

# Link for ModelSim<sup>®</sup>

For Use with MATLAB<sup>®</sup> and Simulink<sup>®</sup>

- Computation
- Visualization
- Programming
- Simulation

User's Guide

*Version 1*



## How to Contact The MathWorks:



www.mathworks.com  
comp.soft-sys.matlab

Web  
Newsgroup



support@mathworks.com  
suggest@mathworks.com  
bugs@mathworks.com  
doc@mathworks.com  
service@mathworks.com  
info@mathworks.com

Technical Support  
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

Mail

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

### *Link for ModelSim® User's Guide*

© COPYRIGHT 2003–2004 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 or for the federal government of the United States. By accepting delivery of the Program, the government hereby agrees that this software qualifies as "commercial" computer software within the meaning of FAR Part 12.212, DFARS Part 227.7202-1, DFARS Part 227-7202-3, DFARS Part 252.227-7013, and DFARS Part 252.227-7014. The terms and conditions of The MathWorks, Inc. Software License Agreement shall pertain to the government's use and disclosure of the Program and Documentation, and shall supersede any conflicting contractual terms or conditions. If this license fails to meet the government's minimum needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to MathWorks.

MATLAB, Simulink, Stateflow, Handle Graphics, and Real-Time Workshop are registered trademarks, and TargetBox is a trademark of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Printing History: August 2003      Online only      New for Version 1 (Release 13SP1)  
February 2004      Online only      Updated for Version 1.1 (Release  
13SP1)

## Getting Started

1

<b>What Is the Link for ModelSim?</b> .....	<b>1-2</b>
Typical Applications .....	1-3
Expected Users .....	1-4
Key Features .....	1-5
The Cosimulation Environment .....	1-5
Modes of Communication .....	1-8
Working with MATLAB and ModelSim .....	1-8
Working with Simulink and ModelSim .....	1-9
<b>Installation and Setup</b> .....	<b>1-11</b>
What Are Your Environment Requirements? .....	1-12
Deciding on a Configuration .....	1-15
Identifying a Server in a Network	
Configuration .....	1-17
Choosing TCP/IP Socket Ports .....	1-17
Checking Product Requirements .....	1-18
Installing Related Application Software .....	1-19
Installing Link for ModelSim .....	1-19
Setting Up ModelSim for Use with the Link for	
ModelSim .....	1-19
<b>Getting Help with the Link for ModelSim</b> .....	<b>1-24</b>
Documentation Overview .....	1-24
Online Help .....	1-25
Demos and Tutorials .....	1-25
<b>Running the ModelSim and MATLAB Random Number</b>	
<b>Generator Demo</b> .....	<b>1-27</b>
<b>Running the Simulink and ModelSim Manchester Receiver</b>	
<b>Demo</b> .....	<b>1-32</b>

## MATLAB and ModelSim Tutorial

---

### 2

<b>Setting Up Tutorial Files</b> .....	<b>2-3</b>
<b>Starting the MATLAB Server</b> .....	<b>2-4</b>
<b>Setting Up ModelSim</b> .....	<b>2-6</b>
<b>Developing the VHDL Code</b> .....	<b>2-8</b>
<b>Compiling the VHDL File</b> .....	<b>2-11</b>
<b>Loading the Simulation</b> .....	<b>2-12</b>
<b>Developing the MATLAB Function</b> .....	<b>2-15</b>
<b>Running the Simulation</b> .....	<b>2-18</b>
<b>Shutting Down the Simulation</b> .....	<b>2-21</b>

## Simulink and ModelSim Tutorial

---

### 3

<b>Developing the VHDL Code</b> .....	<b>3-2</b>
<b>Compiling the VHDL File</b> .....	<b>3-4</b>
<b>Creating the Simulink Model</b> .....	<b>3-6</b>
<b>Setting Up ModelSim for Use with Simulink</b> .....	<b>3-12</b>
<b>Loading Instances of the VHDL Entity for Cosimulation with Simulink</b> .....	<b>3-13</b>

<b>Running the Simulation</b> .....	<b>3-14</b>
<b>Shutting Down the Simulation</b> .....	<b>3-17</b>

## **MATLAB and ModelSim Manchester Receiver Tutorial**

# 4

<b>Background on Manchester Encoding</b> .....	<b>4-3</b>
The Encoding .....	<b>4-3</b>
The Receiver .....	<b>4-5</b>
Decoding with Inphase and Quadrature Convolution .....	<b>4-6</b>
<b>Setting Up Tutorial Files</b> .....	<b>4-8</b>
<b>Developing the Manchester Receiver VHDL</b>	
<b>Code</b> .....	<b>4-9</b>
VHDL Code for the I/Q Convolver .....	<b>4-11</b>
VHDL Code for the Decoder .....	<b>4-14</b>
VHDL Code for the State Counter .....	<b>4-15</b>
<b>Compiling the Manchester Receiver VHDL</b>	
<b>Files</b> .....	<b>4-17</b>
<b>Developing the Manchester Receiver MATLAB</b>	
<b>Functions</b> .....	<b>4-19</b>
MATLAB Function for the I/Q Convolver .....	<b>4-19</b>
MATLAB Function for the Decoder .....	<b>4-23</b>
MATLAB Function for the State Counter .....	<b>4-26</b>
<b>Creating a Manchester Receiver Test Bench</b>	
<b>Script</b> .....	<b>4-30</b>
Documenting the Script .....	<b>4-30</b>
Starting the MATLAB Server from the Test Script .....	<b>4-31</b>
Writing Script Code for the Decoder Test .....	<b>4-31</b>
Writing Script Code for the I/Q Convolver Test .....	<b>4-34</b>

Writing Script Code for the State Counter Test ..... 4-36

**Running the Manchester Receiver**

**Simulation** ..... 4-40

**Coding a Link for ModelSim MATLAB Application**

**5**

**Overview** ..... 5-2

**Coding VHDL Entities for MATLAB**

**Verification** ..... 5-3

Overview of the Steps for Coding VHDL Entities ..... 5-3

Choosing an Entity Name ..... 5-3

Specifying Ports for the Entity ..... 5-4

Specifying Port Direction Modes ..... 5-4

Specifying Port Data Types ..... 5-5

Sample VHDL Entity Definition ..... 5-5

**Compiling and Debugging the VHDL Model** ..... 5-7

**Coding a MATLAB Test Bench Function** ..... 5-8

Overview of the Steps for Coding a MATLAB Test Bench

Function ..... 5-8

Data Type Conversions ..... 5-9

Naming a MATLAB Test Bench Function ..... 5-13

Setting up Expected Parameters ..... 5-13

Gaining Access to and Applying Port

Information ..... 5-15

Converting Data for Manipulation ..... 5-17

Converting Data for Return to ModelSim ..... 5-18

Sample MATLAB Test Bench Function ..... 5-22

**Placing a MATLAB Test Bench Function on the MATLAB**

**Search Path** ..... 5-28

## Starting and Controlling MATLAB Test Bench Sessions

# 6

<b>Overview</b> .....	<b>6-3</b>
<b>Checking the MATLAB Server's Link Status</b> .....	<b>6-5</b>
<b>Starting the MATLAB Server</b> .....	<b>6-7</b>
<b>Starting ModelSim for Use with MATLAB</b> .....	<b>6-10</b>
<b>Loading a VHDL Entity for Verification</b> .....	<b>6-12</b>
<b>Deciding on Test Bench Scheduling Options</b> .....	<b>6-13</b>
<b>Controlling Callback Timing from a MATLAB Test Bench Function</b> .....	<b>6-14</b>
<b>Initializing the Simulator for a MATLAB Test Bench Session</b> .....	<b>6-16</b>
<b>Applying Stimuli with the ModelSim force Command</b> .....	<b>6-21</b>
<b>Running and Monitoring a Test Bench Session</b> .....	<b>6-22</b>
<b>Restarting a Test Bench Session</b> .....	<b>6-25</b>
<b>Stopping a Test Bench Session</b> .....	<b>6-26</b>

## Modeling and Verifying a VHDL Design with Simulink

# 7

<b>Overview</b> .....	<b>7-3</b>
<b>Creating a Hardware Model Design in Simulink</b> .....	<b>7-5</b>
<b>Handling of Signal Values Across Simulation</b>	
<b>Domains</b> .....	<b>7-8</b>
How Simulink Drives Cosimulation Signals .....	<b>7-8</b>
Representation of Simulation Time .....	<b>7-9</b>
Handling of Multirate Signals .....	<b>7-10</b>
Block Simulation Latency .....	<b>7-11</b>
<b>Configuring Simulink for VHDL Models</b> .....	<b>7-17</b>
<b>Running and Testing a Hardware Model in Simulink</b> .....	<b>7-19</b>
<b>Starting ModelSim for Use with Simulink</b> .....	<b>7-20</b>
<b>Loading a VHDL Entity for Cosimulation</b> .....	<b>7-23</b>
<b>Adding the VHDL Representation of a Model Component into a Simulink Model</b> .....	<b>7-24</b>
<b>Configuring a VHDL Cosimulation Block</b> .....	<b>7-26</b>
What Are Your VHDL Cosimulation Block Requirements? .....	<b>7-26</b>
Opening the Block Parameters Dialog .....	<b>7-28</b>
Mapping VHDL Signals to Block Ports .....	<b>7-29</b>
Configuring the Communication Link .....	<b>7-33</b>
Creating Optional Clocks .....	<b>7-35</b>
Specifying Before and After Simulation Tcl Commands .....	<b>7-37</b>
Applying Your Block Parameters Configuration Settings and Closing the Dialog .....	<b>7-40</b>
<b>Running and Testing a Cosimulation Model in Simulink</b> .....	<b>7-42</b>



**Using a Value Change Dump File for Design**

<b>Verification</b> .....	<b>7-43</b>
Generating a VCD File .....	<b>7-44</b>
VCD File Format .....	<b>7-45</b>
A Sample VCD File Application .....	<b>7-48</b>

**MATLAB Functions — Alphabetical List**

**8**

**ModelSim Commands — Alphabetical List**

**9**

**Simulink Blocks — Alphabetical List**

**10**

**Index**



# Getting Started

---

“What Is the Link for ModelSim?”  
(p. 1–2)

Identifies typical applications and expected users, lists key product features, describes the Link for ModelSim cosimulation environment, and provides overviews of how you work with the integrated tool environment.

“Installation and Setup” (p. 1–11)

Explains how to install and set up the Link for ModelSim.

“Getting Help with the Link for ModelSim” (p. 1–24)

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

“Running the ModelSim and MATLAB Random Number Generator Demo” (p. 1–27)

Explains how to run a MATLAB and ModelSim demo.

“Running the Simulink and ModelSim Manchester Receiver Demo” (p. 1–32)

Explains how to run a Simulink and ModelSim demo.

## What Is the Link for ModelSim?

The Link for ModelSim® is a cosimulation interface that integrates MathWorks tools into the Electronic Design Automation (EDA) workflow for field programmable gate array (FPGA) and application-specific integrated circuit (ASIC) development. The interface provides a fast bidirectional link between Model Technology's hardware definition language (HDL) Simulator, ModelSim SE/PE, and the MathWorks products MATLAB® and Simulink® 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

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

The following sections discuss

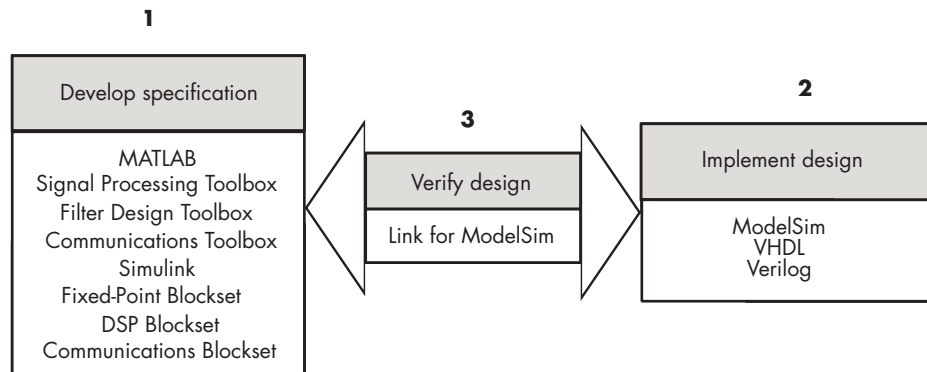
- “Typical Applications” on page 1-3
- “Expected Users” on page 1-4
- “Key Features ” on page 1-5
- “The Cosimulation Environment” on page 1-5
- “Modes of Communication” on page 1-8
- “Working with MATLAB and ModelSim” on page 1-8
- “Working with Simulink and ModelSim” on page 1-9

## Typical Applications

The Link for ModelSim 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 ModelSim and MathWorks products fit into this hardware design scenario.



As the figure shows, the Link for ModelSim connects tools that traditionally have been used discretely to accomplish specific steps in the design process. By connecting the tools, Link for ModelSim 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, the Link for ModelSim enables you to use

- 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

## **Expected Users**

The Link for ModelSim is for hardware engineers who design, implement, or verify FPGAs and ASICs. A typical user might be responsible for any or all of the following:

- Create hardware reference specifications, using MATLAB or Simulink
- Develop implementations of the specifications in HDL, using ModelSim
- Verify the implementation by comparing its results to those of the original specification

The Link for ModelSim enables engineers to cosimulate and verify a design directly between the specification and implementation, eliminating the need for manual comparisons. Link for ModelSim also allows designers to pass on MATLAB and Simulink specifications to implementation and verification teams, without having to first rewrite the design in HDL.

The documentation provided with the Link for ModelSim assumes users have a moderate level of prerequisite knowledgeable in the following subject areas:

- Hardware design and system integration
- VHDL
- ModelSim SE/PE
- MATLAB

Experience with Simulink and the Fixed-Point Blockset 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

DSP Blockset

Filter Design Toolbox

Communications Blockset

Communications Toolbox

## Key Features

Key features of Link for ModelSim include

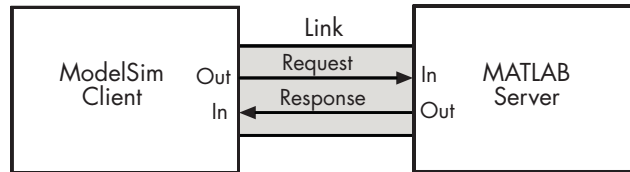
- Ability to link ModelSim to MATLAB and Simulink for bidirectional cosimulation, verification, and visualization
- Support for PE and SE versions of ModelSim
- Support for Linux, Solaris, Windows 2000, and Windows XP platforms
- Support for shared memory and TCP/IP socket modes of communication between MATLAB and Simulink and ModelSim
- A Simulink block for cosimulating HDL models in Simulink
- A Simulink block for exporting test vectors and results as value change dump (VCD) files
- Support for multiple simultaneous ModelSim instances, and multiple HDL entities from within one Simulink model or MATLAB function
- Interactive or batch mode cosimulation, debugging, testing, and verification of HDL code from within MATLAB

## The Cosimulation Environment

The Link for ModelSim is a client/server test bench and cosimulation application. The role that ModelSim plays in a Link for ModelSim simulation environment depends on whether ModelSim links to MATLAB or Simulink.

### MATLAB and ModelSim Links

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



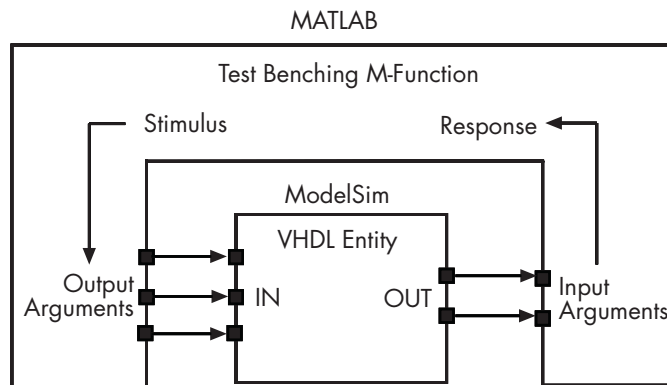
In this scenario, a MATLAB server function waits for service requests that it receives from a ModelSim simulator session. After receiving a request, the server establishes a communication link, and invokes a specified MATLAB function wrapper that computes data for, verifies, or visualizes the HDL model that is under simulation in ModelSim.

---

**Note** You cannot initiate Link for ModelSim communication between MATLAB and ModelSim from MATLAB. The MATLAB server simply responds to function call requests that it receives from ModelSim.

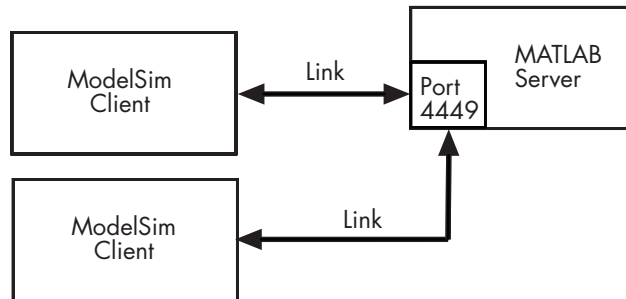
---

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



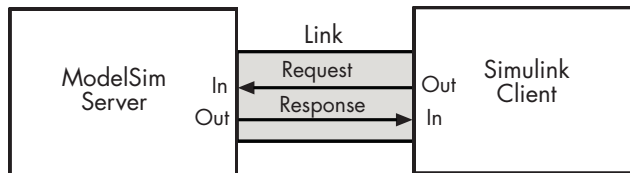
The MATLAB server can service multiple simultaneous ModelSim sessions and HDL entities. However, you should adhere to recommended guidelines to ensure the server can track the I/O associated with each entity and session. The following figure shows a multiple-client scenario connecting to the server at TCP/IP socket port 4449.





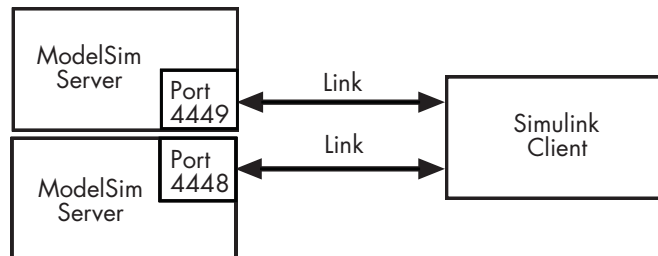
### Simulink and ModelSim Links

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



In this case, ModelSim responds to simulation requests it receives from cosimulation blocks in a Simulink model. You initiate a cosimulation session from Simulink. Once a session is started, you can use Simulink and ModelSim to monitor simulation progress and results. For example, you might add signals to a ModelSim 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 ModelSim, using unique TCP/IP socket ports.



## **Modes of Communication**

The mode of communication that the Link for ModelSim uses for a link between ModelSim and MATLAB or Simulink somewhat depends on whether your simulation application runs in a local, single-system configuration or in a network configuration. If ModelSim 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. It is the optimal choice for applications that have growth potential.

For configurations in which ModelSim and the MathWorks products reside on different systems, each system must be configured for the Ethernet and you must use TCP/IP socket communication.

## **Working with MATLAB and ModelSim**

When linked with MATLAB, ModelSim functions as the client, initiating requests of MATLAB that focus on numerical computing, algorithm development, and visualization. The MATLAB server, which you start with a supplied MATLAB function, waits for connection requests from instances of ModelSim running on the same or different computers. When the server receives a request, it executes a specified wrapper MATLAB function you have coded to perform tasks on behalf of an entity in your VHDL design. Parameters that you specify when you start the server indicate whether the server establishes shared memory or TCP/IP socket communication links.

Once the server is running, you can start and configure ModelSim for use with MATLAB with a supplied Link for ModelSim function. Optional parameters allow you to specify

- 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

During the configuration process, Link for ModelSim equips ModelSim with a set of Link for ModelSim command extensions you use to

- Load the ModelSim simulator, `vsim`, with an instance of a VHDL entity to be tested with MATLAB
- Initiate a MATLAB test bench session for that instance

When you initiate a specific test bench session, you specify parameters that identify

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

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

## Working with Simulink and ModelSim

When linked with Simulink, ModelSim functions as the server. Using the Link for ModelSim communications interface, a VHDL Cosimulation block cosimulates a hardware component by applying input signals to and reading output signals from a VHDL model under simulation in ModelSim. Multiple VHDL Cosimulation blocks in a Simulink model can request the service of multiple instances of ModelSim, using unique TCP/IP socket ports.

Using the **Block Parameters** dialog for a VHDL Cosimulation block, you can configure the following:

- Block input and output ports that correspond to signals, including internal signals of a VHDL model, and an output sample time for block output ports
- Type of communication and communication settings used for exchanging data between the simulation tools

- Rising-edge or falling-edge clocks to apply to your model
- Tcl commands that you want to run before and after the simulation

Using a Link for ModelSim MATLAB function, you can start ModelSim with necessary configurations. Optional parameters allow you to specify

- 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 default mode of communication to be used for the link and, if appropriate, a TCP/IP socket port

During the configuration process, Link for ModelSim equips ModelSim with a set of Link for ModelSim command extensions. Using one of those commands, you execute the ModelSim simulator with an instance of a VHDL entity for cosimulation with Simulink. Once the entity is loaded, you can start the cosimulation session from Simulink.

