foo through multiple instances of the MATLAB server (each running within the scope of a unique MATLAB session)
- Multiple HDL simulator sessions each linked to a different MATLAB function (for example, foo and bar) through the same instance of the MATLAB server
- Multiple HDL simulator sessions each linked to MATLAB function foo through a single instance of the MATLAB server

Although multiple HDL simulator sessions can link to the same MATLAB function in the same instance of the MATLAB server, as this configuration scenario suggests, such links are not recommended. If the MATLAB function maintains state (for example, maintains global or persistent variables), you may experience unexpected results because the MATLAB function does not distinguish between callers when handling input and output data. If you must apply this configuration scenario, consider deriving unique instances of the MATLAB function to handle requests for each HDL entity.



---

## **Notes**

- Shared memory communication is an option for configurations that require only one communication link on a single computing system.
  - TCP/IP socket communication is required for configurations that use multiple communication links on one or more computing systems. Unique TCP/IP socket ports distinguish the communication links.
  - In any configuration, an instance of MATLAB can run only one instance of the EDA Simulator Link MQ MATLAB server (hdldaemon) at a time.
  - In a TCP/IP configuration, the MATLAB server can handle multiple client connections to one or more HDL simulator sessions.
-

## **Valid Configurations For Using the EDA Simulator Link™ MQ Software with Simulink® Software**

The following list provides samples of valid configurations for using the ModelSim® HDL simulator and the EDA Simulator Link™ MQ software with Simulink® software. The scenarios apply whether the HDL simulator is running on the same or different computing system as the MATLAB or Simulink products. In a network configuration, you use an Internet address in addition to a TCP/IP socket port to identify the servers in an application environment.

- An HDL Cosimulation block in a Simulink model linked to a single HDL simulator session
- Multiple HDL Cosimulation blocks in a Simulink model linked to the same HDL simulator session
- An HDL Cosimulation block in a Simulink model linked to multiple HDL simulator sessions
- Multiple HDL Cosimulation blocks in a Simulink model linked to different HDL simulator sessions

---

### **Notes**

- HDL Cosimulation blocks in a Simulink model can connect to the same or different HDL simulator sessions.
  - TCP/IP socket communication is required for configurations that use multiple communication links on one or more computing systems. Unique TCP/IP socket ports distinguish the communication links.
  - Shared memory communication is an option for configurations that require only one communication link on a single computing system.
-

# TCP/IP Socket Communication

---

Choosing TCP/IP Socket Ports  
(p. D-2)

Contains instructions for selecting  
TCP/IP socket ports

Specifying TCP/IP Values (p. D-5)

Provides some examples of valid  
TCP/IP values that can be used for  
TCP/IP socket communication.

TCP/IP Services (p. D-6)

Explains how using TCP/IP services  
may help optimize your application

## Choosing TCP/IP Socket Ports

Depending on your particular configuration (for example, when the MATLAB® software and the HDL simulator reside on separate machines), when creating an EDA Simulator Link™ MQ MATLAB application or defining the block parameters of an HDL Cosimulation block, you may need to identify the TCP/IP socket port number or service name (alias) to be used for EDA Simulator Link MQ connections.

To use the TCP/IP socket communication, you must choose a TCP/IP socket port number for the server component to listen on that is available in your computing environment. Client components can connect to a specific server by specifying the port number on which the server is listening. For remote network configurations, the Internet address helps distinguish multiple connections.

The socket port resource is associated with the server component of an EDA Simulator Link MQ configuration. That is, if you use MATLAB in a test bench configuration, the socket port is a resource of the system running MATLAB. If you use a Simulink® design in a cosimulation configuration, the socket port is a resource of the system running the HDL simulator.

A TCP/IP socket port number (or alias) is a shared resource. To avoid potential collisions, particularly on servers, you should use caution when choosing a port number for your application. Consider the following guidelines:

- If you are setting up a link for MATLAB, consider the EDA Simulator Link MQ option that directs the operating system to choose an available port number for you. To use this option, specify 0 for the socket port number.
- Choose a port number that is registered for general use. Registered ports range from 1024 to 49151.
- If you do not have a registered port to use, review the list of assigned registered ports and choose a port in the range 5001 to 49151 that is not in use. Ports 1024 to 5000 are also registered, however operating systems use ports in this range for client programs.

Consider registering a port you choose to use.

- Choose a port number that does not contain patterns or have a known meaning. That is, avoid port numbers that more likely to be used by others because they are easier to remember.
- Do not use ports 1 to 1023. These ports are reserved for use by the Internet Assigned Numbers Authority (IANA).
- Avoid using ports 49152 through 65535. These are dynamic ports that operating systems use randomly. If you choose one of these ports, you risk a potential port conflict.
- TCP/IP port filtering on either the client or server side can cause the EDA Simulator Link MQ interface to fail to make a connection.

In such cases the error messages displayed by the EDA Simulator Link MQ interface indicate the lack of a connection, but do not explicitly indicate the cause. A typical scenario caused by port filtering would be a failure to start a simulation in the HDL simulator, with the following warning displayed in the HDL simulator if the simulation is restarted:

```
#MLWarn - MATLAB server not available (yet),  
The entity 'entityname' will not be active
```

In MATLAB, checking the server status at this point indicates that the server is running with no connections:

```
x=hlddaemon('status')  
HLDdaemon server is running with 0 connections  
x=  
4449
```

---

**Windows Users** If you suspect that your chosen socket port is filtered, you can check it as follows:

- 1** From the Windows **Start** menu, select **Settings > Network Connections**.
  - 2** Select **Local Area Connection** from the **Network and Dialup Connections** window.
  - 3** From the **Local Area Connection** dialog, select **Properties > Internet Protocol (TCP/IP > Properties > Advanced > Options > TCP/IP filtering > Properties**.
  - 4** If your port is listed in the **TCP/IP filtering Properties** dialog, you should select an unfiltered port. The easiest way to do this is to specify 0 for the socket port number to let the EDA Simulator Link MQ software choose an available port number for you.
-

## Specifying TCP/IP Values

Specifies TCP/IP socket communication for links between the HDL simulator and Simulink® software. 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. The following table lists different ways of specifying `tcp_spec`.