The Link for ModelSim also includes a block for generating value change dump (VCD) files. You might use VCD files generated with this block

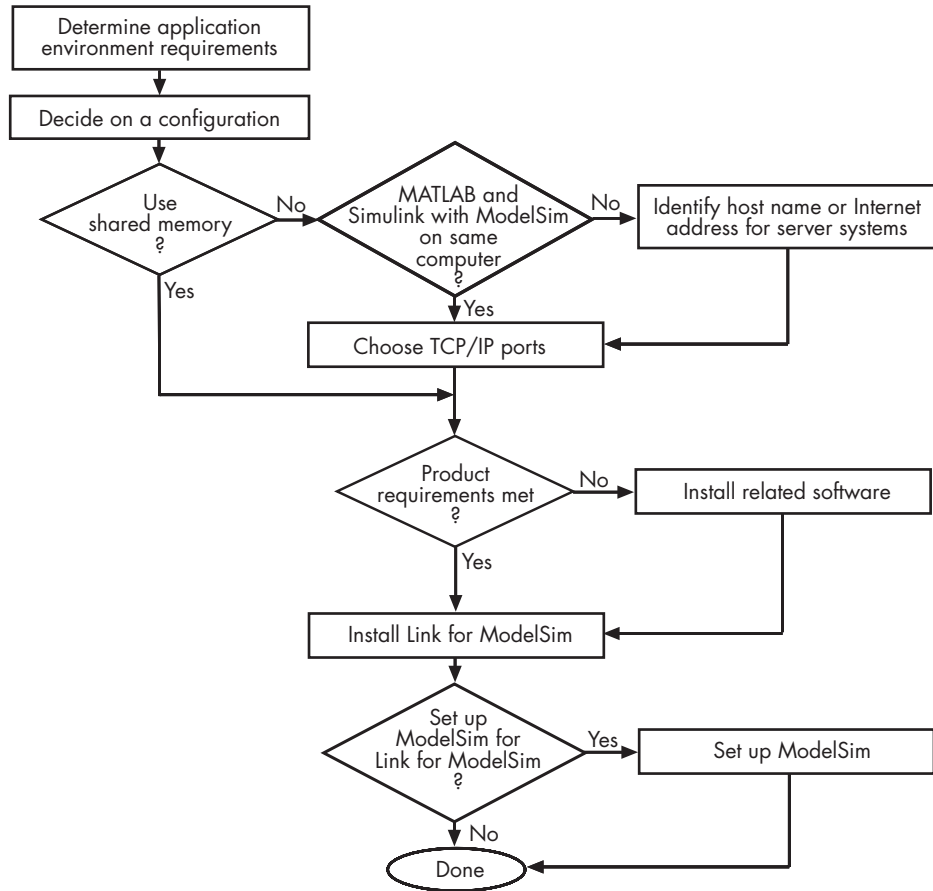
- To view Simulink simulation waveforms in your HDL simulation environment
- To compare results of multiple simulation runs, using the same or different simulation environments
- As input to post-simulation analysis tools

## Installation and Setup

This section helps you to define your Link for ModelSim application environment. Topics include

- “What Are Your Environment Requirements?” on page 1–12
- “Deciding on a Configuration” on page 1–15
- “Identifying a Server in a Network Configuration” on page 1–17
- “Choosing TCP/IP Socket Ports” on page 1–17
- “Checking Product Requirements” on page 1–18
- “Installing Related Application Software” on page 1–19
- “Installing Link for ModelSim” on page 1–19
- “Setting Up ModelSim for Use with the Link for ModelSim” on page 1–19

The following figure summarizes the installation and setup process in a flow diagram. Topics that follow explain the steps in more detail.



## What Are Your Environment Requirements?

As part of the installation and setup process, review the following checklist. It will help you identify environment requirements that pertain to your Link for ModelSim application. If your answer to a question is “yes,” go to the topic listed in the second column of the table for information on how to address the requirement.

## Environment Requirements Checklist

Requirement	For More Information, See...
<b>Configurations</b>	
<input type="checkbox"/> Will your application use ModelSim with the MATLAB, Simulink, or both MATLAB and Simulink?	“Deciding on a Configuration” on page 1–15
<input type="checkbox"/> Will your application use multiple communication links?	“Deciding on a Configuration” on page 1–15
<input type="checkbox"/> How many instances of the MATLAB server are required?	“Deciding on a Configuration” on page 1–15
<input type="checkbox"/> Will a MATLAB server be handling multiple ModelSim client connections? If so, how many? Will they be from the same or different ModelSim sessions?	“Deciding on a Configuration” on page 1–15
<input type="checkbox"/> How many MATLAB functions do you need to write to model your VHDL implementation?	“Deciding on a Configuration” on page 1–15
<input type="checkbox"/> If your application will be using Simulink, how many cosimulation blocks are needed? Will the blocks be connecting to the same or different ModelSim sessions?	“Deciding on a Configuration” on page 1–15
<input type="checkbox"/> To how many ModelSim sessions will your Simulink model connect?	“Deciding on a Configuration” on page 1–15
<b>Mode of Communication</b>	
<input type="checkbox"/> Is performance the highest priority for your application? If so, can you run MATLAB and Simulink and ModelSim on the same computer system?	“Modes of Communication” on page 1–8
<input type="checkbox"/> Does your application require only one communication link (channel) on a single computing system?	“Modes of Communication” on page 1–8
<input type="checkbox"/> Is configuration flexibility a high priority for your application? Does the application have growth potential?	“Modes of Communication” on page 1–8

Requirement	For More Information, See...
<input type="checkbox"/> Do you prefer to use the TCP/IP socket mode of communication for a single-computer configuration? If so, do you want the Link for ModelSim to identify an available socket port on the system or do you want to choose a socket port yourself?	“Choosing TCP/IP Socket Ports” on page 1–17
<b>Network Configurations</b>	
<input type="checkbox"/> Have you identified the computer systems that will function as Link for ModelSim servers?	“Identifying a Server in a Network Configuration” on page 1–17
<input type="checkbox"/> What is the Internet address or host name of each computer system that will function as a server?	“Identifying a Server in a Network Configuration” on page 1–17
<input type="checkbox"/> Do you want the Link for ModelSim to identify an available TCP/IP socket port on server systems for establishing communication links? Or, do you want to choose or identify a TCP/IP socket ports yourself?	“Choosing TCP/IP Socket Ports” on page 1–17
<b>Related Software</b>	
<input type="checkbox"/> Is ModelSim installed on all systems as needed for your application?	“Installing Related Application Software” on page 1–19
<input type="checkbox"/> Is MATLAB installed on all systems as needed for your application?	“Installing Related Application Software” on page 1–19
<input type="checkbox"/> Does your application require the use of any toolboxes? If so, are the toolboxes installed on all systems as needed for your application?	“Installing Related Application Software” on page 1–19
<input type="checkbox"/> Will you be using the Simulink component of the Link for ModelSim? If so, is Simulink and the Fixed-Point Blockset installed on all systems as needed for your application? Are the required blocksets installed?	“Installing Related Application Software” on page 1–19
<b>ModelSim Setup</b>	
<input type="checkbox"/> Do you want to set up ModelSim such that it always starts ready for use with MATLAB and Simulink?	“Setting Up ModelSim for Use with the Link for ModelSim” on page 1–19



## Deciding on a Configuration

As you consider various configurations for an application, keep the following general guidelines in mind:

- 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 Link for ModelSim MATLAB server (hdldaemon) at a time.
- In a TCP/IP configuration, the MATLAB server can handle multiple client connections to one or more ModelSim sessions.
- VHDL Cosimulation blocks in a Simulink model can connect to the same or different ModelSim sessions.
- When using both MATLAB and Simulink, you must use different TCP/IP ports for links between these products and ModelSim.

The following lists provide samples of valid configurations for using ModelSim with MATLAB and Simulink, respectively. The scenarios apply whether ModelSim is running on the same or different computing system as MATLAB or Simulink. 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.

### MATLAB

The following list gives a sampling of valid configurations for using ModelSim with MATLAB:

- A ModelSim session linked to a MATLAB function foo through a single instance of the MATLAB server
- A ModelSim session linked to multiple MATLAB functions (for example, foo and bar) through a single instance of the MATLAB server

- A ModelSim 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 ModelSim 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 ModelSim sessions each linked to a different MATLAB function (for example, `foo` and `bar`) through the same instance of the MATLAB server
- Multiple ModelSim sessions each linked to MATLAB function `foo` through a single instance of the MATLAB server

---

**Note** Although multiple ModelSim sessions can link to the same MATLAB function in the same instance of the MATLAB server, as the last 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 VHDL entity.

---

## Simulink

The following list gives a sampling of valid local configurations for using Simulink with ModelSim:

- A VHDL Cosimulation block in a Simulink model linked to a single ModelSim session
- Multiple VHDL Cosimulation blocks in a Simulink model linked to the same ModelSim session
- A VHDL Cosimulation block in a Simulink model linked to multiple ModelSim sessions
- Multiple VHDL Cosimulation blocks in a Simulink model linked to different ModelSim sessions

## Identifying a Server in a Network Configuration

If you need to set up your Link for ModelSim application such that ModelSim and the MathWorks products reside on different systems, you must set up the systems to use

- TCP/IP networking protocol
- Link for ModelSim TCP/IP socket mode of communication

As part of your application setup, you must identify

- The Internet address or host name of the computer running the server component of your application
- The TCP/IP socket port number or service name (alias) to be used for Link for ModelSim connections

For guidelines on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports” on page 1–17.

## Choosing TCP/IP Socket Ports

To use the TCP/IP socket communication, you must choose a TCP/IP socket port number that is available in your computing environment for use by the Link for ModelSim client and server components. The two components use the port number to establish a TCP/IP connection. Port numbers are particularly important for applications that implement multiple clients and servers and use TCP/IP socket communication on a single node. The port numbers uniquely identify each client and server and enable connections only between components sharing the same port number. For remote network configurations, the Internet address helps distinguish multiple connections.

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 Link for ModelSim 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.

---

**Note** The socket port resource is associated with the server component of a Link for ModelSim 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 Simulink in a cosimulation configuration, the socket port is a resource of the system running ModelSim.

---

## Checking Product Requirements

Link for ModelSim requires the following:

<b>Platform</b>	Linux
	Solaris
	Windows 2000
	Windows XP
<b>Application software</b>	ModelSim SE/PE, a Model Technology Inc. product
	MATLAB

<b>Application software required for cosimulation</b>	Simulink Fixed-Point Blockset
<b>Optional application software</b>	Communications Blockset DSP Blockset Filter Design Toolbox Signal Processing Toolbox

## Installing Related Application Software

Based on your configuration decisions and the software required for your Link for ModelSim 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 for ModelSim MATLAB server, 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 ModelSim, see the installation instructions for that product. For information on installing MathWorks products, see the MATLAB installation instructions.

## Installing Link for ModelSim

Based on your configuration decisions, identify systems on which you need to install the Link for ModelSim. Install it on each system running MATLAB that requires a communication channel for ModelSim and MATLAB or Simulink cosimulation.

For details on how to install the Link for ModelSim, see the MATLAB installation instructions.

## Setting Up ModelSim for Use with the Link for ModelSim

Once all the required software is installed, you can choose to set up ModelSim such that it is always ready for use with MATLAB and Simulink. You can complete this setup immediately after installing the software (or later) either interactively or programmatically from scripts.

To set up ModelSim for use with the Link for ModelSim as part of the installation process, use the MATLAB function `setupmodelsim`. You can use the function in interactive mode or command-line mode. The interactive mode displays messages and prompts you for input. The command-line mode is available for script-based setups.

## Running the Setup Program in Interactive Mode

To run the ModelSim setup program in interactive mode,

- 1 Enter the function at the MATLAB command prompt:

```
setupmodelsim
```

Alternatively, you can specify the function with the property name and property value pair 'action', 'install'.

```
setupmodelsim('action', 'install')
```

The installation script asks you to identify the installed version of ModelSim that you want to use with the Link for ModelSim.

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

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

- 2 Specify an installed version of ModelSim. If you want to explicitly specify the path for an installed version of ModelSim, Enter n. The script prompts you to enter an explicit path.

```
Please enter the path to your ModelSim executable
file (modelsim.exe or vsim.exe):
```

If you prefer that setupmodelsim locate and display a list of installed versions, enter y or press the **Enter** key, enter y. The function searches for installed versions of ModelSim and displays output similar to the following:

```
Select a ModelSim installation:
```

```
[2] pathname2\modelsim\win32      ModelSim SE n.nx
[1] pathname1\modelsim\win32      ModelSim SE n.nx
[0] None
```

```
Selected ModelSim installation:
```

- 3 Depending on your response in step 2, enter one of the following:

- The complete pathname for a ModelSim or vsim executable
- One of the listed numeric installation identifiers (0, 1, 2, and so on)

The function modifies the installation files, displays the following message, and exits:

```
ModelSim successfully configured for MATLAB and Simulink
```

If the specified ModelSim installation has already been modified for use with MATLAB and Simulink, the following message appears:

```
Previous MATLAB startup file found in this installation  
of ModelSim:
```

```
d:\applications\modelsim\win32\..  
\tcl\ModelSimTclFunctionsForMATLAB.tcl  
Do you want to replace this file [y]/n?
```

If you choose to overwrite the file, `setupmodelsim` overwrites it and then displays the following message:

```
ModelSim successfully configured for MATLAB and Simulink
```

Otherwise, `setupmodelsim` displays the following message and exits without modifying the file:

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

---

**Note** Although the installation script identifies an installed version of ModelSim for use with the Link for ModelSim, you can override this setting at any time with subsequent interactive or programmatic calls to `setupmodelsim` or `vsim`.

---

## Running the Setup Program in Command-Line Mode

To run the ModelSim setup program in command-line mode, call the `setupmodelsim` function with property name and property value pairs that specify the following:

- Tcl commands that execute during ModelSim startup
- The pathname of a specific version of a ModelSim executable



- Whether the function is to install or uninstall support for MATLAB and Simulink

See the description of `setupmodelsim` for more information.

### **Removing Link for ModelSim Configuration Information from ModelSim**

To remove Link for ModelSim configuration information from ModelSim, issue the `setupmodelsim` function with the property name/property value pair 'action','uninstall'.

```
setupmodelsim ('action', 'uninstall')
```

## Getting Help with the Link for ModelSim

The following sections explain how to get help with using the Link for ModelSim:

- “Documentation Overview” on page 1–24
- “Online Help” on page 1–25
- “Demos and Tutorials” on page 1–25

### 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. Guides you through product demos.
Chapter 2, “MATLAB and ModelSim Tutorial”	Guides you through the process of setting up and running a sample ModelSim and MATLAB test bench session.
Chapter 3, “Simulink and ModelSim Tutorial ”	Guides you through the basic steps for setting up an application of the Link for ModelSim that uses Simulink to verify a simple VHDL inverter model.
Chapter 4, “MATLAB and ModelSim Manchester Receiver Tutorial”	Guides you through the steps for setting up a script that applies the Link for ModelSim, MATLAB, and ModelSim to verify a VHDL Manchester Receiver model with clock recovery capabilities.
Chapter 5, “Coding a Link for ModelSim MATLAB Application”	Explains how to code VHDL models and MATLAB functions for Link for ModelSim MATLAB applications. Provides details on how the Link for ModelSim interface maps VHDL data types to MATLAB data types and vice versa.

Chapter 6, “Starting and Controlling MATLAB Test Bench Sessions”	Explains how to start and control ModelSim and MATLAB test bench sessions.
Chapter 7, “Modeling and Verifying a VHDL Design with Simulink”	Explains how to use ModelSim and Simulink for cosimulation modeling.
Chapter 8, “MATLAB Functions — Alphabetical List”	Describes the Link for ModelSim functions for use with MATLAB
Chapter 9, “ModelSim Commands — Alphabetical List”	Describes the Link for ModelSim commands for use with ModelSim.
Chapter 10, “Simulink Blocks — Alphabetical List”	Describes the Link for ModelSim blocks for use with Simulink.

## Online Help

The following online help is available:

- Online help in the MATLAB Help browser. Click the Link for ModelSim product link in the browser’s Contents.
- M-help for Link for ModelSim MATLAB functions and ModelSim commands. This help is accessible with the MATLAB `doc` and `help` commands. For example, enter the command line `doc setupmodelsim`.
- Block reference pages accessible through the Simulink interface.

## Demos and Tutorials

The Link for ModelSim 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.

The following topics help you run two of the demos available as part of the product. The first shows how ModelSim works with MATLAB and the second shows how ModelSim works with Simulink:

- “Running the ModelSim and MATLAB Random Number Generator Demo” on page 1–27
- “Running the Simulink and ModelSim Manchester Receiver Demo” on page 1–32

Tutorials provide procedural instruction on how to apply the product. Some focus on features while others focus on application scenarios. The following topics guide you through three tutorials. The first two tutorials listed have a feature focus and each addresses use of ModelSim with either MATLAB or Simulink. The third tutorial has more of an application focus and shows you how you might automate the cosimulation setup and processing.

- Chapter 2, “MATLAB and ModelSim Tutorial”
- Chapter 3, “Simulink and ModelSim Tutorial ”
- Chapter 4, “MATLAB and ModelSim Manchester Receiver Tutorial”

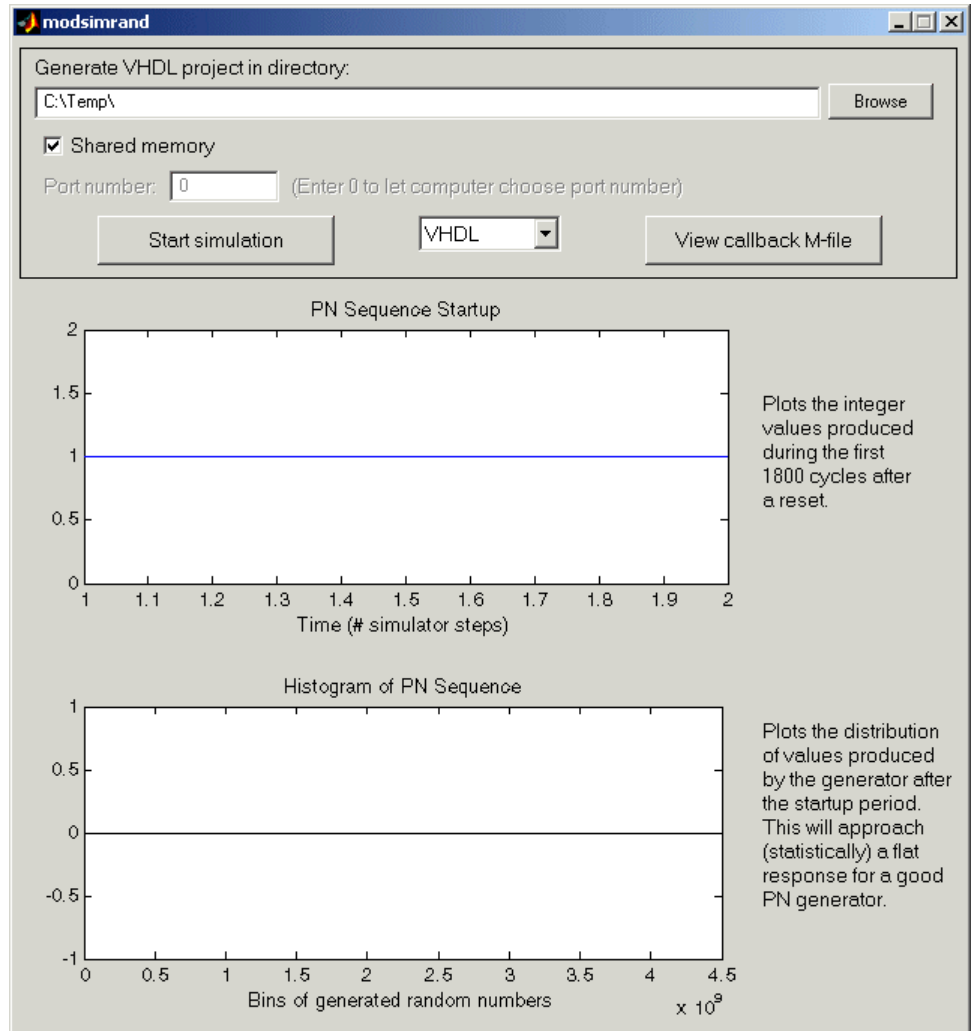
## Running the ModelSim and MATLAB Random Number Generator Demo

Link for ModelSim includes a demo that provides a high-level view of how MATLAB and ModelSim work together. To run the demo, you need to be running MATLAB and ModelSim and Link for ModelSim must be installed. Run the demonstration by entering commands and graphical user interface (GUI) data as explained in the following procedure:

- 1 Invoke MATLAB and make it your active window.
- 2 Set up and change to a writable working directory that is outside the context of your MATAB installation directory.
- 3 Enter the function name `modsimrand` at the prompt in the MATLAB Command Window.

```
modsimrand
```

The function displays the following dialog box.



- 4 Specify the location into which the demo is to place the ModelSim project files that it generates. This location must be writable. You can type a path in the **Generate VHDL project in directory** text field or you can click **Browse** to find an appropriate directory.

A temporary path is created by default.

- 5 Specify a communication mode for the link between ModelSim and MATLAB. By default, the demo uses a shared memory channel for communication. If you prefer to use TCP/IP socket communication, clear the **Shared memory** check box and enter a socket port number in the **Port number** text field. If you specify 0, the operating system running on the computer chooses a port number that is valid and available on your system for you. For information on choosing TCP/IP port numbers, see “Choosing TCP/IP Socket Ports” on page 1–17.

---

**Note** You must specify a socket port number. The demo does not support socket service names.

---

- 6 Choose the version of the HDL model you want to simulate — VHDL or Verilog. If you choose Verilog, the demo applies the Link for ModelSim wrapverilog function to a Verilog version of the model.
- 7 Click **View callback M-file**. MATLAB displays the M-code for the function that executes in the MATLAB environment on behalf of HDL model. Browse through the code to get a sense of what the M-file does. A key task for implementing a MATLAB Link for ModelSim application is to program a MATLAB test bench function such that it can communicate with an HDL model under simulation in ModelSim. When you are done browsing, close the editor window.
- 8 Click **Start simulation**.

The program

- a Starts the MATLAB server. Messages similar to the following appear in the MATLAB Command Window.

```
To enable access from ModelSim, HDLDaemon is used with
appropriate link settings
```

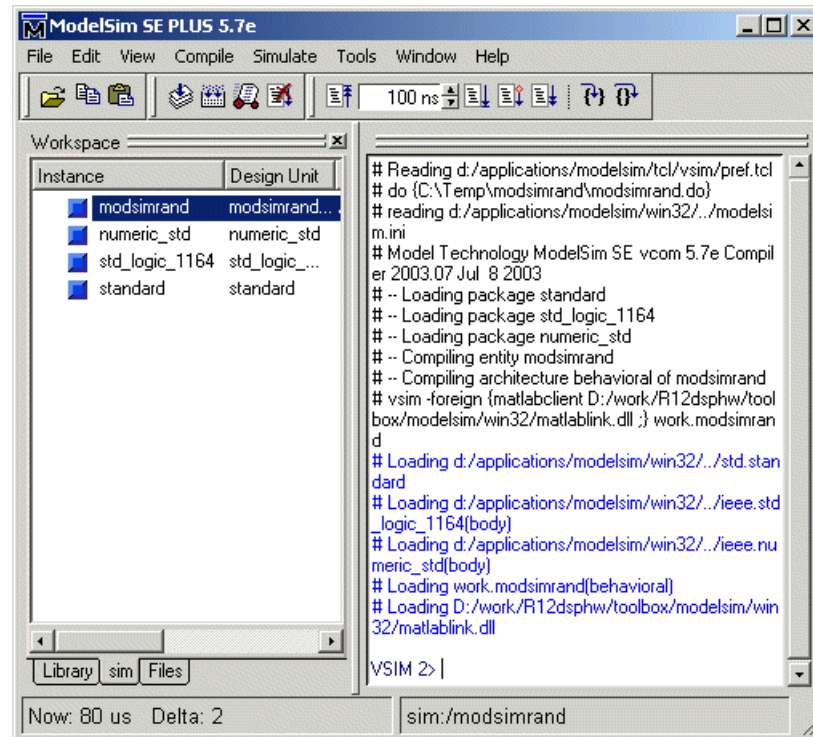
```
The following messages are produced by HDLDaemon to indicate
link status ...
```

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

- b Creates the subfolder modsimrand in the specified VHDL project directory.
- c Generates the macro file modsimrand.do.