<b>Format</b>	<b>Example</b>
<code>&lt;port-num&gt;</code>	4449
<code>&lt;port-alias&gt;</code>	matlabservice
<code>&lt;port-num&gt;@&lt;host&gt;</code>	4449@compa
<code>&lt;host&gt;:&lt;port-num&gt;</code>	compa:4449
<code>&lt;port-alias&gt;@&lt;host-ia&gt;</code>	matlabservice@123.34.55.23

## TCP/IP Services

By setting up the MATLAB® server as a service, you can run the service in the background, allowing it to handle different HDL simulator client requests over time without you having to start and stop the service manually each time. Although it makes less sense to set up a service for the Simulink® software as you cannot really automate the starting of an HDL simulator service, you might want to use a service with Simulink to reserve a TCP/IP socket port.

Services are defined in the `etc/services` file located on each computer; consult the User's Guide for your particular operating system for instructions and more information on setting up TCP/IP services.

For remote connections, the service name must be set up on both the client and server side. For example, if the service name is “matlab-service” and you are performing a Windows-Linux cross-platform simulation, the service name must appear in the service file on both the Windows machine and the Linux machine.



# Race Conditions in HDL Simulators

---

Overview (p. E-2)

Describes the problem of race conditions in hardware simulation

Potential Race Conditions in Simulink® Link Sessions (p. E-3)

Describes race conditions when cosimulating with Simulink® software and how to work around them

Potential Race Conditions in MATLAB® Link Sessions (p. E-5)

Describes race conditions when cosimulating with MATLAB® software and how to work around them

Further Reading (p. E-6)

Provides suggestions for further study about race conditions in hardware simulation

## Overview

A well-known issue in hardware simulation is the potential for nondeterministic results when race conditions are present. Because the HDL simulator is a highly parallel execution environment, you must write the HDL such that the results do not depend on the ordering of process execution.

Although there are well-known coding idioms for ensuring successful simulation of a design under test, you must always take special care at the testbench/DUT interfaces for applying stimulus and reading results, even in pure HDL environments. For an HDL/foreign language interface, such as with a Simulink® or MATLAB® link session, the problem is compounded if there is no common synchronization signal, such as a clock coordinating the flow of data.

## Potential Race Conditions in Simulink® Link Sessions

All the signals on the interface of an HDL Cosimulation block in the Simulink® library have an intrinsic sample rate associated with them. This sample rate can be thought of as an implicit clock that controls the simulation time at which a value change can occur. Because this implicit clock is completely unknown to the HDL engine (that is, it is not an HDL signal), the times at which input values are driven into the HDL or output values are sampled from the HDL are asynchronous to any clocks coded in HDL directly, even if they are nominally at the same frequency.

For Simulink value changes scheduled to occur at a specific simulation time, the HDL simulator does not make any guarantees as to the order that value change occurs versus some other blocking signal assignment. Thus, if the Simulink values are driven/sampled at the same time as an active clock edge in the HDL, there is a race condition.

For cases where your active HDL clock edge and your intrinsic Simulink active clock edges are at the same frequency, you can ensure proper data propagation by offsetting one of those edges. Because the Simulink sample rates are always aligned with time 0, you can accomplish this offset by shifting the active clock edge in the HDL off of time 0. If you are coding the clock stimulus in HDL, use a delay operator ("after" or "#") to accomplish this offset.

When using a Tcl "force" command to describe the clock waveform, you can simply put the first active edge at some nonzero time. Using a nonzero value allows a Simulink sample rate that is the same as the fundamental clock rate in your HDL. This example shows a 20 ns clock (so the Simulink sample rates will also be every 20 ns) with an active positive edge that is offset from time 0 by 2 ns:

```
Mentor> force top.clk 1 {2 ns}, 0 { 12 ns} repeat 20 ns
```

For HDL Cosimulation Blocks with Clock panes, you can define the clock period and active edge in that pane. The waveform definition places the **non-active** edge at time 0 and the **active** edge at time T/2. This placement ensures the maximum setup and hold times for a clock with a 50% duty cycle.

If the Simulink sample rates are at a different frequency than the HDL clocks, then you must synchronize the signals between the HDL and Simulink as you

would do with any multiple time-domain design, even one in pure HDL. For example, you can place two synchronizing flip-flops at the interface.

If your cosimulation does not include clocks, then you must also treat the interfacing of Simulink and the HDL code as being between asynchronous time domains. You may need to over-sample outputs to ensure that all data transitions are captured.

## Potential Race Conditions in MATLAB® Link Sessions

When you use the `-sensitivity`, `-rising_edge`, or `-falling_edge` scheduling options to `matlabtb` or `matlabcp` to trigger MATLAB® function calls, the propagation of values follow the same semantics as a pure HDL design; you are guaranteed that the triggers must occur before the results can be calculated. You still can have race conditions, but they can be analyzed within the HDL alone.