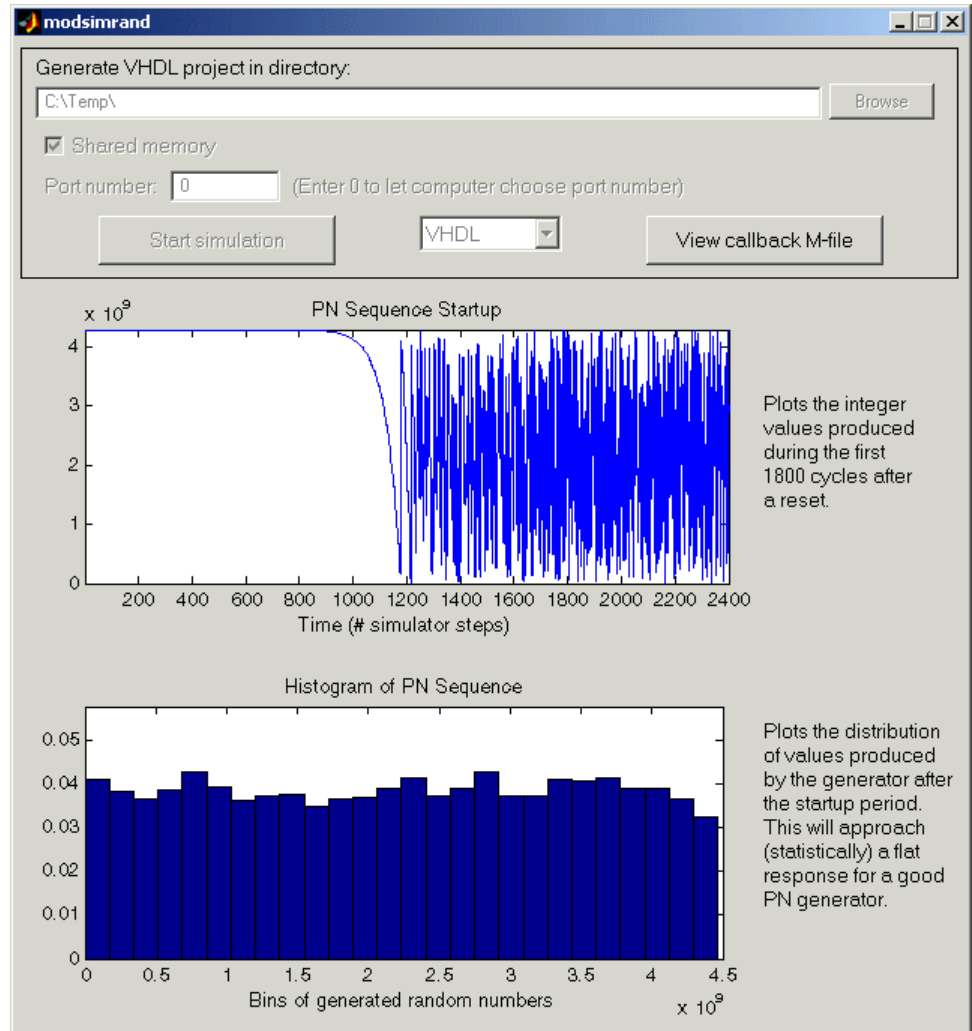
- d Adds the macro file to the modsimrand folder.
- e Wraps the Verilog code if you selected Verilog.
- f Creates a project.
- g Compiles the project entities and architectures.
- h Loads the modsimrand entity for simulation.
- i Starts a simulation.

As the DO macro completes this processing, it displays informational messages in the command line pane of the ModelSim main window, as shown below.



Also note the changes that occur in the **modsimrand** window plots.





- 9 End the simulation in ModelSim by entering the quit command at the VSIM n> prompt.
- 10 Shutdown the MATLAB server, by calling `hdldaemon` with the 'kill' option as follows:

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

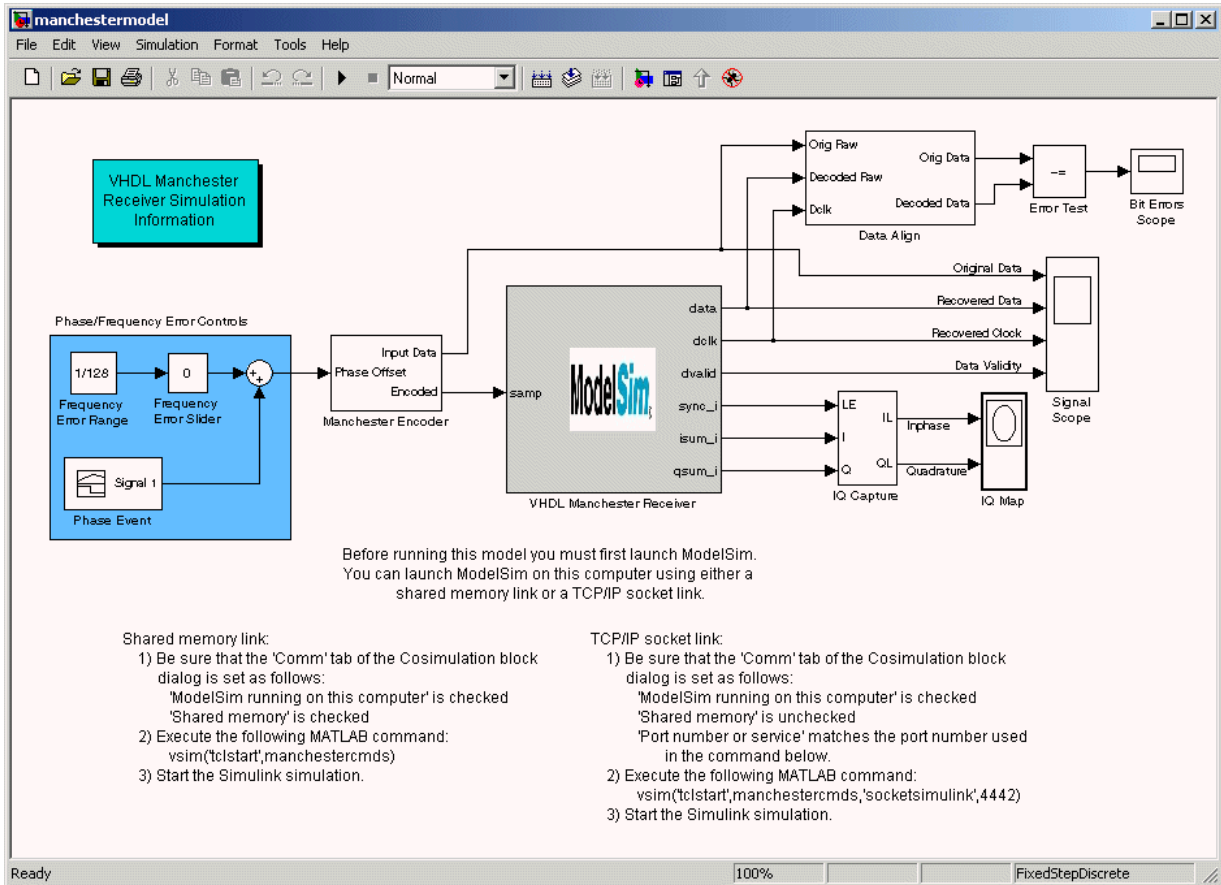
## Running the Simulink and ModelSim Manchester Receiver Demo

The Link for ModelSim includes a demo that provides a high-level view of how Simulink and ModelSim work together. To run the demo, you need to be running MATLAB and the following software must be installed:

- ModelSim
- Simulink
- Link for ModelSim
- Fixed-Point Blockset

Run the demo by entering commands and GUI data as explained in the following procedure:

- 1 Invoke MATLAB and make it your active window.
- 2 Open the Simulink model `manchestermode1`. The following Simulink model window appears.



**Note** If you have the Communications Blockset installed, you have the option of running the demo `manchestermodelcomblks.mdl`.

The VHDL Manchester Receiver block represents a Manchester Receiver design that is coded in VHDL and will be cosimulated in the ModelSim environment.

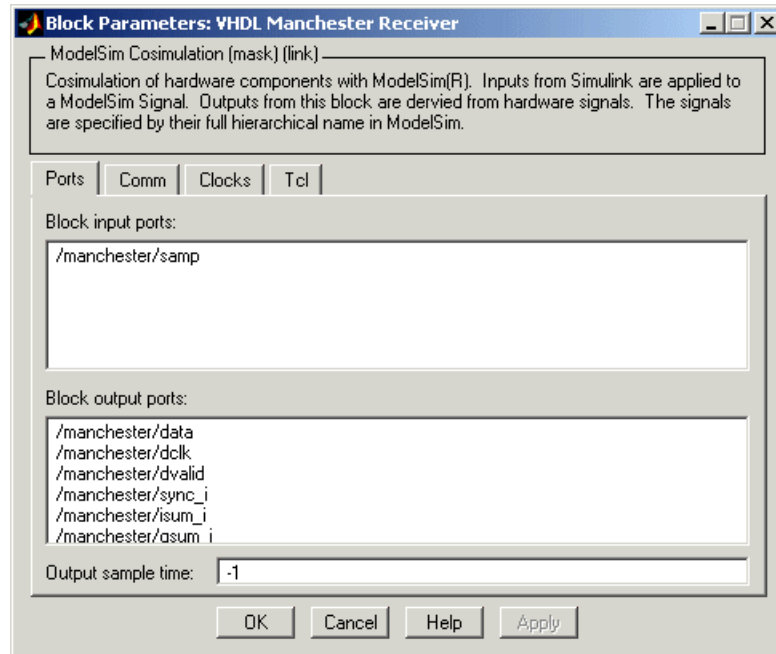
- 3 Save a writable version of the model to a directory outside the context of your MATLAB installation directory.

- 4 Decide on a mode of communication and, if necessary, set the link communication parameters appropriately for your system.

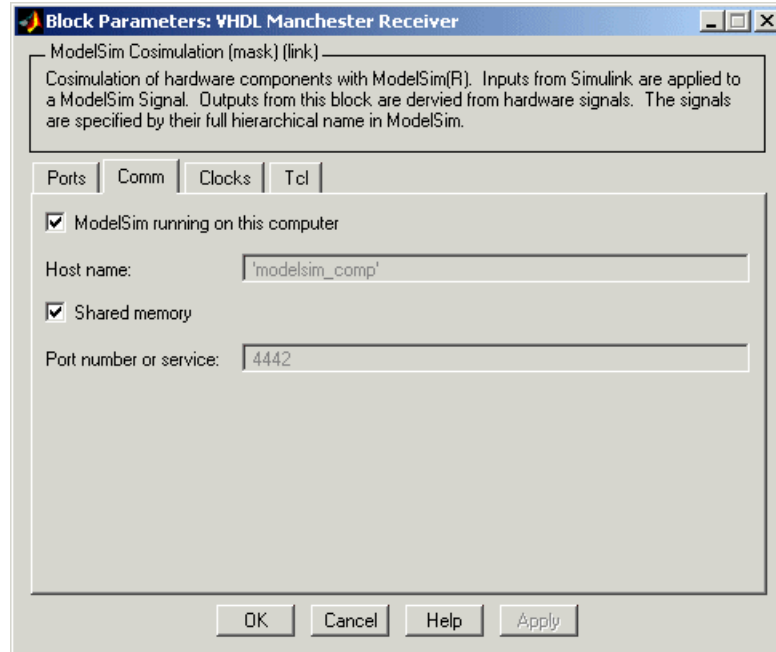
If you prefer to use shared memory, skip to step 5.

To use TCP/IP socket communication, do the following:

- a Double-click the VHDL Manchester Receiver block. The **Block Parameters VHDL Manchester Receiver** dialog appears.



- b Click the **Comm** tab. The dialog displays communication configuration information.



- c Clear the **Shared memory** check box.
  - d If necessary, change the value in the **Port number or service** text box to a valid port number or service name for your system.
  - e Click **Apply** and then **OK**.
- 5 Set up ModelSim for use with Simulink.
- a Select and copy one of the following command lines from the instructions that appear at the bottom of the model window:

Shared memory link

```
vsim('tclstart',manchestercmds)
```

TCP/IP socket link

```
vsim('tclstart',manchestercmds,'socketsimulink',4442)
```

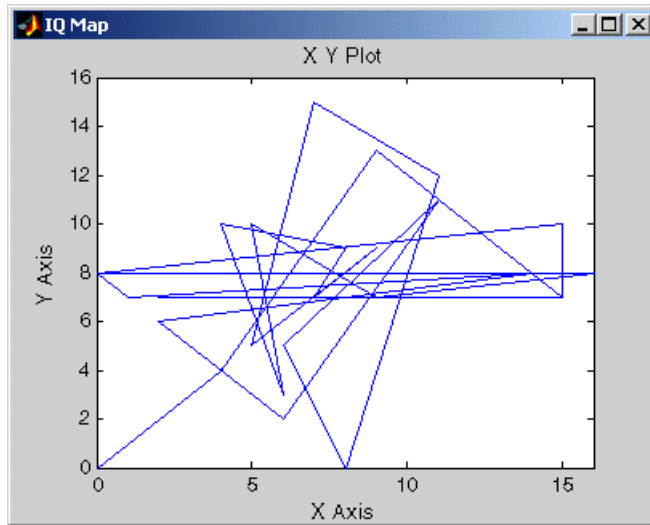
The `vsim` function launches ModelSim for use with the Link for ModelSim. The property name and property value pairs in the command lines specify the following information.

Property Name and Property Value Pair...	Specifies...
'tclstart', manchestercmds,	Tcl commands that execute after ModelSim starts running
'socketsimulink', 4442	TCP/IP socket communication for the link between Simulink and ModelSim, using socket port number 4442

- b** Paste the command line in the MATLAB Command Window.
  - c** If you modified the socket port specification in step 5, replace port number 4442 with the appropriate port number or service name for your system. The socket port that you specify in this command line *must* match the socket port value specified in the **Block Parameters** dialog. If they do not match, ModelSim starts but is not able to establish a communication link with Simulink. If you attempt to run the Simulation, Simulink reports a message indicating that the socket is not connected.
  - d** Press **Enter**. ModelSim starts and processes the Tcl commands specified in the M-file `manchestercmds`. The Tcl commands
    - Create a design library, if one does not already exist
    - Load required packages and compile each of the three VHDL entities included in the VHDL Manchester receiver model.
    - Load an instance of each of the three entities for simulation.
    - Establish a communication link with Simulink.Scroll through the messages displayed in the ModelSim command window for more detail.

To view the Tcl commands, edit the M-file `manchestercmds.m`.
- 6** From the Simulink model window, start the simulation by clicking the model's start button. The cosimulation session runs. The I/Q Map block of the Simulink model opens a figure window and plots a map of the signal

values for inphase and quadrature waveforms. The figure window will look similar to the following:







# MATLAB and ModelSim Tutorial

---

This chapter guides you through the basic steps for setting up an application of Link for ModelSim that uses MATLAB to verify a simple VHDL model of a pseudo random number generator based on the Fibonacci sequence.

---

**Note** To complete the tutorial, MATLAB, ModelSim, and the Link for ModelSim must be installed.

---

“Setting Up Tutorial Files” (p. 2–3)	Explains how to set up folders and files for the tutorial.
“Starting the MATLAB Server” (p. 2–4)	Explains how to start the MATLAB server.
“Setting Up ModelSim ” (p. 2–6)	Explains the basic steps for setting up a ModelSim project.
“Developing the VHDL Code” (p. 2–8)	Introduces Link for ModelSim VHDL coding requirements.
“Compiling the VHDL File” (p. 2–11)	Explains how to compile a sample VHDL file for use with Link for ModelSim.
“Loading the Simulation” (p. 2–12)	Explains how to load the sample simulation.
“Developing the MATLAB Function” (p. 2–15)	Introduces Link for ModelSim MATLAB function coding requirements.

“Running the Simulation” (p. 2–18)	Explains how to start and monitor the sample simulation.
“Shutting Down the Simulation” (p. 2–21)	Explains how to shut down a Link for ModelSim test bench session in an orderly way.

## 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\modelsim demos\modsimrand_plot.m
```

```
MATLABROOT\toolbox\modelsim\modelsim demos\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. These instructions assume you are familiar with the MATLAB user interface:

- 1 Start MATLAB.
- 2 Set your MATLAB current directory to the directory you created in “Setting Up Tutorial Files” on page 2–3.
- 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

```
HDLDaemon is NOT running
```

If the server is running, the message reads

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

- 4 Start or restart the server by calling `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

- 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 “Starting the MATLAB Server” on page 2–4 or 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

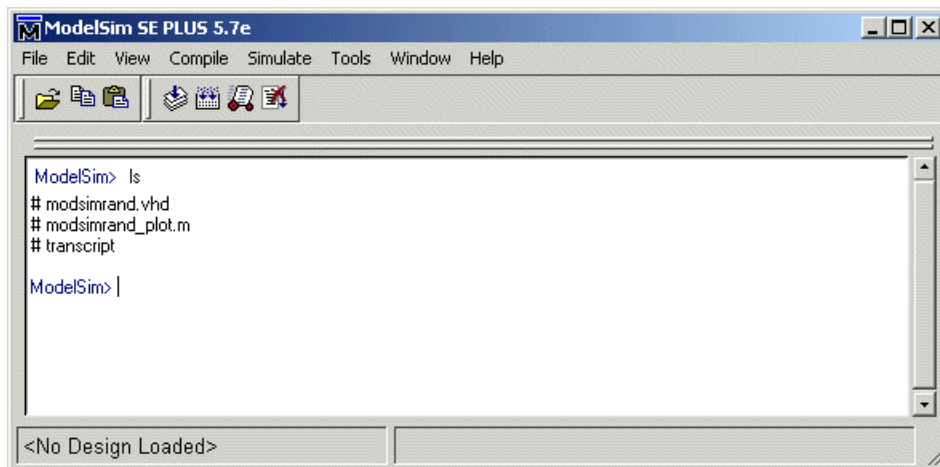
This section describes the basic procedure for starting ModelSim and setting up a ModelSim project. These instructions assume you are familiar with the ModelSim user interface:

- 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 Link for ModelSim. The initial 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.

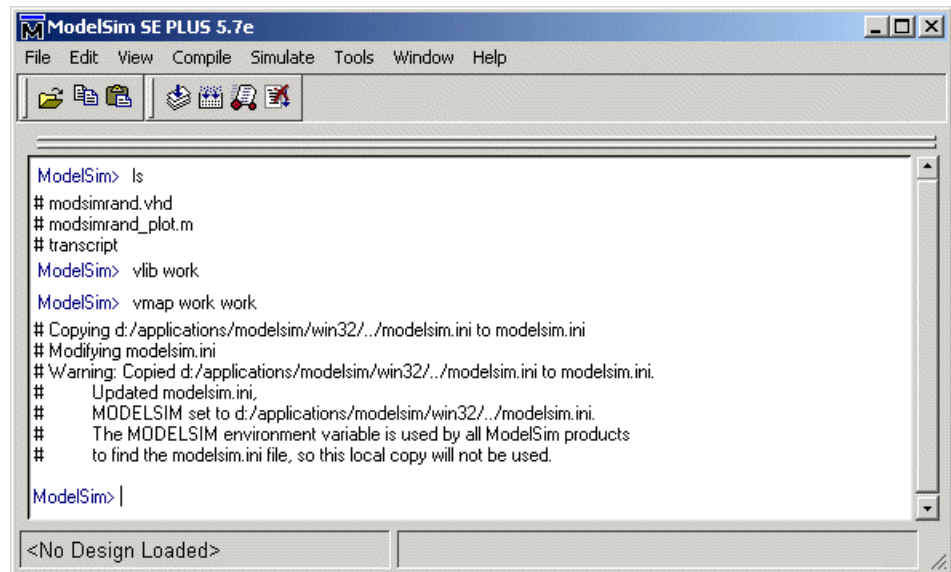


The command should list the files `modsimrand.vhd`, `modsimrand_plot.m`, and `transcript`.