However, when you use the `-time` scheduling option to `matlabtb` or `matlabcp`, or use `"tnext"` within the MATLAB function itself, the driving of signal values or sampling of signal values cannot be guaranteed in relation to any HDL signal changes. It is as if the potential Simulink race conditions in that time-based scheduling are like an implicit clock that is unknown to the HDL engine and not visible by just looking at the HDL code.

The remedies are the same as for the Simulink signal interfacing: ensure the sampling and driving of signals does not occur at the same simulation times as the MATLAB function calls.

## Further Reading

Problems interfacing designs from testbenches and foreign languages, including race conditions in pure HDL environments, are well-known and extensively documented. Some texts that describe these issues include:

- The documentation for each vendor's HDL simulator product
- The HDL standards specifications
- Writing Testbenches: Functional Verification of HDL Models, Janick Bergeron, 2nd edition, © 2003
- Verilog and SystemVerilog Gotchas, Stuart Sutherland and Don Mills, © 2007
- SystemVerilog for Verification: A Guide to Learning the Testbench Language Features, Chris Spear, © 2007
- Principles of Verifiable RTL Design, Lionel Bening and Harry D. Foster, © 2001

## A

- Absolute timing mode 3-21
- action property
  - description of 4-2
- addresses, Internet D-2
- application software 1-9
- application specific integrated circuits (ASICs) 1-3
- applications 1-3
  - coding for EDA Simulator Link™ MQ software 2-4
  - overview of 2-4
  - programming with EDA Simulator Link™ MQ software
    - overview of 2-4
- arguments
  - for matlabcp command 5-2
  - for matlabtb command 5-8
  - for matlabtbeval command 5-13
  - for pingHdlSim function 4-18
  - for tclHdlSim function 4-20
  - for vsimmatlab command 5-17
  - for vsimulink command 5-18
  - for wrapverilog command 5-20
- array data types
  - conversions of 2-20
  - VHDL 2-9
- array indexing
  - differences between MATLAB and VHDL 2-20
- arrays
  - converting to 2-26
  - indexing elements of 2-20
  - of VHDL data types 2-9
- ASICs (application specific integrated circuits) 1-3
- Auto fill
  - in Ports pane of HDL Cosimulation block 6-2
  - using in Ports pane 3-46

## B

- behavioral model 1-3
- BIT data type 2-9
  - conversion of 2-20
  - converting to 2-26
- bit vectors
  - converting for MATLAB 2-24
  - converting to 2-26
- BIT\_VECTOR data type 2-9
  - conversion of 2-20
  - converting for MATLAB 2-24
  - converting to 2-26
- block input ports parameter
  - description of 6-2
- Block input ports parameter
  - mapping signals with 3-46
- block latency 3-32
- block library
  - description of 3-8
- block output ports parameter
  - description of 6-2
- Block output ports parameter
  - mapping signals with 3-46
- block parameters
  - setting programmatically 3-64
- Block Parameters dialog
  - for HDL Cosimulation block 3-45
- block ports
  - mapping signals to 3-46
- blocks
  - HDL Cosimulation
    - description of 6-2
  - To VCD File
    - description of 6-21
- blocksets
  - for creating hardware models 3-5
  - for EDA applications 3-5
  - installing 1-12
- breakpoints
  - setting in MATLAB 2-52

- bypass
  - HDL Cosimulation block 3-59
- C**
- callback specification 2-14
- callback timing 2-40
  - cancel option 5-8
- CHARACTER data type 2-9
  - conversion of 2-20
- client
  - for MATLAB and HDL simulator links 1-5
  - for Simulink and HDL simulator links 1-7
- client/server environment
  - MATLAB and HDL simulator 1-5
  - Simulink and HDL simulator 1-7
- clocks
  - specifying for HDL Cosimulation blocks 3-43
- Clocks pane
  - configuring block clocks with 3-43
  - description of 6-2
- column-major numbering 2-20
- comm status field
  - checking with `hdldaemon` function 2-51
  - description of 4-10
- commands, HDL simulator 5-1
  - See also* HDL simulator commands
- communication
  - configuring for blocks 3-59
  - initializing for HDL simulator and MATLAB session 2-40
  - modes of 1-8
  - socket ports for D-2
- communication channel
  - checking identifier for 2-51
- communication modes
  - checking 2-51
  - specifying for Simulink links 3-70
  - specifying with `hdldaemon` function 2-49
- Communications Blockset
  - as optional software 1-9
  - using for EDA applications 3-5
- compilation, VHDL code 2-11
- compiler, VHDL 2-11
- component function
  - programming for HDL verification 2-12
- configuration file
  - using with ModelSim vsim 1-25
- configurations
  - deciding on for MATLAB C-2
  - deciding on for Simulink C-4
  - MATLAB
    - multiple-link C-2
  - Simulink
    - multiple-link C-4
  - single-system for MATLAB C-2
  - single-system for Simulink C-4
  - valid for MATLAB and HDL simulator C-2
  - valid for Simulink and HDL simulator C-4
- `figuremodelsim` function
  - description of 4-2
- Connection pane
  - configuring block communication with 3-59
  - description of 6-2
- connections status field
  - checking with `hdldaemon` function 2-51
  - description of 4-10
- connections, link
  - checking number of 2-51
  - TCP/IP socket D-2
- Continue button, MATLAB 2-52
- Continue option, MATLAB 2-52
- continuous signals 3-15
- cosimulation
  - bypassing 3-59
  - controlling MATLAB
    - overview of 2-48
  - loading HDL modules for 3-70
  - logging changes to signal values during 3-66



- running Simulink and ModelSim
  - tutorial 3-86
- shutting down Simulink and ModelSim
  - tutorial 3-89
- starting MATLAB
  - overview of 2-48
- starting with Simulink 3-71
- cosimulation environment
  - MATLAB and HDL simulator 1-5
  - Simulink and HDL simulator 1-7
- cosimulation output ports
  - specifying 3-56
- Cosimulation timing
  - absolute mode 6-2
  - relative mode 6-2

## D

- data types
  - conversions of 2-20
  - converting for MATLAB 2-24
  - converting for the HDL simulator 2-26
  - HDL port
    - verifying 2-18
  - unsupported VHDL 2-9
  - VHDL port 2-9
- dbstop function 2-52
- dec2mv1 function
  - description of 4-9
- delta time 3-32
- demos 1-30
  - for EDA Simulator Link™ MQ 1-28
- deposit
  - changing signals with 3-15
  - for iport parameter 2-14
  - with force commands 2-51
- design process, hardware 1-3
- dialogs
  - for HDL Cosimulation block 6-2
  - for To VCD File block 6-21

- discrete blocks 3-15
- do command 3-62
- DO files
  - specifying for HDL Cosimulation blocks 3-62
- documentation
  - overview 1-28
- double values
  - as representation of time 2-40
  - converting for MATLAB 2-24
  - converting for the HDL simulator 2-26
- dspstartup M-file 3-38
- duty cycle 3-43

## E

- EDA (Electronic Design Automation) 1-3
- EDA Simulator Link™ MQ
  - default libraries 1-18
- EDA Simulator Link™ MQ libraries
  - using 1-18
- EDA Simulator Link™ MQ software
  - block library
    - using to add HDL to Simulink with 3-8
  - definition of 1-3
  - installing 1-12
  - setting up the HDL simulator for 1-12
  - workflow for using with MATLAB 1-26
  - workflow for using with Simulink 1-27
- Electronic Design Automation (EDA) 1-3
- entities
  - coding for MATLAB verification 2-7
  - loading for cosimulation 3-85
  - loading for cosimulation with Simulink 3-70
  - sample definition of 2-10
- entities or modules
  - getting port information of 2-14
- enumerated data types 2-9
  - conversion of 2-20
  - converting to 2-26
- environment

- cosimulation with MATLAB and HDL simulator 1-5
  - cosimulation with Simulink and HDL simulator 1-7
  - examples 3-5
    - configuremodelsim function 4-2
    - dec2mvl function 4-9
    - hdldaemon function 4-10
    - matlabcp command 5-2
    - matlabtb command 5-8
    - matlabtbeval command 5-13
    - mv12dec function 4-17
    - nomatlabtb command 5-16
    - pingHdlSim function 4-18
    - Simulink and ModelSim 3-72
    - tclHdlSim function 4-20
    - test bench function 2-29
    - VCD file generation 3-90
    - vsim function 4-21
    - vsimmatlab command 5-17
    - vsimulink command 5-18
    - wrapverilog command 5-20
    - See also* Manchester receiver Simulink model
- F**
- falling option 5-8
    - specifying scheduling options with 2-40
  - falling-edge clocks
    - creating for HDL Cosimulation blocks 3-43
    - description of 6-2
    - specifying as scheduling options 2-40
  - Falling-edge clocks parameter
    - specifying block clocks with 3-43
  - field programmable gate arrays (FPGAs) 1-3
  - files
    - VCD 6-25
  - force command
    - applying simulation stimuli with 2-51
    - resetting clocks during cosimulation with 3-71
  - foreign option
    - with ModelSim vsim 1-25
  - FPGAs (field programmable gate arrays) 1-3
  - Frame-based processing 3-24
    - example of 3-26
    - in cosimulation 3-24
    - performance improvements gained from 3-24
    - requirements for use of 3-24
    - restrictions on use of 3-24
  - functions 4-1
    - resolution 3-15
    - See also* MATLAB functions
- G**
- Go Until Cursor option, MATLAB 2-52
- H**
- hardware description language (HDL). *See* HDL
  - hardware design process 1-3
  - hardware model design
    - creating in Simulink 3-5
    - running and testing in Simulink 3-39
  - HDL (hardware description language) 1-3
  - HDL cosimulation block
    - configuring ports for 3-46
    - opening Block Parameters dialog for 3-45
  - HDL Cosimulation block
    - adding to a Simulink model 3-8
    - black boxes representing 3-5
    - bypassing 3-59
    - configuration requirements for C-4
    - configuring clocks for 3-43
    - configuring communication for 3-59
    - configuring Tcl commands for 3-62
    - description of 6-2
    - design decisions for 3-5