- 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
```



The screenshot shows the ModelSim SE PLUS 5.7e interface. The title bar reads "ModelSim SE PLUS 5.7e". The menu bar includes "File", "Edit", "View", "Compile", "Simulate", "Tools", "Window", and "Help". Below the menu bar is a toolbar with icons for file operations and simulation. The main text area contains the following text:

```
ModelSim> ls
# modsimrand.vhd
# modsimrand_plot.m
# transcript
ModelSim> vlib work
ModelSim> vmap work work
# Copying d:/applications/modsim/win32/./modelsim.ini to modelsim.ini
# Modifying modelsim.ini
# Warning: Copied d:/applications/modsim/win32/./modelsim.ini to modelsim.ini.
# Updated modelsim.ini,
# MODELSIM set to d:/applications/modsim/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> |
```

At the bottom of the window, there is a status bar that says "<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.

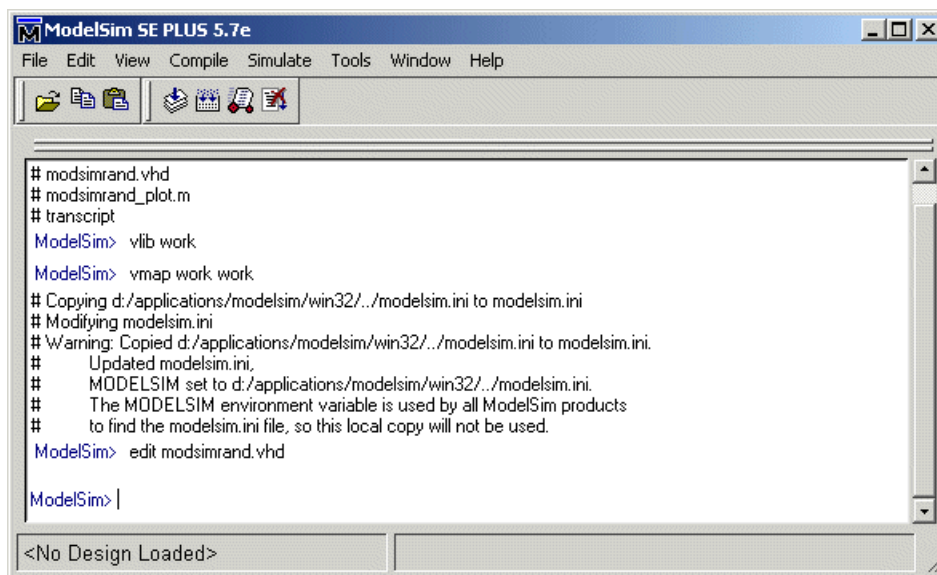
---

## Developing the VHDL Code

After setting up a design library, typically you would use the ModelSim Editor to create and modify your VHDL 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
```



The screenshot shows the ModelSim SE PLUS 5.7e editor window. The title bar reads "ModelSim SE PLUS 5.7e". The menu bar includes "File", "Edit", "View", "Compile", "Simulate", "Tools", "Window", and "Help". Below the menu bar is a toolbar with icons for file operations and simulation. The main text area displays the following text:

```
# modsimrand.vhd
# modsimrand_plot.m
# transcript
ModelSim> vlib work
ModelSim> vmap work work
# Copying d:/applications/modsim/win32/./modelsim.ini to modelsim.ini
# Modifying modelsim.ini
# Warning: Copied d:/applications/modsim/win32/./modelsim.ini to modelsim.ini
# Updated modelsim.ini,
# MODELSIM set to d:/applications/modsim/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> edit modsimrand.vhd

ModelSim> |
```

At the bottom of the window, there is a status bar that says "<No Design Loaded>".

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



```

1 -----
2 -- Psuedo Random Word Generator
3 -- Demonstration of 'Link for ModelSim'
4 --
5 --
6 --
7 -- Modelsim
8 -- >vsim work.modsimrand -foreign "matlabclient matlablink.so;"
9 -- >matlabtb modsimrand -mfunc modsimrand_plot -rising /modsimrand/clock -socket 4448
10 -- >force /modsimrand/clock 0 0,1 5 ns -repeat 10 ns
11 -- >force /modsimrand/clock_en 1
12 -- >force /modsimrand/reset 1 0,0 50 ns
13

```

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 Link for

ModelSim interface. For a list of the supported types, see “Coding VHDL Entities for MATLAB Verification” on page 5–3.

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 VHDL Entities for MATLAB Verification” on page 5–3.

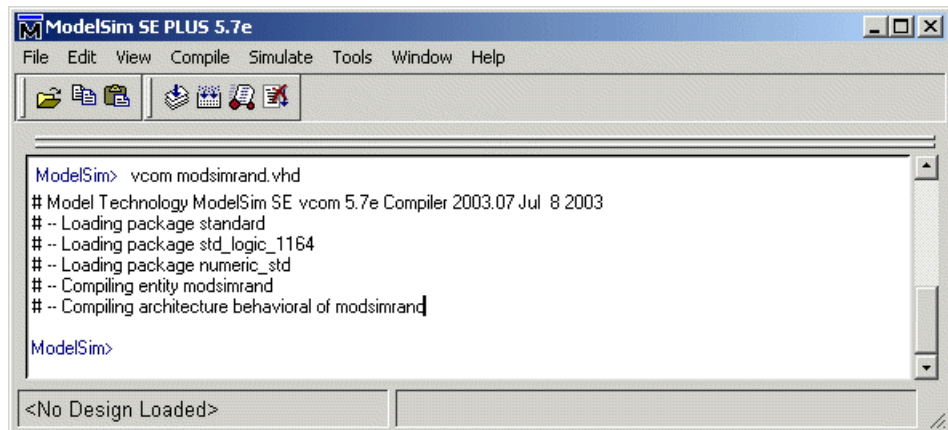
- 3 Browse through the rest of `modsimrand.vhd`. The remaining code defines a behavioral architecture for `modsimrand` that writes a randomly generated Fibonacci sequence to an output register when the clock experiences a rising edge.
- 4 Close the ModelSim **edit** window.

## 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 filename 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.



```
ModelSim SE PLUS 5.7e
File Edit View Compile Simulate Tools Window Help
ModelSim> vcom modsimrand.vhd
# Model Technology ModelSim SE vcom 5.7e Compiler 2003.07 Jul 8 2003
# -- Loading package standard
# -- Loading package std_logic_1164
# -- Loading package numeric_std
# -- Compiling entity modsimrand
# -- Compiling architecture behavioral of modsimrand
ModelSim>
<No Design Loaded>
```

## Loading the Simulation

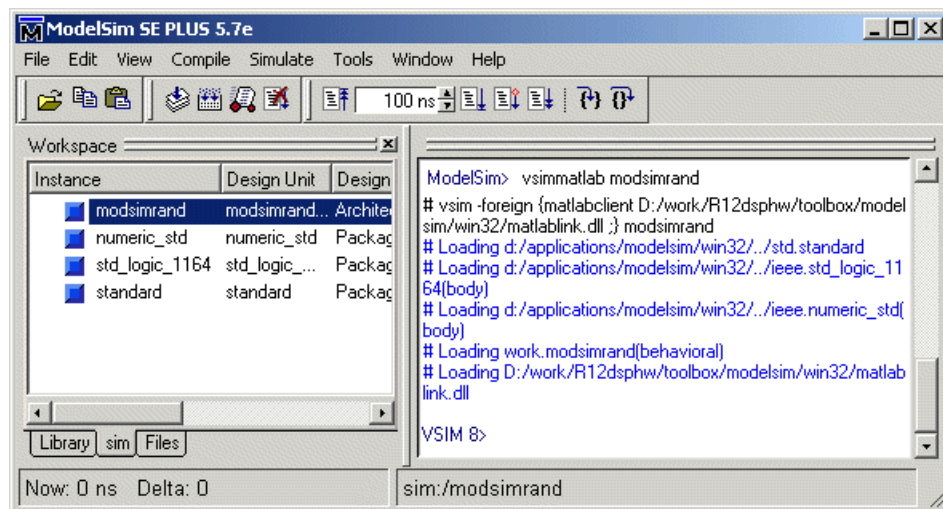
Once you successfully compile the VHDL source file, you are ready to load the model for simulation. This section explains how to load an instance of entity `modsimrand` for simulation:

- 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
```

---

**Note** 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. If you need to verify the port number, issue a call to the `hdldaemon` function with 'status' as follows:

```
hdldaemon('status')
HDLDaemon socket server is running on port 4795 with 0 connections
```

This function call indicates that the server is using TCP/IP socket communication with socket port 4795 and is running with no connections. If a shared memory link is in use, the message will reflect that mode of communication.

---

Arguments in the command line specify the following:

<code>modsimrand</code>	The instance of the VHDL entity that is to be attached to a MATLAB function.
<code>-mfunc modsimrand_plot</code>	The MATLAB function to be called on behalf of entity <code>modsimrand</code> .
<code>-rising /modsimrand/clock</code>	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 pathname format. If you do not specify a full pathname, the command applies ModelSim rules to resolve signal specifications.
<code>-socket <i>portnum</i></code>	The TCP/IP socket port <i>portnum</i> 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 Link for ModelSim invokes a MATLAB function that has the same name as the specified entity 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 a number of mechanisms, including ModelSim force commands and an `iport` parameter (see “Developing the MATLAB Function” on page 2-15). 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

Link for ModelSim verifies VHDL hardware in MATLAB as a function. Typically, at this point you would create or edit a MATLAB function that meets the Link for ModelSim requirements. For this tutorial, open and examine the existing file `modsimrand_plot.m`. This section highlights areas of code in `modsimrand_plot.m` that are required for MATLAB to verify a VHDL model:

- 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 in that it represents the communication channel between MATLAB and ModelSim. When coding the function definition, consider the following:

- By default, Link for ModelSim 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`. The following table briefly describes the purpose of each parameter:

<code>iport</code>	Structure that specifies IN ports to be forced.
<code>tnext</code>	Specifies an optional future time at which the MATLAB function is called back.
<code>oport</code>	Structure that receives signal values from the OUT ports defined for the corresponding VHDL entity at the time specified by <code>tnow</code> .

<code>tnow</code>	Receives the simulation time at which the MATLAB function is called.
<code>portinfo</code>	For the first invocation of the function only, receives an array of information that describes the ports defined for the corresponding VHDL entity.

For more information on the required MATLAB function parameters, see “Setting up Expected Parameters” on page 5–13.

- 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 Make note of the data types of ports defined for the entity under simulation. The Link for ModelSim interface 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.

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

For more information on interface data type conversions, see “Data Type Conversions” on page 5–9.

- 4 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. “Converting Data for Manipulation” on page



5–17 provides a summary of the types of data conversions to consider when coding your own MATLAB functions.

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

## Running the Simulation

This section explains how to start and monitor a simulation:

- 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 Set ModelSim to be your active window and enter the following run command:

```
VSIM n> run 80000
```

This command advances the simulation 80000 time steps. If you are using default settings for the simulation time step, ModelSim runs the simulation for 80,000 nanoseconds.

- 4 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 `matlab` or `matlabtb` command. Thus, after restarting ModelSim, you must reissue the previous command or issue a new command.

---

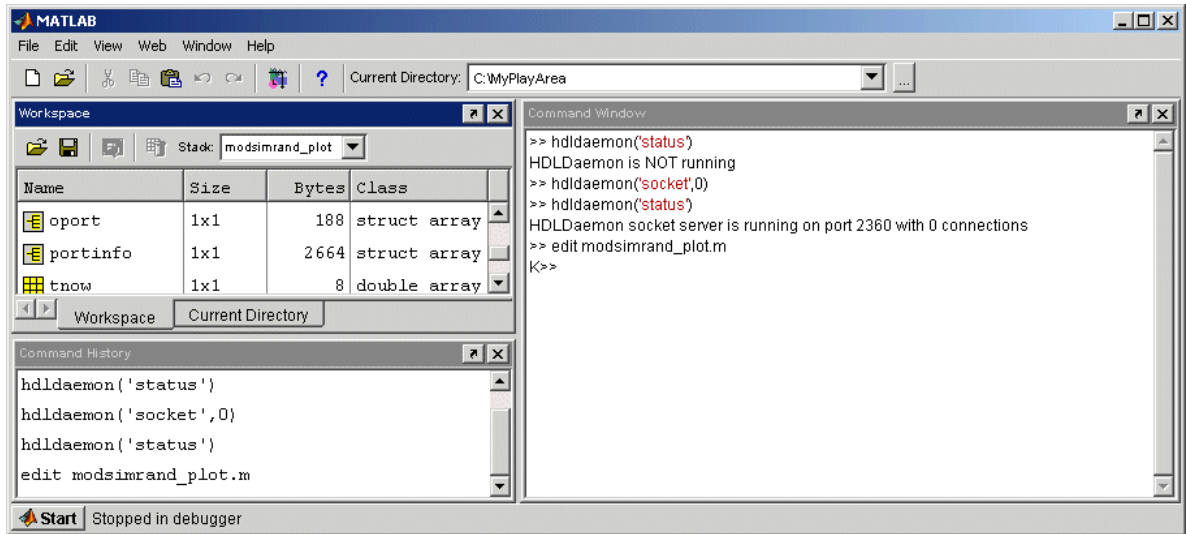
- 5 Reissue the `matlabtb` command.

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

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

```
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  
Vsim n> run 80000
```

The simulation runs in MATLAB until it reaches the breakpoint that you just set in `modsimrand_plot.m`. 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.



- 9 Examine `oport`, `portinfo`, and `tnow`.
- 10 Click **Debug**→**Continue** in the MATLAB Edit/Debug window. Note that `portinfo` disappears after the first function invocation. Also note that the value of `tnow` changes from 0 to 5e-009 .
- 11 Clear the breakpoint by clicking the red breakpoint marker.
- 12 Unblock ModelSim and continue the simulation by clicking **Debug**→**Continue** in the MATLAB Edit/Debug window.  
The simulation runs to completion.

## Shutting Down the Simulation

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

In ModelSim,

- 1 Stop the simulation on the client side by selecting **Simulate->End Simulation** or entering the quit command.
- 2 Close the modsimrand project by selecting **File->Close->Project**. A warning dialog appears. Click **OK**.
- 3 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
```



# Simulink and ModelSim Tutorial

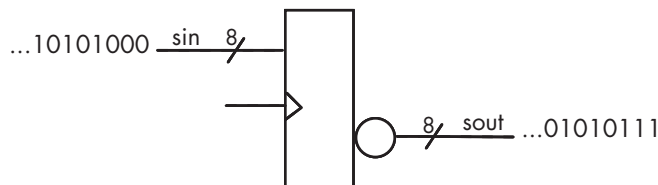
---

This chapter guides you through the basic steps for setting up a Link for ModelSim application that uses Simulink to verify a simple VHDL model that inverts bits.

- |   |   |
|---|---|
| “Developing the VHDL Code” (p. 3–2)   | Guides you through editing VHDL code for a simple inverter model with the ModelSim VHDL editor.               |
| “Compiling the VHDL File” (p. 3–4)  | Explains how to compile the VHDL code.  |
| “Creating the Simulink Model” (p. 3–6)  | Guides you through the process of creating a simple Simulink model that includes the VHDL inverter model.     |
| “Setting Up ModelSim for Use with Simulink” (p. 3–12)                           | Explains how to start ModelSim from MATLAB and configure it for use with Simulink.                            |
| “Loading Instances of the VHDL Entity for Cosimulation with Simulink” (p. 3–13) | Explains how to load an instance of the VHDL inverter model for cosimulation with Simulink.                   |
| “Running the Simulation” (p. 3–14)  | Guides you through a scenario of running and monitoring Simulink and Link for ModelSim of the Simulink model. |
| “Shutting Down the Simulation” (p. 3–17)  | Explains how to shut down a cosimulation in an orderly way.   |

## 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. This is the scenario introduced in this tutorial. 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:

- 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:

```
-----
-- Simulink and ModelSim Inverter Tutorial
--
-- Copyright 2003 The MathWorks, Inc.
-- $Date: 2003/11/13 22:18:11 $
-----
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY inverter IS PORT (
```



```
    sin : IN  std_logic_vector(7 DOWNTO 0);
    sout: OUT std_logic_vector(7 DOWNTO 0);
    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
            sout <= NOT sin;
        END IF;
    END PROCESS;
END behavioral;
```

**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`:

- 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".
```

---

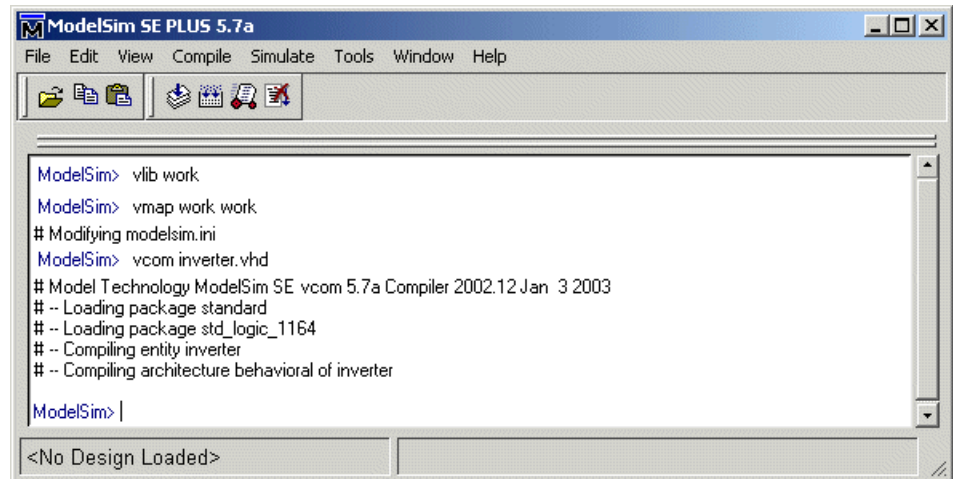
**Note** You must use the ModelSim **File** menu or `vlib` command to create the library directory to ensure that the required `_info` file is created. Do not create the library with operating system commands.

---

- 3 Compile the VHDL file. One way of compiling the file is to click the filename 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.






```
ModelSim SE PLUS 5.7a
File Edit View Compile Simulate Tools Window Help
ModelSim> vlib work
ModelSim> vmap work work
# Modifying modelsim.ini
ModelSim> vcom inverter.vhd
# Model Technology ModelSim SE vcom 5.7a Compiler 2002.12 Jan 3 2003
# -- Loading package standard
# -- Loading package std_logic_1164
# -- Compiling entity inverter
# -- Compiling architecture behavioral of inverter
ModelSim> |
<No Design Loaded>
```

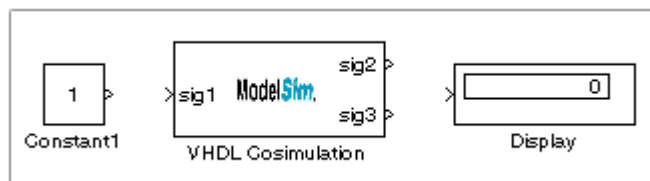
## Creating the Simulink Model

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 3–2 and displays the inverted output:

- 1 Start MATLAB, if it is not already running.
- 2 Open the Simulink Library Browser.
- 3 Open a new model window.
- 4 Drag the following blocks from the Simulink Library Browser to your model window.

- Constant block  in Simulink Source library
- VHDL Cosimulation block  in Link for ModelSim library
- Display block  in Simulink Sink library

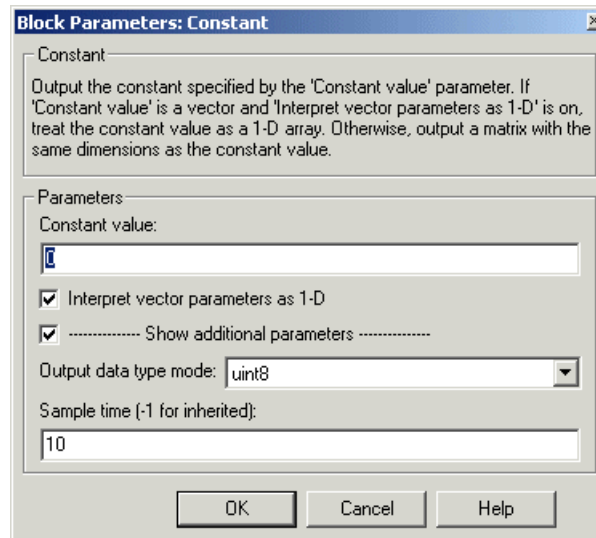
Arrange the three blocks as shown below:



- 5 Configure the Constant block, which is the model's input source.
  - a Double-click the Constant block icon. The **Block Parameters Constant** dialog appears.
  - b Type 0 in the **Constant value** text box. This is just an initial value. Later you can change it to any unsigned integer in the range 0 to 255.

- c Click the **Show additional parameters** check box. The dialog expands and displays additional options.
- d Select uint8 from the **Output data type mode** menu. This type specification is supported by the Link for ModelSim 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)`.
- e Type 10 in the **Sample time** text box. Later you can change it to see the affect various sample times have on the simulation.

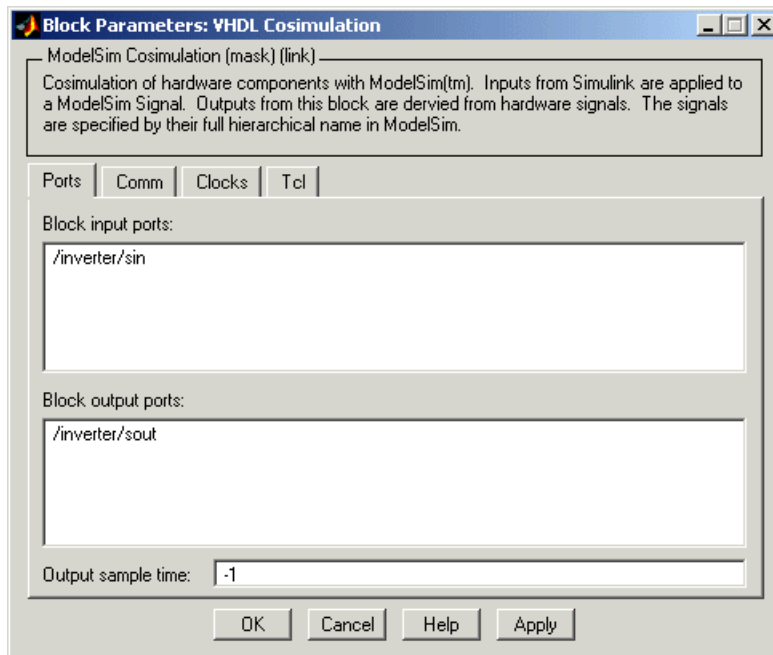
The dialog should now appear as follows.



- f Click **OK**. The **Block Parameters Constant** dialog disappears and the value in the Constant block icon changes to 0.
- 6 Configure the VHDL Cosimulation block, which represents the inverter model written in VHDL.
- a Double-click the VHDL Cosimulation block icon. The **Block Parameters Constant** dialog appears.
  - b In the **Block input ports** text box, replace the sample signal pathname `/top/sig1` with `/inverter/sin`.

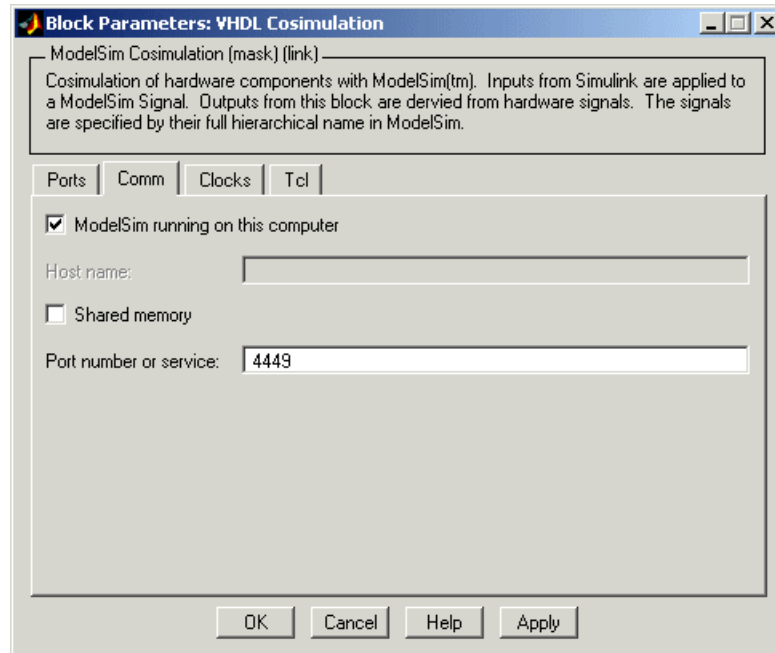
- c In the **Block output ports** text box, replace the sample signal pathnames with `/inverter/sout`.

The **Ports** pane should appear as follows.



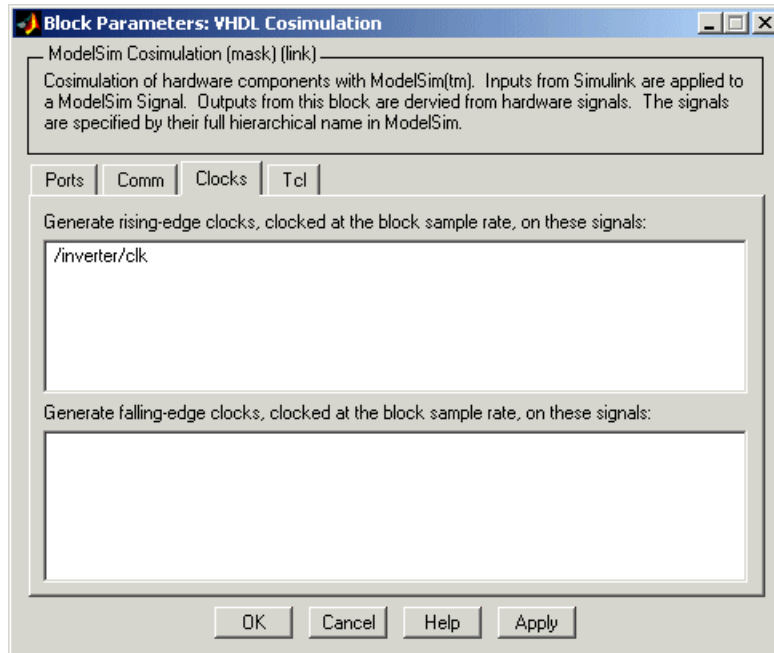
- d Click **Apply**.
- e Click the **Comm** tab.
- f Clear the Shared memory check box.
- g In the **Port number or service** text box, enter socket port number 4449 or, if this port is not available on your system, another valid port number or service name. The model will use TCP/IP socket communication 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 **Comm** pane should appear as follows:



- h** Click **Apply**.
- i** Click the **Clocks** tab.
- j** In the **Generate rising-edge clocks** text box, add the signal path `/inverter/clock`.

The **Clocks** pane should appear as follows:



**k** Click **Apply**.

**l** Click the **Tcl** tab.

**m** In the **Before simulation command** text box, enter the following Tcl command:

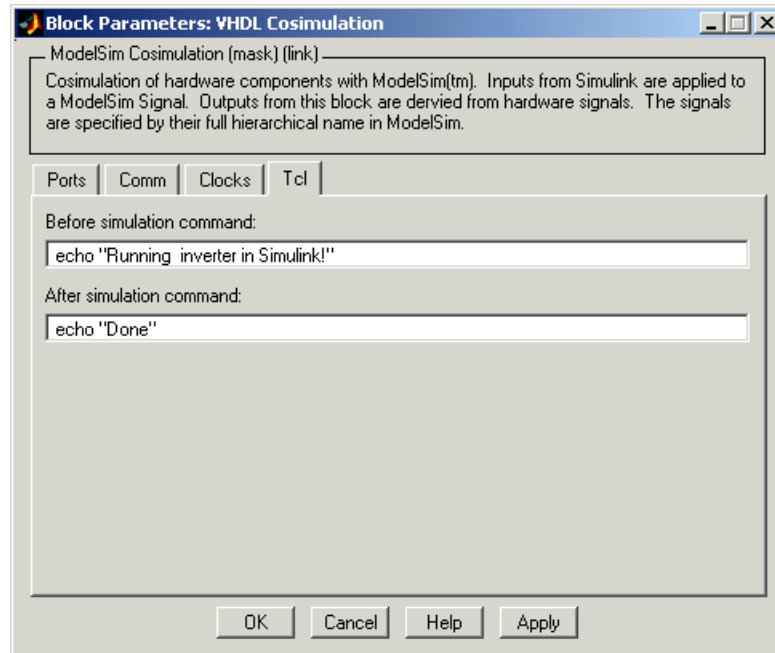
```
echo "Running inverter in Simulink!"
```

**n** In the **After simulation command** text box, enter