- handling of signal values for 3-14
    - in EDA Simulator Link™ MQ
      - environment 1-7
    - scaling simulation time for 3-15
    - valid configurations for C-4
  - HDL entities
    - loading for cosimulation with Simulink 3-70
  - HDL models 1-3
    - adding to Simulink models 3-8
    - compiling 2-11
    - configuring Simulink for 3-38
    - cosimulation 1-3
    - debugging 2-11
    - porting 3-66
    - running in Simulink 3-71
    - testing in Simulink 3-71
    - verifying 1-3
    - See also* VHDL models
  - HDL module
    - associating with link function 2-37
  - HDL modules
    - coding for MATLAB verification 2-7
    - getting port information of 2-14
    - loading for verification 2-12
    - naming 2-8
    - using port information for 2-18
    - validating 2-18
    - verifying port direction modes for 2-18
  - HDL simulator
    - handling of signal values for 3-14
    - initializing for MATLAB session 2-40
    - simulation time for 3-15
    - starting 1-23
    - starting for use with Simulink 3-70
  - HDL simulator commands 5-2
    - force
      - applying simulation stimuli with 2-51
      - resetting clocks during cosimulation with 3-71
  - matlabcp
    - description of 5-2
  - matlabtb
    - description of 5-8
    - initializing HDL simulator with 2-40
  - matlabtbeval
    - description of 5-13
    - initializing HDL simulator with 2-40
  - nomatlabtb 5-16
  - run 2-52
  - specifying scheduling options with 2-40
  - vsimmatlab
    - loading HDL modules for verification with 2-12
  - vsimulink
    - loading HDL modules for cosimulation with 3-70
  - HDL simulator running on this computer
    - parameter
      - description of 6-2
  - hdldaemon function
    - checking link status of 2-51
    - configuration restrictions for C-2
    - description of 4-10
    - starting 2-49
  - help
    - for EDA Simulator Link™ MQ software 1-28
  - Host name parameter
    - description of 6-2
    - specifying block communication with 3-59
  - hostnames
    - identifying HDL simulator server 3-59
    - identifying MATLAB server 2-40
    - identifying server with D-2
- I**
- IN direction mode 2-8
    - verifying 2-18
  - INOUT direction mode 2-8

- verifying 2-18
- INOUT ports
  - specifying 6-2
- input 2-8
  - See also* input ports
- input ports
  - attaching to signals 3-15
  - for HDL model 2-8
  - for MATLAB component function 2-14
  - for MATLAB link function 2-14
  - for MATLAB test bench function 2-14
  - for test bench function 2-14
  - mapping signals to 3-46
  - simulation time for 3-15
- installation
  - of EDA Simulator Link™ MQ software 1-12
  - of related software 1-12
- installation of EDA Simulator Link™ MQ 1-12
- int64 values 2-40
- INTEGER data type 2-9
  - conversion of 2-20
  - converting to 2-26
- Internet address D-2
  - identifying server with D-2
  - specifying 2-40
- interprocess communication identifier 2-51
- ipc\_id status field
  - checking with hdldaemon function 2-51
  - description of 4-10
- iport parameter 2-14

## K

- kill option
  - description of 4-10
  - shutting down MATLAB server with 2-72

## L

- latency

- block 3-32
  - clock signal 3-32
- link function
  - associating with HDL module 2-37
  - matlabcp 2-12
  - matlabtb 2-12
  - matlabtbeval 2-12
- link functions
  - coding for HDL verification 2-12
  - programming for HDL verification 2-12
- link session. *See* link function
- link status
  - checking MATLAB server 2-51
  - function for acquiring 4-10
- links
  - MATLAB and HDL simulator 1-5
  - Simulink and HDL simulator 1-7

## M

- MATLAB
  - as required software 1-9
  - in EDA Simulator Link™ MQ cosimulation environment 1-5
  - installing 1-12
  - quitting 2-54
  - working with HDL simulator links to 1-5
- MATLAB component functions
  - adding to MATLAB search path 2-49
  - defining 2-14
  - specifying required parameters for 2-14
- MATLAB data types
  - conversion of 2-20
- MATLAB functions 4-1
  - coding for HDL verification 2-12
  - configuremodelsim
    - description of 4-2
  - dbstop 2-52
  - dec2mvl
    - description of 4-9

- defining 2-14
  - for MATLAB and ModelSim tutorial 2-66
  - hdlldaemon 2-49
    - description of 4-10
  - mv12dec
    - description of 4-17
  - naming 2-37
  - pingHdlSim
    - description of 4-18
  - programming for HDL verification 2-12
  - sample of 2-29
  - scheduling invocation of 2-40
  - specifying required parameters for 2-14
  - tc1HdlSim
    - description of 4-20
  - test bench 1-5
  - vsim 3-70
    - description of 4-21
  - which 2-49
- MATLAB link functions
- defining 2-14
  - specifying required parameters for 2-14
- MATLAB link sessions
- controlling
    - overview 2-48
  - starting
    - overview 2-48
- MATLAB search path 2-49
- MATLAB server
- checking link status with 2-51
  - configuration restrictions for C-2
  - configurations for C-2
  - function for invoking 1-5
  - identifying in a network configuration D-2
  - starting 2-49
  - starting for MATLAB and ModelSim tutorial 2-57
- MATLAB test bench functions
- defining 2-14
  - specifying required parameters for 2-14
- matlabcp command
- description of 5-2
- matlabtb command
- description of 5-8
  - initializing HDL simulator for MATLAB session 2-40
  - specifying scheduling options with 2-40
- matlabtbeval command
- description of 5-13
  - initializing HDL simulator for MATLAB session 2-40
  - specifying scheduling options with 2-40
- mfunc option
- specifying test bench function with 2-40
  - with matlabcp command 5-2
  - with matlabtb command 5-8
  - with matlabtbeval command 5-13
- models
- compiling VHDL 2-11
  - debugging VHDL 2-11
  - for Simulink and ModelSim tutorial 3-75
- ModelSim
- setting up for MATLAB and ModelSim tutorial 2-58
  - setting up for Simulink and ModelSim tutorial 3-84
- ModelSim commands
- vcd2w1f 3-66
  - vsimmatlab
    - description of 5-17
  - vsimulink
    - description of 5-18
  - wrapverilog 5-20
- ModelSim Editor 2-60
- ModelSim SE/PE
- as required software 1-9
  - in EDA Simulator Link™ MQ environment 1-7
  - installing 1-12