```
echo "Done"
```

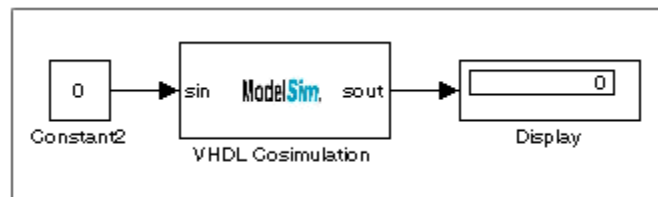
The Tcl dialog panel should appear as follows:





o Click **Apply** and then **OK**.

7 Connect the blocks such that the model appears as follows:



At this point, you might also want to consider adjusting block annotations.

8 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 VHDL 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 a Link for ModelSim variant of the ModelSim `vsim` command. It is made available as part of the ModelSim configuration.

To load an instance of the `inverter` entity,

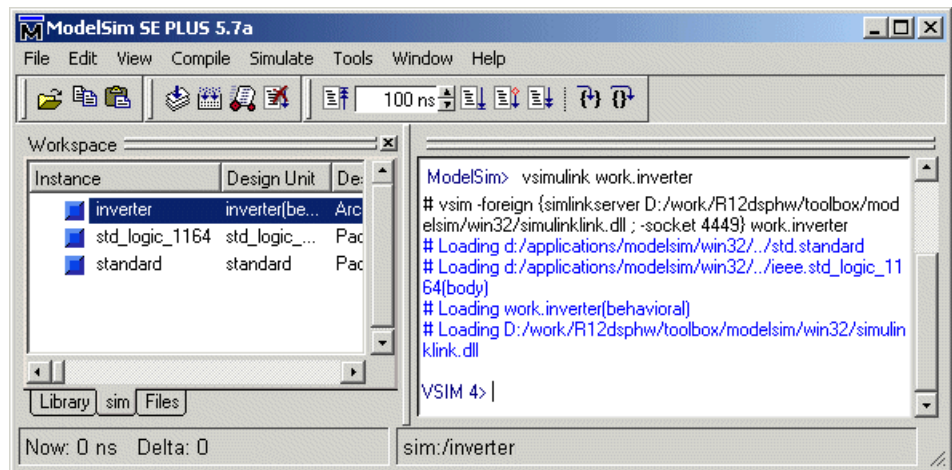
- 1 Change your input focus to the ModelSim window.
- 2 If necessary, change your directory to the location of your `inverter.vhd` file. For example:

```
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.



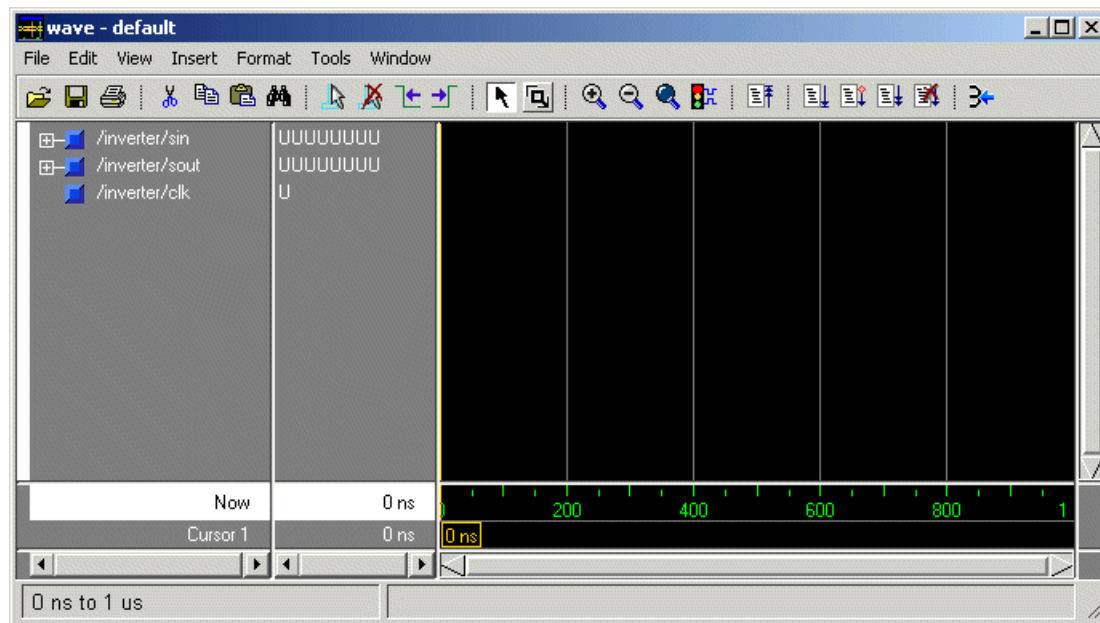
## Running the Simulation

This section guides you through a scenario of running and monitoring a cosimulation session.

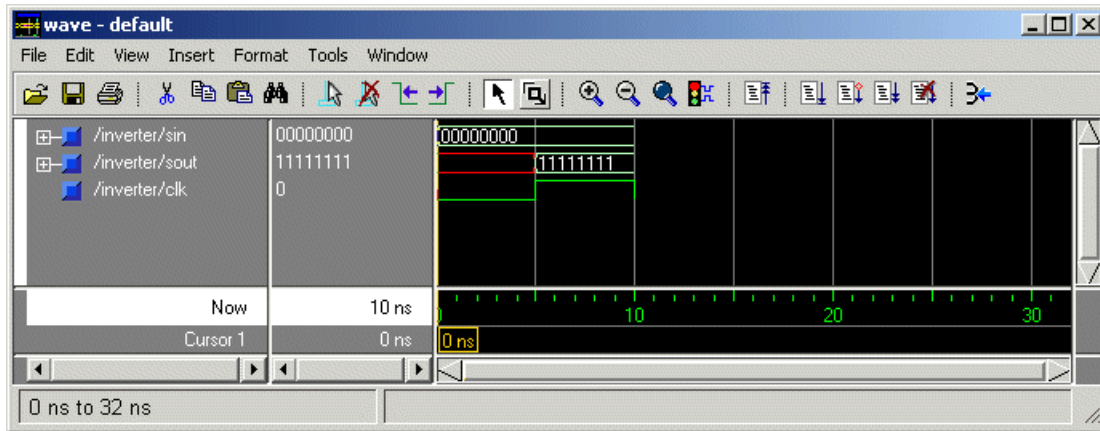
- 1 Open and add the inverter signals to a wave window by entering the following ModelSim command:

```
VSIM nm> add wave /inverter/*
```

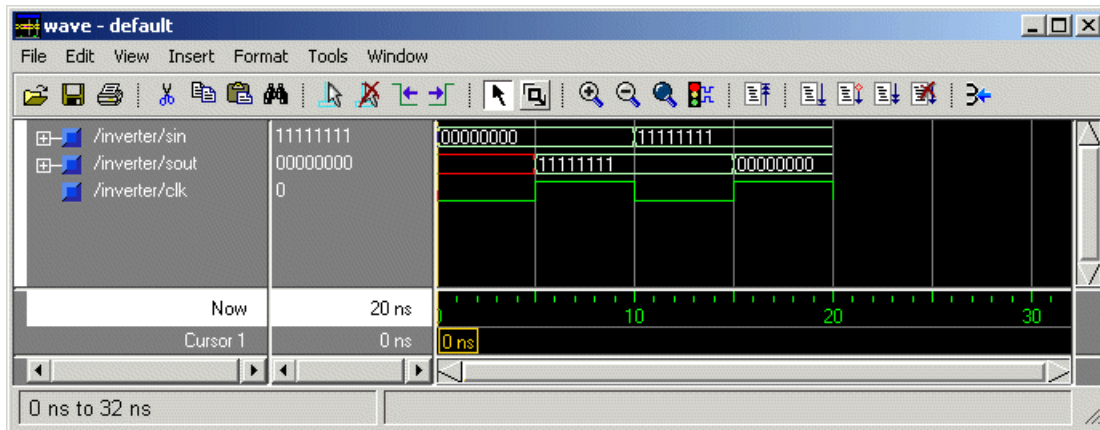
The following wave window appears.



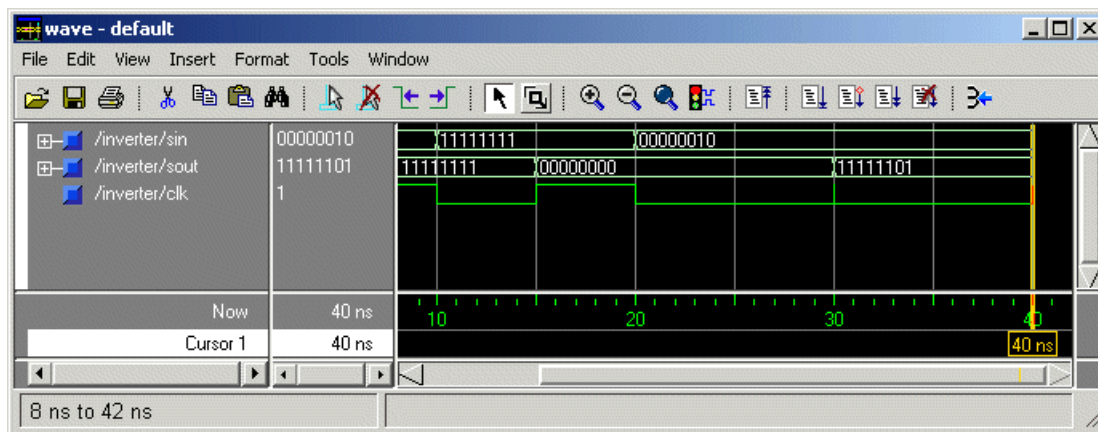
- 2 Change your input focus to your Simulink model window.
- 3 Start a Simulink simulation. The value in the Display block changes to 255. Also note the changes that occur in the ModelSim wave window. You might need to zoom in to get a better view of the signal data.



- 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 follows.



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:

- 1 In ModelSim, stop the simulation by selecting **Simulate->End Simulation**.
- 2 Quit ModelSim.
- 3 Close the Simulink model window.





# MATLAB and ModelSim Manchester Receiver Tutorial

---

This chapter guides you through the steps for setting up an M-file that runs as a test script that applies the Link for ModelSim, MATLAB, and ModelSim to verify a VHDL Manchester Receiver model with clock recovery capabilities.

---

**Note** To complete the tutorial, MATLAB, ModelSim, and Link for ModelSim must be installed.

---

“Background on Manchester Encoding” (p. 4–3)

Introduces you to Manchester encoding, the subject of this tutorial.

“Setting Up Tutorial Files” (p. 4–8)

Explains how to set up files for this tutorial.

“Developing the Manchester Receiver VHDL Code” (p. 4–9)

Guides you through the Manchester Receiver VHDL code.

“Compiling the Manchester Receiver VHDL Files” (p. 4–17)

Explains how to compile the Manchester Receiver VHDL files.

“Developing the Manchester Receiver MATLAB Functions” (p. 4–19)

Guides you through the Manchester Receiver MATLAB function code.

“Creating a Manchester Receiver Test Bench Script” (p. 4–30)	Explains how to create a Manchester Receiver test bench script.
“Running the Manchester Receiver Simulation” (p. 4–40)	Explains how to start and monitor the Manchester Receiver test script.

## Background on Manchester Encoding

Transmission of digital data frequently requires some form of modulation to overcome limits in a physical signal channel. One technique used for modulating digital data is Manchester Encoding. This technique has the following useful characteristics:

- The transmit clock signal can be easily extracted from the received data.
- The encoded signal never produces frequency components near DC, regardless of the data, which is useful for transmission over channels that require AC coupling.
- The encoding circuit is very simple and stateless.

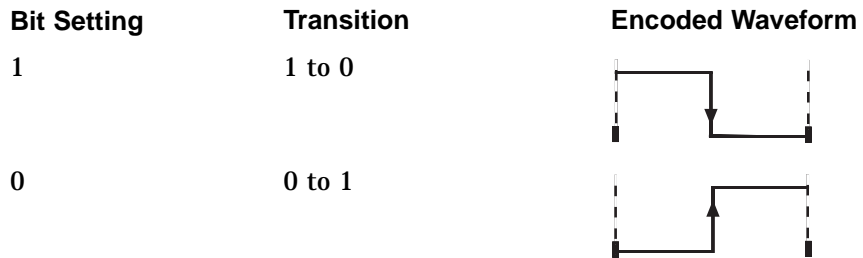
On the negative side, Manchester Encoding requires substantial bandwidth (above the Shannon limit), which tends to limit its usefulness in wireless applications. However, for connected applications, such as short haul Optical fiber and Ethernet, it is frequently a good solution.

The following sections discuss

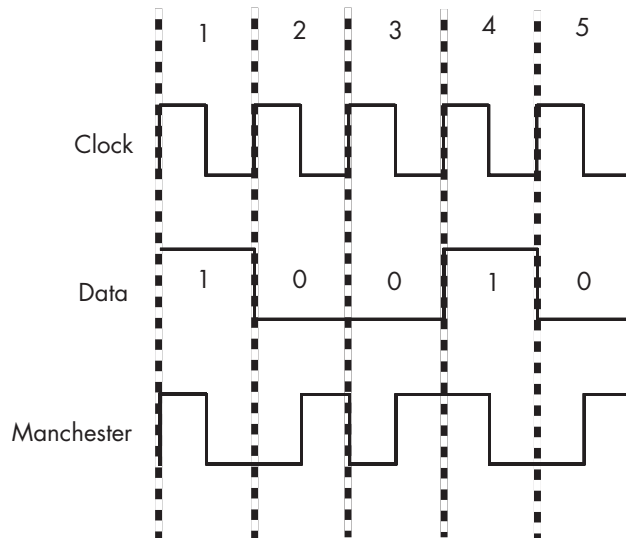
- “The Encoding” on page 4–3
- “The Receiver” on page 4–5
- “Decoding with Inphase and Quadrature Convolution” on page 4–6

### The Encoding

Manchester Encoding involves a transmitter that encodes clock and data signals in a synchronous bit stream, such that each bit represents a signal transition. The following table shows how each bit setting is defined for an encoding.



Transitions in the Manchester Encoding always occur at the center of each clock cycle. The transition at the center is defined by the bit value. Transitions at the edges of data periods are possible, depending on the values of the previous and next bits. Consider the following diagram.



As the Manchester encoded signal in the diagram shows:

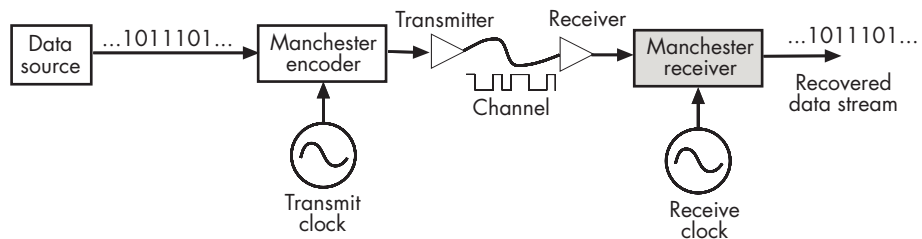
- The value of 1 for the first bit forces a high-to-low transition at the center of that bit.

- The value of 0 for the second bit forces a low-to-high transition at the center of that bit and, because the first bit transitioned from high-to-low, no transition occurs at the start of that bit.
- The value of 0 for the third bit forces a low-to-high transition at the center of that bit and because the second bit transitioned from low-to-high, a high-to-low transition occurs at the start of that bit.
- The value of 1 for the fourth bit forces a high-to-low transition at the center of that bit and, because the third bit transitioned from low-to-high, no transition occurs at the start of that bit.
- The value of 0 for the fifth bit forces a low-to-high transition at the center of that bit and, because the fourth bit transitioned from high-to-low, no transition occurs at the start of that bit.

## The Receiver

A device that receives the encoded bit stream is responsible for decoding the bit stream by extracting the data from the received signal. In most cases, the receiver must retrieve the original data stream by using the encoded signal without any additional information about the transmit clock. This simplifies the communications channel, but means the receiver must overcome the following:

- Differences between the clock used to encode the signal and the clock in the receiver, as shown in the figure below. (The highlighted component, Manchester receiver, is the component you model in this tutorial.)
- The phase between the clocks will be arbitrary.



The Manchester receiver component validates the computations performed by a Manchester receiver device that is modeled in VHDL and simulated in ModelSim. Numerous approaches are available for implementing a

Manchester receiver. The model for this tutorial uses a Delay Lock Loop (DLL) that requires the receiver to use a clock that is very close in frequency to the transmit clock. This results in a simple clock recovery circuit that has a limited frequency lock range.

The receiver clock over-samples the received data stream at 16 times the rate of the transmitter clock. Thus, the receiver clock must have a nominal period of 1/16th the data period of the transmitter clock. To compensate for minor differences between the transmitter and receiver clocks or drifts in the channel delay, the receiver clock adjusts its data period by up to one receive clock (+/-) per data period. Thus, the receiver clock can use 15, 16, or 17 cycles to recover the data encoded in the incoming sampled signal. For example, when the receiver clock is slightly faster than the transmitter clock (frequency error), the receiver clock occasionally needs to add an extra receive clock to compensate.

Large sudden phase errors, such as those that occur at startup time, require multiple data periods to acquire a good lock on the signal. By limiting the maximum phase correction to 1/16th of the total data period, the receiver can be slow to correct large phase errors.

### **Decoding with Inphase and Quadrature Convolution**

Decoding a received Manchester signal can occur in several ways, but the approach taken in the model for this tutorial is to consider Manchester Encoding as a digital phase modulation with two symbols: +180 and -180 degrees. By convolving the incoming signal with a reference inphase (I) and quadrature (Q) waveform at the modulation frequency, it is possible to extract the data and retrieve information about any phase errors in the received waveform. After one data cycle, the receiver computes two values (referred to as *isum* and *qsum* in the VHDL code), which are measurements of the inphase and quadrature convolution values. The receiver then decodes the values to predict

- The original transmitted data value for the cycle
- An estimate of the phase error between the incoming signal and the receiver's data period

A critical aspect of this design is the interpretation of the I/Q convolution measurements. At the end of a data receive cycle, the decoder translates the I/Q values into an estimate of the transmitted data and phase error. One way to visualize the receiver's condition is to plot I/Q measurements. This tutorial presents the I/Q maps of a receiver design.

Data is considered invalid if  $i_{sum}$  and  $q_{sum}$  are completely ambiguous about the data value of the received waveform.

In a similar way, you can generate an I/Q mapping of the phase adjustment value in plot format. Such a plot gives a visual representation of the decoding block. In practice, the details of this mapping have strong impact on the stability and performance of the Manchester receiver. In the ideal case where the receiver is perfectly locked to the incoming waveform, the receive cycle is 16 cycles long and the measured I/Q convolution values are easy to interpret. However, data cycles that are 15 or 17 cycles long create some bias in the measurement of the I/Q convolution. It is possible to customize the I/Q measurement during these cycles, but that would increase the size and complexity of the receiver. Instead, the data acquisition cycle is extended or shortened with no change in decoding the resulting values. However, this decoder bias can create problems with dithering or reduced noise immunity. This tutorial examines these issues.

### Setting Up Tutorial Files

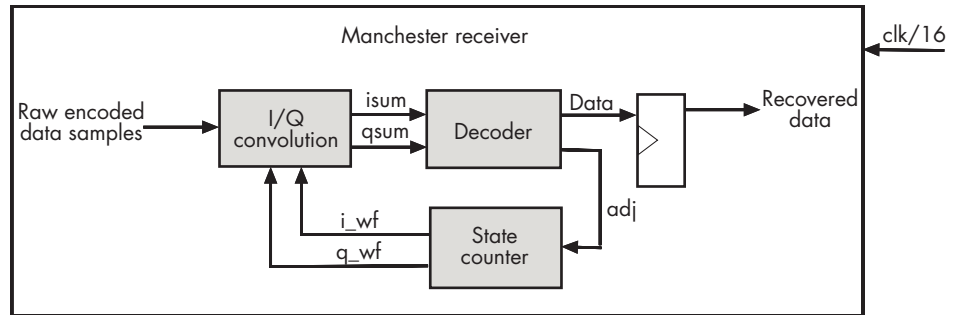
To ensure that others can access copies of the tutorial files, set up a directory for your own tutorial work:

- 1 Create a directory outside the context of your MATLAB installation directory into which you can copy the tutorial files. The tutorial in this chapter assumes that you create the directory `C:\MyPlayArea`.
- 2 Copy the contents of the `MATLABROOT/toolbox/modelsim/modelsimdemos` directory to the directory you just created.



## Developing the Manchester Receiver VHDL Code

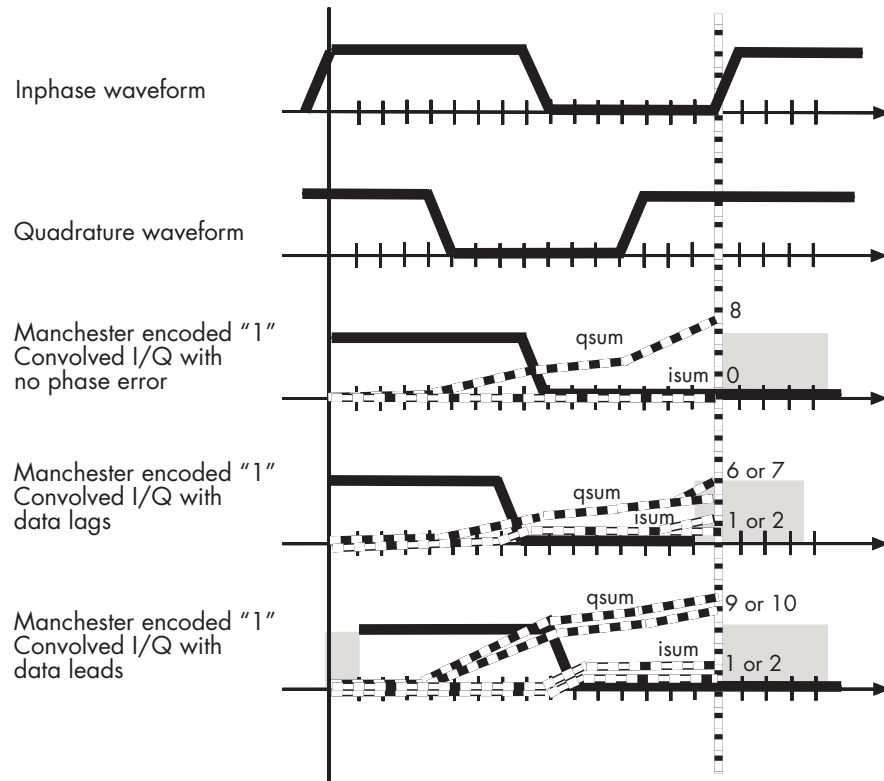
The focus of this tutorial is the verification of a VHDL implementation of a Manchester receiver. Decoding a Manchester encoded signal presents several challenges, the most prominent of which is clock recovery. The clock is embedded in the received signal and must be extracted to reproduce the original data stream. The figure below shows the Manchester receiver's model design, which is divided into three VHDL entities.



The following table describes the three sections of code.

I/Q convolver	Samples the received signal and computes the convolution for the inphase (I) and quadrature (Q) waveforms. For each waveform, the computation is implemented as the sum of XOR operations on the sample and decoded waveform received from the state counter.
Decoder	Executes a combinatorial circuit that interprets the results of the I/Q convolver.
State counter	Generates the I/Q waveforms that are convolved with received signals, taking into account phase errors (lags and leads), as necessary. The phase of the I/Q generator is adjusted to match the incoming Manchester encoded waveform. To accomplish the necessary adjustment, at the beginning of a new cycle, the state counter checks an adjustment value, $adj$ , and then changes the period of the next I/Q cycle. This adjustment value is limited to adding or removing a single clock period from the 16 periods that are nominally used for an I/Q waveform.

The following timing diagram shows an inphase waveform, quadrature waveform, and the convolved results with no phase error, data lags, and data leads.



The following sections highlight areas of code in each of the three VHDL files that are of interest for a ModelSim and MATLAB test bench. The files are located in the `modelsimdemos/vhdl/manchester` directory:

- “VHDL Code for the I/Q Convolver” on page 4–11
- “VHDL Code for the Decoder ” on page 4–14
- “VHDL Code for the State Counter” on page 4–15

### VHDL Code for the I/Q Convolver

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

- 1 Start ModelSim from MATLAB by issuing a call to the MATLAB `vsim` function.
- 2 In ModelSim, change your current directory to the `/vhdl/manchester` subdirectory you created in “Setting Up Tutorial Files” on page 4–8. If you set up the files elsewhere, adjust the path accordingly.

```
ModelSim> cd C:/MyPlayArea/vhdl/manchester
```

- 3 Open `iqconv.vhd` in the edit window with the `edit` command, as follows:

```
ModelSim> edit iqconv.vhd
```

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

- 4 Search for ENTITY `iqconv`. This statement defines the entity `iqconv`.

```
ENTITY iqconv IS
PORT (
    clk      : IN std_logic ;
    enable   : IN std_logic ;
    reset    : IN std_logic ;

    i_wf : IN std_logic ;
    q_wf : IN std_logic ;
    samp : IN std_logic ;

    isum : OUT std_logic_vector(4 DOWNTO 0);
    qsum : OUT std_logic_vector(4 DOWNTO 0);
)
END iqconv;
```

You will be verifying this entity in the MATLAB environment. Note the following:

- The name of the entity is `iqconv`. The MATLAB server assumes the default name for the corresponding MATLAB function is `iqconv`.
- 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 Link for

ModelSim interface. For a list of the supported types, see “Coding VHDL Entities for MATLAB Verification” on page 5–3.

The entity `iqconv` in this example is defined with six input ports — `clk`, `enable`, `reset`, `i_wf`, `q_wf`, and `samp` — of type `STD_LOGIC` and two output ports — `isum` and `qsum` — of type `STD_LOGIC_VECTOR`. The output ports pass simulation output data out to the MATLAB function for verification. The `reset`, `waveform`, and `sample data` input ports receive signals from the MATLAB function. As you will see in “MATLAB Function for the I/Q Convolver” on page 4–19 the MATLAB function does not use the clock signals.

---

**Note** Alternatively, the input ports can be driven with the ModelSim `force` command.

---

For more information on coding port entities for use with MATLAB, see Coding VHDL Entities for MATLAB Verification.

- 5 Browse through the rest of `iqconv.vhd`. The remaining code defines a behavioral architecture for `iqconv` that
  - a Performs an XOR on the data with each of the I/Q waveforms generated by the state counter.
  - b Performs the XOR operation.
  - c Clocks the `isum` and `qsum` into a register.

---

**Note** XOR is used here because it is the logic equivalent of multiplying two streams of data that are encoded as `-1` and `+1`. If you replace logic `'0'` with `1` and logic `'1'` with `0` in an XOR truth table, the result is a multiple that is the basis of a convolution.

---

- 6 Close the ModelSim edit window.

## VHDL Code for the Decoder

Use the ModelSim Editor to open and examine the existing file `decoder.vhd`. This section highlights areas of code in `decoder.vhd` that are of interest for a ModelSim and MATLAB test bench:

- 1 Start ModelSim, if it is not already running, from MATLAB by issuing a call to the MATLAB `vsim` function.
- 2 In ModelSim, change your current directory to the `/vhd1/manchester` subdirectory you created in “Setting Up Tutorial Files” on page 4–8. If you set up the files elsewhere, adjust the path accordingly.

```
ModelSim> cd C:/MyPlayArea/vhd1/manchester
```

- 3 Open `decoder.vhd` in the edit window with the `edit` command, as follows:

```
ModelSim> edit decoder.vhd
```

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

- 4 Search for ENTITY. This statement defines the entity decoder:

```
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;
```

You will verify this entity in the MATLAB environment. Note the following:

- The name of the entity is `decoder`. The MATLAB server assumes the name for the corresponding MATLAB function is `decoder`.
- The PORT clause for this entity, defines two input ports — `isum` and `qsum` — and three output ports — `adj`, `dvalid`, and `odata`. The input ports are 5-bit vectors of type `STD_LOGIC_VECTOR` that receive signals from the MATLAB function. The output port `adj` is a 2-bit vector of

type `STD_LOGIC_VECTOR`, and `dvalid` and `odata` are of type `STD_LOGIC`. The output ports pass simulation output data out to the function for verification. For more information on coding port entities for use with MATLAB, see “Coding VHDL Entities for MATLAB Verification” on page 5–3.

- 5 Browse through the rest of `decoder.vhd`. The remaining code defines a behavioral architecture for `decoder`. The architecture models a combinatorial circuit that translates the results of the I/Q convolver, `isum` and `qsum`, at the end of each data receive cycle, into an estimate of the transmitted data and phase error. An `adj` value of 00 indicates that the waveforms are in phase. Values of 01 and 11 indicate a data lead or lag, respectively.
- 6 Close the ModelSim edit window.

## VHDL Code for the State Counter

Use the ModelSim Editor to open and examine the existing file `statecnt.vhd`. This section highlights areas of code in `statecnt.vhd` that are of interest for a ModelSim and MATLAB test bench:

- 1 Start ModelSim, if it is not already running, from MATLAB by issuing a call to the MATLAB `vsim` function.
- 2 In ModelSim, change your current directory to the `/vhd1/manchester` subdirectory you created in “Setting Up Tutorial Files” on page 4–8. If you set up the files elsewhere, adjust the path accordingly:

```
ModelSim> cd C:/MyPlayArea/vhd1/manchester
```

- 3 Open `statecnt.vhd` in the edit window with the `edit` command, as follows:

```
ModelSim> edit statecnt.vhd
```

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

- 4 Search for `ENTITY`. This statement defines the entity `statecnt`:

```
ENTITY statecnt IS
PORT (
    clk      : IN std_logic ;
    enable   : IN std_logic ;
```

```
    reset : IN std_logic ;
    adj   : IN std_logic_vector (1 DOWNTO 0);
    sync  : OUT std_logic;
    i_wf  : OUT std_logic;
    q_wf  : OUT std_logic;
  )
END statecnt;
```

You will verify this entity in the MATLAB environment. Note the following:

- The name of the entity is `statecnt`. The MATLAB server assumes the name for the corresponding MATLAB function is `statecnt`.
- The PORT clause for this entity defines four input ports — `clk`, `enable`, `reset`, and `adj` — and three output ports — `sync`, `i_wf`, and `q_wf`. All ports except `adj` are of type `STD_LOGIC`. The input port `adj` is of type `STD_LOGIC_VECTOR` and is significant in that it receives data rate adjustments from the decoder that account for phase errors.

The output ports are of type `STD_LOGIC`. Port `sync` represents a data clock that has a nominal frequency of 1/16th of the data period. The ports `i_wf` and `q_wf` pass decoded inphase and quadrature waveforms to the I/Q convolver where they are convolved with raw sampled Manchester encoded data.

For more information on coding port entities for use with MATLAB, see [Coding VHDL Entities for MATLAB Verification](#).

- 5 Browse through the rest of `statecnt.vhd`. The remaining code defines a behavioral architecture for `statecnt`. The architecture defines two signals — `state` and `next_state` — that it uses to define a simple state machine. Signals `state` and `next_state` are of type `state_type`, an enumerated type that represents the 17 possible clock cycles. The 17th cycle accounts for data lead phase errors. When a phase is complete, the `state` signal reaches a `DECODE_ME` state, which triggers code that
  - Applies the data rate adjustment received from the decoder
  - Synchronizes the data clock with the receiver clock
  - Passes the inphase and quadrature waveforms of the current phase data to the I/Q convolver
- 6 Close the ModelSim edit window.



## Compiling the Manchester Receiver VHDL Files

After you create or edit your VHDL source files, you compile them. As part of this tutorial, set up a design library and compile `iqconv.vhd`, `decoder.vhd`, and `statecnt.vhd`:

- 1 Start ModelSim, if it is not already running, from MATLAB by issuing a call to the MATLAB `vsim` function.
- 2 Check that your current directory is set to the `/vhdl/manchester` subdirectory you created in “Setting Up Tutorial Files” on page 4–8. If you set up the files elsewhere, adjust the path accordingly.

```
ModelSim> cd C:/MyPlayArea/vhdl/manchester
```

- 3 Verify that the files are in the current directory by entering the `ls` command.
- 4 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
```

---

**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.

---

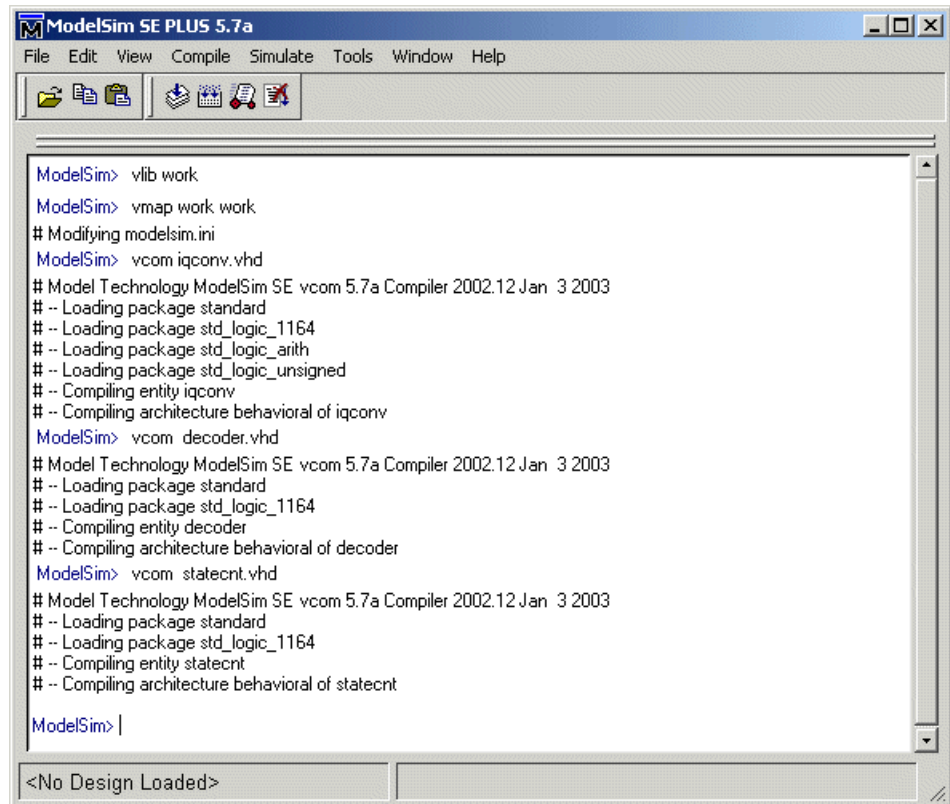
- 5 Compile the three VHDL files. One way of compiling a file is to click the filename in the project workspace and select **Compile**→**Compile All**. Another alternative is to specify the name of the VHDL file with the `vcom` command, as follows:

```
ModelSim> vcom iqconv.vhd
```

```
ModelSim> vcom decoder.vhd
```

```
ModelSim> vcom statecnt.vhd
```

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



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

ModelSim> vlib work
ModelSim> vmap work work
# Modifying modelsim.ini
ModelSim> vcom iqconv.vhd
# Model Technology ModelSim SE vcom 5.7a Compiler 2002.12 Jan 3 2003
# -- Loading package standard
# -- Loading package std_logic_1164
# -- Loading package std_logic_arith
# -- Loading package std_logic_unsigned
# -- Compiling entity iqconv
# -- Compiling architecture behavioral of iqconv
ModelSim> vcom decoder.vhd
# Model Technology ModelSim SE vcom 5.7a Compiler 2002.12 Jan 3 2003
# -- Loading package standard
# -- Loading package std_logic_1164
# -- Compiling entity decoder
# -- Compiling architecture behavioral of decoder
ModelSim> vcom statecnt.vhd
# Model Technology ModelSim SE vcom 5.7a Compiler 2002.12 Jan 3 2003
# -- Loading package standard
# -- Loading package std_logic_1164
# -- Compiling entity statecnt
# -- Compiling architecture behavioral of statecnt
ModelSim> |

<No Design Loaded>
```

## Developing the Manchester Receiver MATLAB Functions

Link for ModelSim verifies VHDL hardware in MATLAB as a function. You must develop a MATLAB function for each model component you need to verify. Given that the VHDL model for the Manchester receiver consists of three sections of VHDL code, we need three corresponding MATLAB functions:

I/Q convolver	Verifies that the VHDL I/Q convolver code computes expected output for a randomly generated stream of samples. The MATLAB function verifies this by computing the convolution for the inphase and quadrature waveforms ( <code>i_wf</code> and <code>q_wf</code> ). The computation is implemented as an XOR and accumulation of the binary signals.
Decoder	Displays a plot of the I/Q mapping generated by the decoder for visual verification.
State counter	Generates the inphase and quadrature waveforms. The MATLAB test bench function has complete control of signals applied during the simulation, including clock generation, resets, and so on.

The following sections highlight areas of code in each of the three MATLAB function files that are of interest for a ModelSim and MATLAB test bench. The files are located in `modelsimdemos`.

- “MATLAB Function for the I/Q Convolver” on page 4–19
- “MATLAB Function for the Decoder” on page 4–23
- “MATLAB Function for the State Counter” on page 4–26

### MATLAB Function for the I/Q Convolver

Typically, at this point you would create or edit a MATLAB function that meets Link for ModelSim requirements. For this tutorial, open and examine the existing file `manchester_iqconv.m`. This function

- 1 Disables resets, marking the start of a cycle.
- 2 Establishes a random cycle length of 15, 16, or 17.

```
icycle = 15 + floor(rand*3);
```

- 3 Generates three vectors of random binary states. One vector represents a data sample. The other two vectors represent the inphase and quadrature waveforms of that data sample.

```
samp_vect = randbin(icycle);  
i_wf_vect = randbin(icycle);  
q_wf_vect = randbin(icycle);
```

- 4 Uses the function `binary_xor` to compute the sum of XOR operations on the generated sample and I/Q waveforms and compares the results with the `isum` and `qsum` values received from the VHDL entity. Here, computation results produced by MATLAB are being used to verify the convolved results produced by the VHDL model.

```
test_isum = binary_xor(i_wf_vect,samp_vect);  
test_qsum = binary_xor(q_wf_vect,samp_vect);  
if (test_isum ~= bin2dec(oport.isum)),  
    disp(['Failed on iteration ' num2str(iters) ',...  
        Expected ISUM = 'dec2bin(test_isum,5) ',...  
        Received ISUM = ' oport.isum']);  
end  
if (test_qsum ~= bin2dec(oport.qsum)),  
    disp(['Failed on iteration ' num2str(iters) ',...  
        Expected QSUM = 'dec2bin(test_qsum,5) ',...  
        Received QSUM = ' oport.qsum']);  
end
```

- 5 Enables resets, marking the end of a cycle.

```
iport.reset = '1';
```

- 6 Forces the values of the test-generated sample data and I/Q waveforms onto signals connected to the VHDL entity's input ports, `samp`, `i_wf`, and `q_wf`.

```
iport.i_wf = i_wf_vect(icycle);  
iport.q_wf = q_wf_vect(icycle);  
iport.samp = samp_vect(icycle);
```

The rest of this section highlights areas of code in `manchester_iqconv.m` required for MATLAB to verify `iqconv.vhd`:

- 1 Start MATLAB, if it is not already running.
- 2 In MATLAB, change your current directory to the directory you created in “Setting Up Tutorial Files” on page 4–8. If you set up the files elsewhere, adjust the path accordingly:

```
cd C:/MyPlayArea
```

- 3 Open `manchester_iqconv.m` in the MATLAB Edit/Debug window. Use the menu option **File**→**Open** and double-click the filename `manchester_iqconv.m` or enter the edit command as follows:

```
edit manchester_iqconv.m
```

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

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

This function definition is significant in that it represents the entity test bench. When coding the function definition, consider the following:

- Names the function `manchester_iqconv`. Because this name does not match the name of the corresponding VHDL entity, you need to specify the test bench name explicitly later when you register the test bench with ModelSim.
- You *must* define the function with two input parameters, `iport` and `tnext`, and three output parameters, `oport`, `tnow`, and `portinfo`.