- invoking for use with EDA Simulator Link™
  - MQ software 1-23
  - setting up during installation 1-12
  - starting from MATLAB 2-51
  - working with Simulink links to 1-7
- ModelSim®
  - in EDA Simulator Link™ MQ cosimulation environment 1-5
  - working with MATLAB links to 1-5
- modes
  - communication 2-49
  - port direction 2-18
- module names
  - specifying paths
    - for MATLAB link sessions 2-38
    - in Simulink 3-41 6-2
- modules
  - coding for MATLAB verification 2-7
  - loading for verification 2-12
  - naming 2-8
- multirate signals 3-24
- mv12dec function
  - description of 4-17

## N

- names
  - for HDL modules 2-8
  - for test bench functions 2-37
  - shared memory communication channel 2-51
  - verifying port 2-18
- NATURAL data type 2-9
  - conversion of 2-20
  - converting to 2-26
- network configuration D-2
  - nocompile option 5-20
- nomatlabtb command 5-16
- Number of input ports parameter 6-21
- Number of output ports parameter

- description of 6-21
- numeric data
  - converting for MATLAB 2-24
  - converting for the HDL simulator 2-26

## O

- online help
  - where to find it 1-28
- oport parameter 2-14
- options
  - for matlabcp command 5-2
  - for matlabtb command 5-8
  - for matlabtbeval command 5-13
  - for vsimulink command 5-18
  - for wrapverilog command 5-20
  - kill 4-10
  - property
    - with configuremodelsim function 4-2
    - with hdldaemon function 4-10
    - with vsim function 4-21
  - status 4-10
- OS platform. *See* EDA Simulator Link™ MQ product requirements page on The MathWorks web site
- OUT direction mode 2-8
  - verifying 2-18
- output ports
  - for HDL model 2-8
  - for MATLAB component function 2-14
  - for MATLAB link function 2-14
  - for MATLAB test bench function 2-14
  - for test bench function 2-14
  - mapping signals to 3-46
  - simulation time for 3-15
- Output sample time parameter
  - description of 6-2
  - specifying sample time with 3-46

**P**

- parameters
  - for HDL Cosimulation block 6-2
  - for To VCD File block 6-21
  - required for MATLAB component functions 2-14
  - required for MATLAB link functions 2-14
  - required for MATLAB test bench functions 2-14
  - required for test bench functions 2-14
  - setting programmatically 3-64
- path specification
  - for ports/signals and modules
    - for MATLAB link sessions 2-38
    - in Simulink 3-41
  - for ports/signals and modules in Simulink with HDL Cosimulation block 6-2
- phase, clock 3-43
- pingHdlSim function
  - description of 4-18
- platform support
  - required 1-9
- port names
  - specifying paths
    - for MATLAB link sessions 2-38
    - in Simulink 3-41
  - specifying paths in Simulink with HDL Cosimulation block 6-2
  - verifying 2-18
- Port number or service parameter
  - description of 6-2
  - specifying block communication with 3-59
- port numbers D-2
  - checking 2-51
  - specifying for MATLAB server 2-49
  - specifying for the HDL simulator 2-40
- portinfo parameter 2-14
- portinfo structure 2-18
- ports
  - getting information about 2-14
  - specifying direction modes for 2-8
  - specifying VHDL data types for 2-9
  - using information about 2-18
  - verifying data type of 2-18
  - verifying direction modes for 2-18
- Ports pane
  - Auto fill option 6-2
  - configuring block ports with 3-46
  - description of 6-2
  - using Auto fill 3-46
- ports, block
  - mapping signals to 3-46
- Post- simulation command parameter
  - specifying block Tcl commands with 3-62
- Post-simulation command parameter
  - description of 6-2
- postprocessing tools 3-66
- Pre-simulation command parameter
  - description of 6-2
  - specifying block simulation Tcl commands with 3-62
- prerequisites
  - for using EDA Simulator Link™ MQ software 1-9
- properties
  - action 4-2
  - for configuremodelsim function 4-2
  - for hdldaemon function 4-10
  - for starting HDL simulator for use with Simulink 3-70
  - for starting MATLAB server 2-49
  - for vsim function 4-21
  - socket 4-10
  - socketsimulink 4-21
  - startupfile 4-21
  - tclstart
    - with configuremodelsim function 4-2
    - with vsim function 4-21
  - time
    - description of 4-10

- vsimdir
  - with configuremodelsim function 4-2
  - with vsim function 4-21
- property option
  - for configuremodelsim function 4-2
  - for hdldaemon function 4-10
  - for vsim function 4-21

## R

- race conditions
  - in HDL simulation 3-32
- rate converter 3-24
- real data
  - converting for MATLAB 2-24
  - converting for the HDL simulator 2-26
- REAL data type 2-9
  - conversion of 2-20
  - converting to 2-26
- real values, as time 2-40
- relative timing mode
  - definition of 3-18
  - operation of 3-18
- repeat option 5-2
  - specifying scheduling options with 2-40
- requirements
  - application software 1-9
  - checking product 1-9
  - platform 1-9
- resolution functions 3-15
- resolution limit 2-18
- rising option 5-2
  - specifying scheduling options with 2-40
- rising-edge clocks
  - creating for HDL Cosimulation blocks 3-43
  - description of 6-2
  - specifying as scheduling options 2-40
- Rising-edge clocks parameter
  - specifying block clocks with 3-43
- run command 2-52

Run option, MATLAB 2-52

## S

- sample periods 3-5
  - See also* sample times
- sample times 3-32
  - design decisions for 3-5
  - handling across simulation domains 3-14
  - specifying for block output ports 3-46
- Sample-based processing 3-24
- Save and Run option, MATLAB 2-52
- scalar data types
  - conversions of 2-20
  - VHDL 2-9
- scheduling options 2-40
- script
  - HDL simulator setup 1-12
- search path 2-49
- sensitivity lists 2-40
  - sensitivity option 5-2
    - specifying scheduling options with 2-40
- server activation 4-10
- server shutdown 4-10
- server, MATLAB
  - checking link status of MATLAB 2-51
  - for MATLAB and HDL simulator links 1-5
  - for Simulink and HDL simulator links 1-7
  - identifying in a network configuration D-2
  - starting for MATLAB and ModelSim
    - tutorial 2-57
  - starting MATLAB 2-49
- Set/Clear Breakpoint option, MATLAB 2-52
- set\_param
  - for specifying post-simulation Tcl commands 3-62
- shared memory communication 1-8
  - as a configuration option for MATLAB C-2
  - as a configuration option for Simulink C-4
  - for Simulink applications 3-70



- specifying for HDL Cosimulation blocks 3-59
- specifying with `hdldaemon` function 2-49
- Shared memory parameter
  - description of 6-2
  - specifying block communication with 3-59
- signal datatypes
  - specifying 3-56
- signal names
  - specifying paths
    - for MATLAB link sessions 2-38
    - in Simulink 3-41
  - specifying paths in Simulink
    - with HDL Cosimulation block 6-2
- signal path names
  - displaying 3-46
  - specifying for block clocks 3-43
  - specifying for block ports 3-46
- Signal Processing Blockset
  - as optional software 1-9
  - using for EDA applications 3-5
- signals
  - continuous 3-15
  - defining ports for 2-8
  - driven by multiple sources 3-15
  - exchanging between simulation domains 3-14
  - handling across simulation domains 3-14
  - how Simulink drives 3-15
  - logging changes to 3-66
  - logging changes to values of 3-66
  - mapping to block ports 3-46
  - multirate 3-24
  - read/write access
    - mapping 3-46
    - required 3-15
  - read/write access required 6-2
- signed data 2-24
- SIGNED data type 2-26
- simulation analysis 3-66
- simulation time 2-14
  - guidelines for 3-15
  - representation of 3-15
  - scaling of 3-15
- simulations
  - comparing results of 3-66
  - ending 2-54
  - loading for MATLAB and ModelSim
    - tutorial 2-63
  - logging changes to signal values during 3-66
  - quitting 2-54
  - running for MATLAB and ModelSim
    - tutorial 2-68
  - running Simulink and ModelSim
    - tutorial 3-86
  - shutting down for MATLAB and ModelSim
    - tutorial 2-72
  - shutting down Simulink and ModelSim
    - tutorial 3-89
- simulator communication
  - options 3-39
- simulator resolution limit 2-18
- simulators
  - handling of signal values between 3-14
  - HDL simulator
    - initializing for MATLAB session 2-40
  - ModelSim SE/PE
    - starting from MATLAB 2-51
- Simulink
  - as optional software 1-9
  - configuration restrictions for C-4
  - configuring for HDL models 3-38
  - creating hardware model designs with 3-5
  - driving cosimulation signals with 3-15
  - in EDA Simulator Link™ MQ
    - environment 1-7
  - installing 1-12
  - running and testing hardware model in 3-39
  - setting up ModelSim for use with 3-84
  - simulation time for 3-15
  - starting the HDL simulator for use with 3-70

- working with HDL simulator links to 1-7
- Simulink Fixed Point
  - as optional software 1-9
  - using for EDA applications 3-5
- Simulink models
  - adding HDL models to 3-8
  - for Simulink and ModelSim tutorial 3-75
- sink device
  - adding to a Simulink model 3-8
  - specifying block ports for 3-46
  - specifying clocks for 3-43
  - specifying communication for 3-59
  - specifying Tcl commands for 3-62
- socket numbers 2-51
  - See also* port numbers
- socket option
  - specifying TCP/IP socket with 2-40
  - with `matlabcp` command 5-2
  - with `matlabtb` command 5-8
  - with `matlabtbeval` command 5-13
  - with `vsimulink` command 5-18
- socket port numbers D-2
  - as a networking requirement D-2
  - checking 2-51
  - specifying for HDL Cosimulation blocks 3-59
  - specifying for TCP/IP link 3-70
  - specifying with `-socket` option 2-40
- socket property
  - description of 4-10
  - specifying with `hdldaemon` function 2-49
- sockets 1-8
  - See also* TCP/IP socket communication
- socketsimulink property
  - description of 4-21
  - specifying TCP/IP socket for HDL simulator with 3-70
- software
  - installing EDA Simulator Link™ MQ 1-12
  - installing related application software 1-12
  - optional 1-9
  - required 1-9
- source device
  - adding to a Simulink model 3-8
  - specifying block ports for 3-46
  - specifying clocks for 3-43
  - specifying communication for 3-59
  - specifying Tcl commands for 3-62
- standard logic data 2-24
- standard logic vectors
  - converting for MATLAB 2-24
  - converting for the HDL simulator 2-26
- start time 3-15
- startupfile property
  - description of 4-21
  - specifying DO file for ModelSim startup with 3-70
- status option
  - checking value of 2-51
  - description of 4-10
- status, link 2-51
- STD\_LOGIC data type 2-9
  - conversion of 2-20
  - converting to 2-26
- STD\_LOGIC\_VECTOR data type 2-9
  - conversion of 2-20
  - converting for MATLAB 2-24
  - converting to 2-26
- STD\_ULONGIC data type 2-9
  - conversion of 2-20
  - converting to 2-26
- STD\_ULONGIC\_VECTOR data type 2-9
  - conversion of 2-20
  - converting for MATLAB 2-24
  - converting to 2-26
- Step button
  - MATLAB 2-52
- Step-In button, MATLAB 2-52
- Step-Out button, MATLAB 2-52
- stimuli, block internal 3-43
- stop time 3-15