<code>iport</code>	Forces (by deposit) values onto signals connected to input ports of the VHDL entity — <code>reset</code> , <code>i_wf</code> , <code>q_wf</code> , and <code>samp</code> .
<code>tnext</code>	Specifies an optional time at which the MATLAB function is to be called back.
<code>oport</code>	Receives signal values from the output ports of the VHDL entity — <code>isum</code> and <code>qsum</code> — at the time specified by <code>tnow</code> .

<code>tnow</code>	Receives the simulation time at which the MATLAB function is called. By default, time is represented in seconds.
<code>portinfo</code>	For the first invocation of the MATLAB function (at the start of a simulation) only, receives an array of information that describes the ports defined for the VHDL entity.

---

**Note** You can substitute your own names for the preceding parameters. For example, the following function definition is valid:

```
function [a, b] = foo(c, d, e)
```

---

For more information on the required MATLAB function parameters, see “Setting up Expected Parameters” on page 5–13.

- 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 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.
- 5 Make note of the data types of ports defined for the entity under simulation. The Link for ModelSim interface 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.

The entity `iqconv` consists of six input ports of type `STD_LOGIC` and two output ports of type `STD_LOGIC_VECTOR`. The interface converts scalar data of type `STD_LOGIC` to a character that matches the character literal for the corresponding enumerated type. Data of type `STD_LOGIC_VECTOR` consists of a column vector of characters with one bit per character.

For more information on interface data type conversions, see “Data Type Conversions” on page 5–9.

- 6 Search for `iport.reset`. This assignment statement marks the start of a cycle by disabling resets.
- 7 Search for `oport.isum`. This line of code shows how the data that a MATLAB function receives from ModelSim might be converted to a numeric value and compared.

```
if (test_isum ~= bin2dec(oport.isum')),
```

In this case, the function receives `STD_LOGIC_VECTOR` data on `oport.isum`. The MATLAB function `bin2dec` converts the bit vector to a decimal value that can be compared to the numeric value `test_isum`.

Just below this area of code, the same conversion is performed for the bit vector `oport.qsum`.

- 8 Search for `iport.reset`. This assignment statement marks the end of a cycle by enabling a reset.
- 9 Search for `iport.i_wf`. This line of code and the two lines that follow force values onto the signals connected to VHDL entity ports `i_wf`, `q_wf`, and `samp`.
- 10 Browse through the rest of `manchester_iqconv.m`.
- 11 Close the MATLAB Edit/Debug window.

## MATLAB Function for the Decoder

Open and examine the existing file `manchester_decoder.m`. This MATLAB function

- 1 Provides a mechanism that allows you to easily reset the plot that it generates by calling `manchester_decoder` directly from the MATLAB command line with no arguments.
- 2 Sets up a timing parameter such that the simulator calls back the MATLAB function every nanosecond.

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

- 3 Sets up the layout of the plot figure window — positioning of two subplots, axis lines, and labels. One plot shows clock adjustments for phase errors. The second plot shows instances of invalid data and the values of valid data. Invalid data is data for which the clock cycle is less than 15 or

greater than 17. As part of this setup, the VHDL entity's `isum` and `qsum` values are cleared. These actions are applied during the first callback from ModelSim only.

- 4 Gets the phase error adjustment values, data valid setting, and actual sample data values from the decoder VHDL entity.
- 5 For each cycle
  - a Plots the clock adjustment data.
    - Black o indicates inphase data
    - Red < indicates data leads
    - Blue > indicates data lags
  - b Plots the instances of invalid data and values of valid data.
    - Red x indicates invalid data
    - Green o indicates valid and 0
    - Black . indicates valid and 1
  - c Creates new test values for `isum` and `qsum` and drives them to the VHDL entity.

The rest of this section highlights areas of code in `manchester_decoder.m` required for MATLAB to verify `decoder.vhd`:

- 1 Start MATLAB, if it is not already running.
- 2 In MATLAB, change your current directory to the directory you created in "Setting Up Tutorial Files" on page 4–8. If you set up the files elsewhere, adjust the path accordingly:

```
cd C:/MyPlayArea
```

- 3 Open `manchester_decoder.m` in the MATLAB Edit/Debug window. Use the menu option **File->Open** and double-click the filename `manchester_iqconv.m` or enter the edit command as follows:

```
edit manchester_decoder.m
```

- 4 Look at line 1. This line defines the name and required parameters of the MATLAB function that is to service VHDL entity decoder:



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

In this case, the function definition:

- Names the function `manchester_decoder`. Because this name does not match the name of the corresponding VHDL entity, you need to specify the test bench name explicitly later when you register the test bench with ModelSim.
- Defines the function with the required input and output parameters. The function uses
  - The `iport` parameter to force values onto signals connected to the VHDL entity’s input ports `isum` and `qsum`
  - The `tnext` parameter to register a ModelSim callback of the MATLAB function
  - The `oport` parameter to receive signal values from the entity’s output ports `adj`, `dvalid`, and `odata`

For more information on the required MATLAB function parameters, see “Setting up Expected Parameters” on page 5–13.

**5 Make note of the data types of ports defined for the entity under simulation.**

The entity `decoder` consists of two input ports — `isum` and `samp` — of type `STD_LOGIC_VECTOR` and three output ports — `adj`, `dvalid`, and `odata` — of type `STD_LOGIC`. The interface converts the scalar data to a character that matches the character literal for the corresponding enumerated type. Data of type `STD_LOGIC_VECTOR` is converted to a column vector of characters with one bit per character.

For more information on interface data type conversions, see “Data Type Conversions” on page 5–9.

- 6 Search for `tnext =`. This assignment statement registers a callback to occur one nanosecond after the current callback.
- 7 Search for `iport.isum`. This line and the line that follows, clears the entity’s `isum` and `qsum` values.
- 8 Search for `adj(isum)`. This line of code and the line below it show how the data that a MATLAB function receives from ModelSim might need to be converted for use in the MATLAB environment.

```
adj(isum) = bin2dec(oport.adj');  
data(isum) = bin2dec(oport.dvalid oport.odata)];
```

In the first case, the function receives `STD_LOGIC_VECTOR` data on `oport.adj`. The MATLAB function `bin2dec` converts the bit vector to a decimal value that is assigned to `adj(isum)`. The decimal value is used later for numeric comparisons that determine how to plot the adjustment for each `qsum` value.

In the next line of code, the function receives `STD_LOGIC` data on `oport.dvalid` and `oport.odata`. The `bin2dec` function converts the bits to a decimal value that is assigned to `data(isum)`. This decimal value is used later for numeric comparisons that determine how to plot the data validity and value information for each `qsum` value.

- 9 Search for `iport.isum`. This line of code and similar lines below it force values onto the signals connected to VHDL entity ports `isum` and `qsum`. Before the values are forced, the function `dec2bin` converts a decimal value to a bit vector. This is necessary because the VHDL entity defines `isum` and `qsum` as `STD_LOGIC_VECTOR` data.
- 10 Browse through the rest of `manchester_decoder.m`.
- 11 Close the MATLAB Edit/Debug window.

### **MATLAB Function for the State Counter**

Open and examine the existing file `manchester_statecnt.m`. This MATLAB function

- 1 Declares persistent variables `i_wf_vect`, `q_wf_vec`, and `ploti` for storing data between test bench invocations.

```
persistent i_wf_vect;  
persistent q_wf_vect;  
persistent ploti;
```

- 2 Declares the global variable `testisdone`. As a global variable, it can be accessed from outside the scope of the test bench.

```
global testisdone;
```

- 3 Sets up a timing parameter such that the simulator calls back the MATLAB function every 10 nanoseconds ( $10e^{-9}$  seconds).
- 4 Sets up the layout for a plot figure window — positioning three subplots, axis lines, and labels. The three plots show the waveforms for a long cycle, nominal cycle, and short cycle. As part of this setup, the MATLAB function clears the VHDL entity's reset value, sets its enable value, and sets its adj value to '11' (lag data).
- 5 Gets the VHDL entity's inphase and quadrature waveform data.
- 6 For each cycle, plots the long, nominal, and short cycle waveforms.

The rest of this section highlights areas of code in `manchester_statecnt.m` required for MATLAB to verify `statecnt.vhd`:

- 1 Start MATLAB, if it is not already running.
- 2 In MATLAB, change your current directory to the directory you created in “Setting Up Tutorial Files” on page 4–8. If you set up the files elsewhere, adjust the path accordingly.

```
cd C:/MyPlayArea
```

- 3 Open `manchester_statecnt.m` in the MATLAB Edit/Debug window. Use the menu option **File**→**Open** and double-click the filename `manchester_statecnt.m` or enter the edit command as follows:

```
edit manchester_statecnt.m
```

- 4 Look at line 1. This line defines the name and required parameters of the MATLAB function that is to service the entity `statecnt`:

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

In this case, the function definition:

- Names the function `manchester_statecnt`. Because this name does not match the name of the corresponding VHDL entity, you need to specify the test bench name explicitly later when you register the test bench with ModelSim.

- Defines the function with the required input and output parameters. The function uses the
  - The `iport` parameter to force values onto signals connected to the VHDL entity's input ports `reset`, `enable`, and `adj`
  - The `tnext` parameter to instruct ModelSim to call back the function every 10 nanoseconds
  - The `oport` parameter to receive signal values from the entity's output ports `i_wf`, `q_wf`, and `sync`
  - The `tnow` parameter to check whether the test bench is complete

For more information on the required MATLAB function parameters, see “Setting up Expected Parameters” on page 5–13.

### 5 Make note of the data types of ports defined for the entity under simulation.

The entity `statecnt` consists of four input ports — `clk`, `enable`, `reset`, and `adj` — and three output ports — `sync`, `i_wf`, and `q_wf`. All ports except `adj` are of type `STD_LOGIC`. The interface converts the scalar data to a character that matches the character literal for the corresponding enumerated type. The `adj` port is of type `STD_LOGIC_VECTOR`. This data is converted to a column vector of characters with one bit per character.

For more information on interface data type conversions, see “Data Type Conversions” on page 5–9.

- 6 Search for `tnext =`. This assignment statement sets up a timing parameter `tnext` such that the simulator calls back the MATLAB function every 10 nanoseconds.
- 7 Advance one line. Here, the MATLAB function uses the value of `tnow` or the presence of `portinfo` to check for the first call from the simulator.
- 8 Go to the next line. This assignment statement forces the VHDL entity's `reset` signal to a cleared state.
- 9 Go to the next line. This assignment statement forces the VHDL entity's `enable` signal to a set state, enabling the clock.
- 10 Go to the next line. This assignment statement forces the VHDL entity's `adj` signal to an initial state of '11', indicating a data lag.
- 11 Search for `tnow >`. Here, the function uses the value of `tnow` to check whether the test bench is done.

- 12 Search for `i_wf_vect`. This line of code, and the line that follows get the entity's inphase and quadrature waveform data.
- 13 Go to the next line. The MATLAB function checks whether the entity's `sync` signal is set. When this signal is set, the data clock is synchronized with the receiver clock, indicating a phase is complete.
- 14 Search for `iport.adj`. This assignment statement, and the two other `adj` assignment statements that follow, force the VHDL entity's phase adjustment to the next possible value for the next test cycle.
- 15 Browse through the rest of `manchester_statecnt.m`.
- 16 Close the MATLAB Edit/Debug window.

## Creating a Manchester Receiver Test Bench Script

Now that you are familiar with the VHDL code and MATLAB functions and have compiled the three VHDL files, this section shows you how to set up an M-code script that sets up and runs the Manchester receiver test bench simulation.

To create the test bench script, open a MATLAB Edit/Debug window and enter M-code as instructed in the following sections:

- “Documenting the Script” on page 4–30
- “Starting the MATLAB Server from the Test Script” on page 4–31
- “Writing Script Code for the Decoder Test ” on page 4–31
- “Writing Script Code for the I/Q Convolver Test” on page 4–34
- “Writing Script Code for the State Counter Test” on page 4–36

### Documenting the Script

Start writing your script by documenting at least its name and purpose. For this tutorial, open a MATLAB Edit/Debug window, include the following initial lines of comment code, and save the file as `manchester_tb.m`:

```
% Manchester Receiver Script
%
% This script sets up and executes tests for the
% following Manchester Receiver VHDL components:
%
% vhdl\manchester\decoder.vhd
%   Models a combinatorial circuit that interprets
%   the results of the inphase and quadrature
%   convolution
% vhdl\manchester\iqconv.vhd
%   Samples signals and computes the convolution for
%   inphase and quadrature waveforms
% vhdl\manchester\statecnt.vhd
%   Generates inphase and quadrature waveforms with
%   received signals, taking into account phase
%   errors
%
```

## Starting the MATLAB Server from the Test Script

Start the MATLAB server as follows:

- 1 Add the following `hdldaemon` function call:

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

This function call starts the server, such that it uses TCP/IP socket communication with a socket port number identified as available by the operating system.

- 2 Get the assigned port number by adding the following call to `hdldaemon`:

```
dstat = hlddaemon('status');
```

The 'status' argument instructs the function to return the assigned port number. The returned value is stored in the structure `dstat`.

- 3 Assign the port number portion of `dstat` to a variable for future use:

```
portnum = dstat.ipc_id;
```

Both the server and client parts of an application link must use the same port number. Thus, at some point, your script needs to forward `portnum` over to ModelSim.

- 4 Add the following global variable definition:

```
global testisdone;
```

You will use this variable as a completion flag for each test. Because the variable is global, it can verify the state of test bench execution.

## Writing Script Code for the Decoder Test

Add the script code for the decoder test as follows:

- 1 Clear the `testisdone` flag and display informational messages that inform users about what the test does.

```
testisdone = 0;
disp('=====');
disp('MATLAB testing Manchester Receiver component decoder.vhd...');
disp('Creates two plots of the entity"s transfer function,');
disp('providing a visualization of the decoder behavior.');
```

- 2 Set the project directory to a directory that has write access and is suitable for holding a ModelSim project. This tutorial assumes the writable project directory is `unixprojectdir`:

```
projectdir = pwd;
```

- 3 Change the format of the project directory and decoder VHDL file specifications to the UNIX format, which ModelSim and Tcl use, by replacing backslashes (\) with forward slashes (/):

```
% ModelSim and Tcl use the UNIX file specification format
unixprojectdir = strep(projectdir,'\','/');
unixsrcfile = strep(fullfile(matlabroot,'toolbox','modelsim',...
    'modelsimdemos','vhdl','manchester','decoder.vhd'),'','/');
```

- 4 Define a sequence of Tcl commands to be executed in the context of ModelSim. Define `tclcmd` as follows:

```
tclcmd = { ['cd ' unixprojectdir ],...
    'catch {wm geometry . 500x200+0+0}',...
    'vlib work',...
    ['vcom -performdefaultbinding ' unixsrcfile],...
    'vsimmatlab work.decoder',...
    ['matlabtb decoder -mfunc Manchester_decoder,...
    -socket ' num2str(portnum)],...
    'run 3000',...
    'quit -f'};
```

The following list explains what each Tcl command does:

- a The `cd` command changes to a writable directory.
- b The `wm` command adjusts the placement of the ModelSim window so it does not obscure the MATLAB Command Window. This command works in ModelSim SE environments only.
- c The `vlib` command creates the design library `work`.
- d The `vcom` command compiles the VHDL file. The `-performdefaultbinding` option enables default bindings in the event that they have been disabled in the `modelsim.ini` file.
- e The `vsimmatlab` command, a variant of the ModelSim `vsim` command, loads an instance of the VHDL entity decoder for MATLAB verification. This command is a Link for ModelSim extension to the ModelSim command set.
- f The `matlabtb` command initiates a MATLAB test bench session for the loaded instance of entity decoder. This command is a Link for ModelSim extension to the ModelSim command set. The command specifies:



- The entity instance.
  - The `-mfunc` option, which specifies the MATLAB function that is to test the entity (`manchester_decoder.m`). This option is required because the MATLAB function name is not the same as the entity name.
  - TCP/IP socket communication with socket port `portnum`. For a link to be established between ModelSim and MATLAB, the value specified with `-socket` must match the socket port that was specified when the MATLAB server (`hdldaemon`) was started.
- g** The `run` command starts and runs a ModelSim simulation such that it runs for 3000 iterations of the current resolution limit. By default, the simulation runs for 3000 nanoseconds.
- h** The `quit` command quits ModelSim. The `-f` option causes the command to quit without asking for confirmation.
- 5** Start ModelSim for use with the Link for ModelSim with the following call to function `vsim`:

```
vsim('startupfile','decoder.do', 'tclstart',tclcmd);
```

This command starts ModelSim with a Tcl command script that executes some general-purpose startup commands and then the user-defined commands specified with the property name/property value pair `'tclstart' tclcmd`.

The `'startupfile'` property causes `vsim` to write the entire startup Tcl command script to `decoder.do` for future reference or use.

- 6** Add the following lines of code to display informational messages and wait for `manchester_decoder.m` to run to completion:

```
disp('Waiting for testing of "decoder.vhd" to complete...');
disp('Flag from manchester_decoder.m indicates completion...');
while testisdone == 0,
    pause(0.001);
end
pause(1);
disp('MATLAB test of decoder.vhd is complete. Check the');
disp('generated plot for results.');
```

Press any key to continue to the next test.');

```
pause;
```

## Writing Script Code for the I/Q Convolver Test

Add the script code for the I/Q convolver test as follows:

- 1 Clear the `testisdone` flag and display informational messages that inform users about what the test does:

```
testisdone = 0;
disp('=====');
disp('MATLAB testing Manchester Receiver component iqconv.vhd...');
disp('Checks isum and qsum output for a randomly generated');
disp('stream of data samples.');
```

- 2 Set the project directory to a directory that has write access and is suitable for holding a ModelSim project. This tutorial assumes the writable project directory is `unixprojectdir`:

```
projectdir = pwd;
```

- 3 Change the format of the project directory and I/Q convolver VHDL file specifications to the UNIX format, which ModelSim and Tcl use, by replacing backslashes (`\`) with forward slashes (`/`):

```
% ModelSim and Tcl use the UNIX file specification format
unixprojectdir = strrep(projectdir, '\', '/');
unixsrcfile = strrep(fullfile(matlabroot, 'toolbox', 'modelsim', ...
    'modelsimdemos', 'vhdl', 'manchester', 'iqconv.vhd'), '\', '/');
```

- 4 Define a sequence of Tcl commands to be executed in the context of ModelSim. Define `tclcmd` as follows:

```
tclcmd = { ['cd ' unixprojectdir ],...
    'catch {wm geometry . 500x200+0+0}',...
    'vlib work',...
    ['vcom -performdefaultbinding ' unixsrcfile],...
    'vsimmatlab work.iqconv',...
    'force /iqconv/clock 1 0, 0 5 ns -repeat 10 ns ',...
    'force /iqconv/enable 1',...
    'force /iqconv/reset 1',...
    'run 100',...
    ['matlabtb iqconv -rising /iqconv/clock -mfunc,...
    Manchester_iqconv -socket ' num2str(portnum)],...
    'run 1000',...
    'quit -f'};
```

The following list explains what each Tcl command does:

- a** The `cd` command changes to the writable UNIX style project directory.
- b** The `wm` command adjusts the placement of the ModelSim window so it does not obscure the MATLAB window. This command works in ModelSim SE environments only.
- c** The `vlib` command creates the design library work.
- d** The `vcom` command compiles the VHDL file. The `-performdefaultbinding` option enables default bindings in the event that they have been disabled in the `modelsim.ini` file.
- e** The `vsimmatlab` command loads an instance of the VHDL entity `iqconv` for MATLAB verification. This command is a Link for ModelSim extension to the ModelSim command set.
- f** The `force` commands drive the entity's `clk`, `enable`, and `reset` signals, which get passed on to the test bench as `oport` data. The first `force` command sets `clk` at time equals 0, clears it after 5 nanoseconds, and repeats the high-to-low cycle every 10 nanoseconds. The second and third `force` commands set the `enable` and `reset` signals.
- g** The `run` command starts and runs the ModelSim simulation for 100 iterations of the current limit. By default, the simulation runs for 100 nanoseconds. This accounts for the startup phase.
- h** The `matlabtb` command initiates a MATLAB test bench session for the loaded instance of entity `iqconv`. This command is a Link for ModelSim extension to the ModelSim command set. The command specifies
  - The entity instance `iqconv`.
  - The `-rising` option, which triggers an invocation of the MATLAB function when `clk` experiences a rising edge.
  - The `-mfunc` option, which specifies the MATLAB function that is to test the entity (`manchester_iqconv.m`). This option is required because the MATLAB function name is not the same as the entity name.
  - TCP/IP socket communication with socket port `portnum`. For a link to be established between ModelSim and MATLAB, the value specified with `-socket` must match the socket port that was specified when the MATLAB server (`hdldaemon`) was started.

- i The run command starts runs the ModelSim simulation for 1000 iterations of the current resolution limit. By default, the simulation runs for 1000 nanoseconds.
  - j The quit command quits ModelSim. The -f option causes the command to quit without asking for confirmation.
- 5 Start ModelSim for use with the Link for ModelSim with the following call to function `vsim`:

```
vsim ('startupfile','iqconv.do', 'tclstart',tclcmd);
```

This command starts ModelSim with a Tcl command script that executes some general-purpose startup commands and then the user-defined commands specified with the property-value pair 'tclstart' tclcmd.

The 'startupfile' property causes `vsim` to write the entire startup Tcl command script to `iqconv.do` for future reference or use.