strings, time value 2-40  
 subtypes, VHDL 2-9

## T

### Tcl commands

configuring for block simulation 3-62  
 configuring the HDL simulator to start  
   with 3-70  
 post-simulation  
   using set\_param 3-62  
 pre-simulation  
   using set\_param 3-62  
 specified in Tcl pane of HDL Cosimulation  
   block 6-2

### Tcl pane

description of 6-2

### tclHdlSim function

description of 4-20

### tclstart property

with configuremodelsim function 4-2  
 with vsim function 4-21

### TCP/IP networking protocol 1-8

as a networking requirement D-2  
*See also* TCP/IP socket communication

### TCP/IP socket communication

as a communication option for MATLAB C-2  
 as a communication option for Simulink C-4  
 for Simulink applications 3-70  
 mode 1-8  
 specifying with hlldaemon function 2-49

### TCP/IP socket ports D-2

specifying for HDL Cosimulation blocks 3-59  
 specifying with -socket option 2-40

### test bench functions

adding to MATLAB search path 2-49  
 coding for HDL verification 2-12  
 defining 2-14  
 for MATLAB and ModelSim tutorial 2-66  
 naming 2-37

programming for HDL verification 2-12  
 sample of 2-29  
 scheduling invocation of 2-40  
 specifying required parameters for 2-14

### test bench sessions

logging changes to signal values during 3-66  
 monitoring 2-52  
 restarting 2-54  
 running 2-52  
 stopping 2-54

### The HDL simulator is running on this computer parameter

specifying block communication with 3-59

### time 3-15

callback 2-14  
 delta 3-32  
 simulation 2-14  
   guidelines for 3-15  
   representation of 3-15

*See also* time values

### TIME data type 2-9

conversion of 2-20  
 converting to 2-26

### time property

description of 4-10  
 setting return time type with 2-49

### time scale, VCD file 6-25

### time units 2-40

### time values 2-40

specifying as scheduling options 2-40  
 specifying with hlldaemon function 2-49

### Timescales pane

description of 6-2

### timing errors 3-15

### Timing mode

absolute 3-58  
 configuring for cosimulation 3-58  
 relative 3-58

### tnext parameter 2-14

controlling callback timing with 2-40

- specifying as scheduling options 2-40
- time representations for 2-40
- tnow parameter 2-14
- To VCD File block
  - description of 6-21
  - uses of 1-7
- tools, postprocessing 3-66
- tscale parameter 2-18
- tutorial files 2-56
- tutorials 1-30
  - for EDA Simulator Link™ MQ 1-28
  - Simulink and ModelSim 3-72
  - VCD file generation 3-90

## U

- unsigned data 2-24
- UNSIGNED data type 2-26
- unsupported data types 2-9
- users
  - for EDA Simulator Link™ MQ software 1-9

## V

- value change dump (VCD) files 3-66
  - See also* VCD files
- VCD file name parameter
  - description of 6-21
- VCD files
  - example of generating 3-90
  - format of 6-25
  - using 3-66
- vcd2wlf command 3-66
- vectors
  - converting for MATLAB 2-24
  - converting to 2-26
- verification
  - coding test bench functions for 2-12
- verification sessions
  - logging changes to signal values during 3-66

- monitoring 2-52
- restarting 2-54
- running 2-52
- stopping 2-54
- Verilog data types
  - conversion of 2-20
- Verilog modules
  - coding for MATLAB verification 2-7
- VHDL code
  - compiling for MATLAB and ModelSim tutorial 2-63
  - compiling for Simulink and ModelSim tutorial 3-73
  - for MATLAB and ModelSim tutorial 2-60
  - for Simulink and ModelSim tutorial 3-72
- VHDL data types
  - conversion of 2-20
- VHDL entities
  - coding for MATLAB verification 2-7
  - for Simulink and ModelSim tutorial
    - loading for cosimulation 3-85
    - sample definition of 2-10
    - verifying port direction modes for 2-18
- VHDL models 1-3
  - compiling 2-11
  - debugging 2-11
  - See also* HDL models
- visualization
  - coding functions for 2-12
- vsim
  - for ModelSim
    - using configuration file with 1-25
    - using 1-23
    - using -foreign option 1-25
- vsim function
  - description of 4-21
  - starting ModelSim with 2-51
- vsimdir property
  - with configuremodelsim function 4-2
  - with vsim function 4-21

vsimmatlab command  
  description of 5-17  
  loading HDL modules for verification  
    with 2-12  
vsimulink command  
  description of 5-18

## **W**

Wave Log Format (WLF) files 3-66  
wave window, HDL simulator 3-46  
waveform files 3-66  
which function 2-49

WLF files 3-66  
workflow  
  EDA Simulator Link™ MQ with  
    MATLAB 1-26  
  EDA Simulator Link™ MQ with  
    Simulink 1-27  
  ModelSim SE/PE with MATLAB 2-2  
  ModelSim SE/PE with Simulink 3-2  
wrapverilog command 5-20

## **Z**

zero-order hold 3-15