- 6 Add the following lines of code to display informational messages and wait for `manchester_iqconv.m` to run to completion:

```
while testisdone == 0,  
    pause(0.001);  
end  
pause(1);  
disp('MATLAB test of iqconv.vhd is complete.');
```

```
disp('If the test fails, an error message is displayed.');
```

```
disp('Press any key to continue to the next test.');
```

```
pause;
```

## Writing Script Code for the State Counter Test

Add the script code for the state counter test as follows:

- 1 Clear the `testisdone` flag and display informational messages that inform users about what the test does.

```
testisdone = 0;  
disp('=====');  
disp('MATLAB testing Manchester Receiver component statecnt.vhd...');  
disp('Creates and checks isum and qsum outputs for a randomly');  
disp('generated stream of data samples.');
```

- 2 Set the project directory to a directory that has write access and is suitable for holding a ModelSim project. This tutorial assumes the writable project directory is `unixprojectdir`.

```
projectdir = pwd;
```

- 3** Change the format of the project directory and state counter VHDL file specifications to the UNIX format, which ModelSim and Tcl use, by replacing backslashes (\) with forward slashes (/).

```
% ModelSim and Tcl use the UNIX file specification format
unixprojectdir = strep(projectdir,'\','/');
unixsrcfile = strep(fullfile(matlabroot,'toolbox','modelsim',...
    'modelsimdemos','vhdl','manchester','iqconv.vhd'),'','/');
```

- 4** Define a sequence of Tcl commands to be executed in the context of ModelSim. Define `tclcmd` as

```
tclcmd = { ['cd ' unixprojectdir ],...
    'catch {wm geometry . 500x200+0+0}',...
    'vlib work',...
    ['vcom -performdefaultbinding ' unixsrcfile],...
    'vsimmatlab -t 1ns work.statecnt ',...
    'force /statecnt/clock 1 0, 0 5 ns -repeat 10 ns ',...
    ['matlabtb statecnt -mfunc Manchester_statecnt,...
    -socket ' num2str(portnum)],...
    'run 30000',...
    'quit -f'};
```

The following list explains what each Tcl command does:

- a** The `cd` command changes to the writable UNIX style project directory.
- b** The `wm` command adjusts the placement of the ModelSim window so it does not obscure the MATLAB window. This command works in ModelSim SE environments only.
- c** The `vlib` command creates the design library `work`.
- d** The `vcom` command compiles the VHDL file. The `-performdefaultbinding` option enables default bindings in the event that they have been disabled in the `modelsim.ini` file.
- e** The `vsimmatlab` command loads an instance of the VHDL entity `statecnt` for MATLAB verification. This command is a Link for ModelSim extension to the ModelSim command set. The `-t` option specifies a ModelSim simulator time resolution of 1 nanosecond (the default).
- f** The `force` command drives the entity's `clk` signal, which gets passed on to the test bench as `oport` data. The command specifies that `clk` be

set at time equals 0, cleared after 0 after 5 nanoseconds, and that the high-to-low cycle be repeated every 10 nanoseconds.

- g** The `matlabtb` command initiates a MATLAB test bench session for the loaded instance of entity `statecnt`. This command is a Link for ModelSim extension to the ModelSim command set. The command specifies
  - The entity instance `statecnt`.
  - The `-mfunc` option, which specifies the MATLAB function that is to test the entity (`manchester_statecn.m`). This option is required because the MATLAB function name is not the same as the entity name.
  - TCP/IP socket communication with socket port `portnum`. For a link to be established between ModelSim and MATLAB, the value specified with `-socket` must match the socket port that was specified when the MATLAB server (`hdldaemon`) was started.
- h** The `run` command starts and runs the ModelSim simulation for 30000 iterations of the current resolution limit. By default, the simulation runs for 30000 nanoseconds.
- i** The `quit` command quits ModelSim. The `-f` option causes the command to quit without asking for confirmation.

- 5** Start ModelSim for use with the Link for ModelSim with the following call to function `vsim`:

```
vsim ('startupfile','statecnt.do', 'tclstart',tclcmd);
```

This command starts ModelSim with a Tcl command script that executes some general-purpose startup commands and then the user-defined commands specified with the property-value pair `'tclstart' tclcmd`.

The `'startupfile'` property causes `vsim` to write the entire startup Tcl command script to `statecnt.do` for future reference or use.

- 6** Add the following lines of code to display informational messages and wait for `manchester_statecnt.m` to run to completion:

```
while testisdone == 0,  
    pause(0.001);  
end  
pause(1);
```

```
disp('MATLAB test of statecnt.vhd is complete. Check the');  
disp('generated plot for results.');
```

```
disp('Press any key to exit test script.');
```

```
pause;
```

- 7 Save the test script file as `manchester_tb.m` and close the Edit/Debug window.

## Running the Manchester Receiver Simulation

This section explains how to start and monitor the Manchester Receiver simulation:

- 1 Start MATLAB, if it is not already running.
- 2 At the MATLAB command prompt, enter the following command:

```
manchester_tb
```

This command starts the Manchester Receiver test script that you created in “Creating a Manchester Receiver Test Bench Script” on page 4–30. The following informational messages appear in the MATLAB Command Window:

```
MATLAB testing Manchester Receiver component decoder.vhd...  
Creates two plots of the entity's transfer function  
providing a visualization of the decoder behavior.
```

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

```
Waiting for testing of 'decoder.vhd' to complete  
(flag from manchester_decoder.m indicates completion)
```

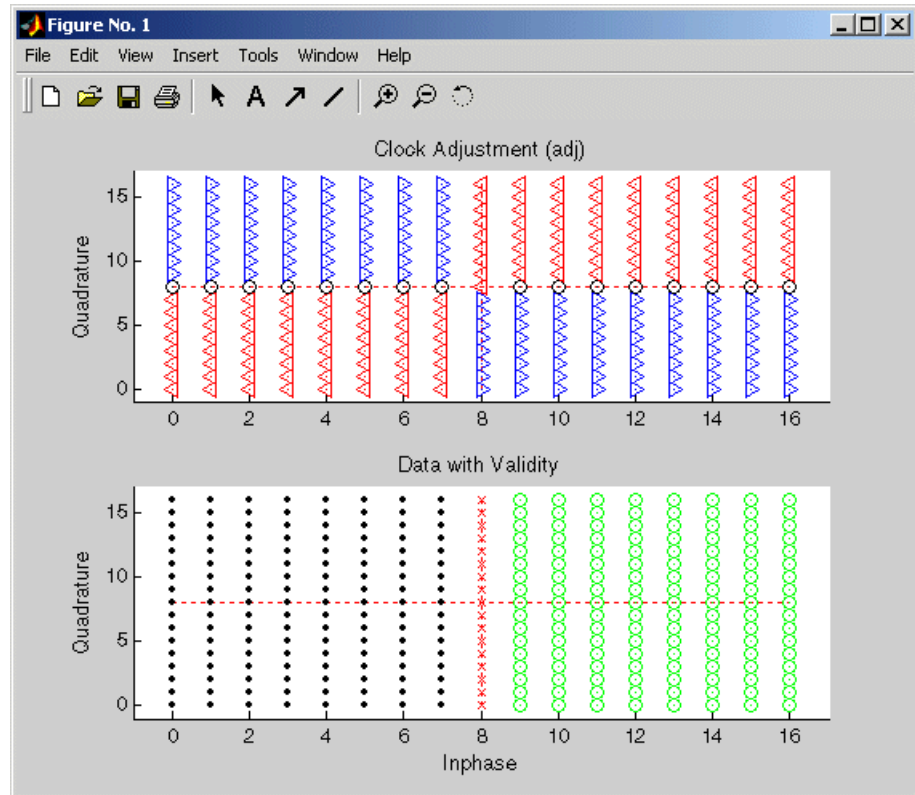
---

**Note** If the server was already running, the HDLDaemon message informs you that the existing connection is disconnected and that a new connection has been established.

---

- 3 The following figure window appears.





- 4 The decoder test then displays the following message in the MATLAB Command Window:

```
MATLAB test of decoder.vhd is complete. Check the
generated plot for results.
Press any key to continue to the next test.
```

- 5 With the input focus in the MATLAB Command Window, press any key on the keyboard. The test script starts the I/Q convolver test and displays the following:

```
MATLAB testing Manchester Receiver component iqconv.vhd...
Checks isum and qsum output for a randomly generated
stream of data samples.
MATLAB test of iqconv.vhd is complete.
```

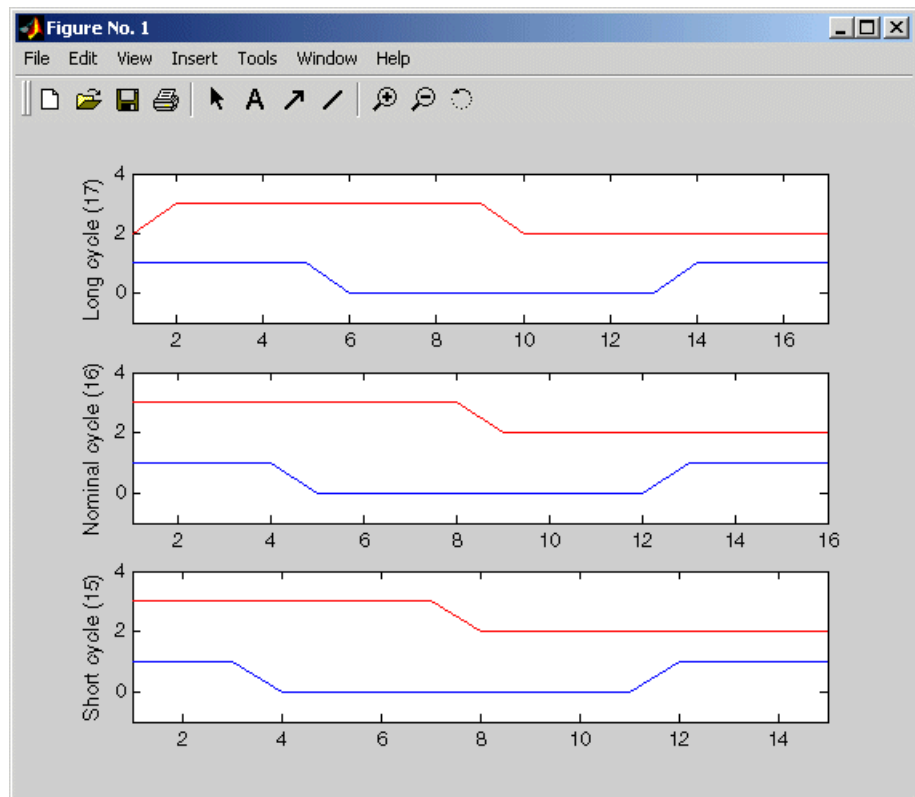
If the test fails, an error message is displayed.');

Press any key to continue to the next test.

- 6 With the input focus in the MATLAB Command Window, press any key on the keyboard. The test script starts the state counter test and displays the following:

```
MATLAB testing Manchester Receiver component statecnt.vhd...
Creates and checks isum and qsum outputs for a randomly
generated stream of data samples.
```

- 7 The following figure window appears.



- 8 The state counter test then displays the following message in the MATLAB Command Window:

MATLAB test of statecnt.vhd is complete. Check the generated plot for results.  
Press any key to exit the test script.

- 9 With the input focus in the MATLAB Command Window, press any key on the keyboard. The MATLAB prompt reappears.



# Coding a Link for ModelSim MATLAB Application

---

The Link for ModelSim provides an interface for verifying and visualizing ModelSim VHDL models within the MATLAB environment. To apply the interface, you need to code a VHDL model and a MATLAB function such that they can share data. This chapter explains what you need to do with respect to coding each of these components:

“Overview” (p. 5–2)

Provides an overview of the steps involved with coding a Link for ModelSim MATLAB application.

“Coding VHDL Entities for MATLAB Verification” (p. 5–3)

Explains how to code a VHDL entity to be verified in the MATLAB environment.

“Compiling and Debugging the VHDL Model” (p. 5–7)

Explains how to compile a VHDL design.

“Coding a MATLAB Test Bench Function” (p. 5–8)

Explains how to code a MATLAB function to verify or visualize a VHDL entity.

“Placing a MATLAB Test Bench Function on the MATLAB Search Path” (p. 5–28)

Explains how to place a MATLAB function on the MATLAB search path.

### Overview

This section provides an overview of the steps required to develop a VHDL model for use with the MATLAB component of the Link for ModelSim. To program the VHDL component of a Link for ModelSim application, do the following:

- 1 Code the VHDL model for MATLAB verification
- 2 Compile and debug the VHDL model
- 3 Code the test bench MATLAB functions
- 4 Place the MATLAB functions on the MATLAB search path

## Coding VHDL Entities for MATLAB Verification

The most basic element of communication in the Link for ModelSim interface is the VHDL entity. The interface passes all data between ModelSim and MATLAB as port data. The Link for ModelSim works with any existing VHDL entity. However, when coding a VHDL entity 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 following sections cover these topics:

- “Overview of the Steps for Coding VHDL Entities” on page 5–3
- “Choosing an Entity Name” on page 5–3
- “Specifying Ports for the Entity” on page 5–4
- “Specifying Port Direction Modes” on page 5–4
- “Specifying Port Data Types” on page 5–5
- “Sample VHDL Entity Definition” on page 5–5

### Overview of the Steps for Coding VHDL Entities

To code a VHDL entity for verification in the MATLAB environment,

- 1 Consider choosing an entity name that can be used as a valid MATLAB function name.
- 2 Determine the number of ports required and name them.
- 3 Specify a direction mode for each port.
- 4 Specify a VHDL data type that is supported by the Link for ModelSim interface for each port.
- 5 Compile the model.

The following sections provide more detail on the preceding steps.

### Choosing an Entity Name

Although not required, when naming the VHDL entity, consider choosing a name that also can be used as a MATLAB function name. (Generally, naming rules for VHDL and MATLAB are compatible.) By default, the Link for

ModelSim interface assumes that a VHDL entity and its simulation function share the same name.

For example, if you name a VHDL entity `decoder`, the Link for ModelSim interface assumes the corresponding MATLAB function is `decoder` in file `decoder.m`. If the entity and function names do not match, you must specify the MATLAB function name explicitly when you initialize a test bench session with the ModelSim `matlabtb` or `matlabtbeval` command.

---

**Note** VHDL is not case sensitive and ignores mixing of uppercase and lowercase characters in names.

---

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

### Specifying Ports for the Entity

Determine the number of ports required for the entity to be simulated and tested and name them within the `PORT` clause. Within the `PORT` clause, you can group ports that share the same direction mode or type.

### Specifying Port Direction Modes

In your entity statement, you must specify each port with a direction mode, `IN`, `OUT`, or `INOUT`. The following table defines the three modes:

Use Mode...	For Ports that...
<code>IN</code>	Represent signals that can be driven by a MATLAB function
<code>OUT</code>	Represent signal values that are passed to a MATLAB function
<code>INOUT</code>	Represent signals that can be driven by or pass values to a MATLAB function



## Specifying Port Data Types

In your entity statement, you must define each port, which you plan to test with MATLAB, with a VHDL data type that is supported by the Link for ModelSim interface. The interface can convert scalar and composite 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** If you use unsupported types, Link for ModelSim issues a warning and ignores the port at runtime. For example, if you define your interface with five ports, one of which is an access port, at runtime the interface displays a warning and your M-code sees only four ports.

---

For details on how Link for ModelSim converts data types for the MATLAB environment, see “Data Type Conversions” on page 5–9.

## Sample VHDL Entity Definition

The sample VHDL code fragment below defines the entity decoder. By default, the entity is exercised by 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 Link for ModelSim interface converts data of standard logic scalar and composite types for use in the MATLAB environment, see “Data Type Conversions” on page 5–9.

```
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 VHDL Model

After you create or edit your VHDL source files, use the ModelSim compiler to compile and debug the code. You have the option of invoking the compiler from menus in the ModelSim graphic interface or from the command line with the `vcom` command. The following sequence of ModelSim commands create and map design library `work` and compile the VHDL file `modsimrand.vhd`.

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

For more examples, see the [Link for ModelSim tutorials](#). For details on using the ModelSim compiler, see the ModelSim documentation.

## Coding a MATLAB Test Bench Function

When coding a MATLAB function that is to verify or visualize a VHDL model, 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 ModelSim. The following sections cover these topics:

- “Overview of the Steps for Coding a MATLAB Test Bench Function” on page 5–8
- “Data Type Conversions” on page 5–9
- “Naming a MATLAB Test Bench Function” on page 5–13
- “Setting up Expected Parameters” on page 5–13
- “Gaining Access to and Applying Port Information” on page 5–15
- “Converting Data for Manipulation” on page 5–17
- “Converting Data for Return to ModelSim” on page 5–18
- “Sample MATLAB Test Bench Function” on page 5–22

### Overview of the Steps for Coding a MATLAB Test Bench Function

To code a MATLAB function that is to verify or visualize a VHDL model,

- 1 Understand how the Link for ModelSim interface converts entity data for use in the MATLAB environment.
- 2 Consider naming the MATLAB function with the name of the VHDL entity the function is to test.
- 3 Define expected parameters in the function definition line.
- 4 Determine the types of port data being passed into the function.
- 5 Extract and, if appropriate for the simulation, apply information received in the portinfo structure.
- 6 Convert data for manipulation in the MATLAB environment, as necessary.
- 7 Convert data that needs to be returned to ModelSim.

The following figure shows these steps in a flow diagram.

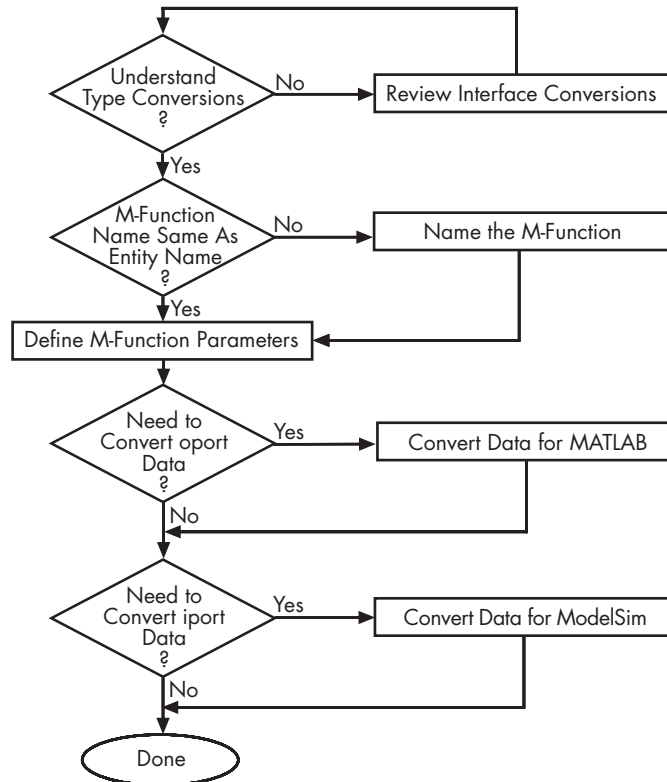


Figure 5–1: Coding a MATLAB Test Bench Function

## Data Type Conversions

The Link for ModelSim interface converts VHDL entity data to types that apply in the MATLAB environment. To program a MATLAB function for a VHDL model, you must understand the type conversions that pertain to your application. You need to know what type of data is being passed into the function so you know:

- What types of conversions are required before and after data is manipulated
- What types of conversions are required to return data to ModelSim

The following table summarizes how the Link for ModelSim converts supported VHDL data types to MATLAB types based on whether the type is scalar and composite.

---

**Note** Internally, MATLAB indexes array elements by using a column-major numbering scheme, starting with column 1. That is, MATLAB internally stores data elements from the first column first, the second column second, and so on through the last column. This tends to reverse the order of indexes between MATLAB and VHDL. Consider the following VHDL port definition:

```
PORT (  
  sta : OUT ARRAY(1 TO 2) OF BIT_VECTOR(1 TO 8););
```

In VHDL, to access the second element in the seventh column, you specify

```
sta(2)(7) <= '1'
```

The MATLAB array indexing equivalent is

```
import.sta(7,2) = '1';
```

Also, VHDL arrays are commonly defined as (0 to  $n$ ) or ( $n$  DOWNTO 0). In such cases, an offset of 1 is applied because MATLAB array indexing always begins at 1.

---

**VHDL-to-MATLAB Data Type Conversions**

<b>VHDL Types...</b>	<b>As Scalar Converts to...</b>	<b>As Composite Converts to...</b>
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 above) with one bit per character.
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 Types...	As Scalar Converts to...	As Composite Converts to...
TIME	Type <code>double</code> for time values in seconds and type <code>int64</code> for values representing simulator time increments (see the description of the 'time' option in "Starting the MATLAB Server" on page 6–7).	Arrays of type <code>double</code> or <code>int64</code> 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 <code>high</code> converts to <code>'high'</code> and the character literal <code>'c'</code> converts to <code>"c"</code> .	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 <code>(one, '2', three)</code> converts to the column vector <code>[one; "2"; three]</code> . A user-defined enumerated type that contains only character literals, converts to a vector or array of characters as indicated for the types <code>STD_LOGIC_VECTOR</code> , <code>STD_ULOGIC_VECTOR</code> , <code>BIT_VECTOR</code> , <code>SIGNED</code> , and <code>UNSIGNED</code> .



## Naming a MATLAB Test Bench Function

You can name and specify a MATLAB test bench function however you like, so long as you adhere to MATLAB function and file naming guidelines. However, keep in mind that by default the Link for ModelSim interface assumes the name for a MATLAB function matches the name of the VHDL entity that the function verifies or visualizes. For example, if you name the VHDL entity `mystdlogic`, Link for ModelSim assumes the corresponding MATLAB function is `mystdlogic` and resides in the file `mystdlogic.m`. This feature eliminates the need for you to specify the MATLAB function explicitly in commands that initialize ModelSim for test bench sessions and can simplify the documentation and tracking for a VHDL project.

---

**Note** VHDL is not case sensitive and converts names that include a mix of lower- and uppercase characters.

---

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

## Setting up Expected Parameters

The Link for ModelSim interface expects a MATLAB test bench function to be defined with the following function definition line:

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

The data that gets passed into the function through the output parameters is defined by the structure of the corresponding VHDL entity. The following table explains the purpose of each parameter:

Parameter	Purpose
<code>iport</code>	Structure that forces (by deposit) values onto signals connected to ports of the associated VHDL entity.
<code>tnext</code>	Specifies an optional time at which the MATLAB function is to be called back. If you leave this parameter unassigned or if you assign it an empty value ( <code>[]</code> ), no new entries are added to the simulation schedule. By default, time is represented in seconds. The interface accepts 64-bit integers, which are interpreted as multiples of the ModelSim resolution limit.
<code>oport</code>	Structure that receives VHDL signal values from the output ports defined for the associated VHDL entity at the time specified by <code>tnow</code> .
<code>tnow</code>	Receives the simulation time at which the MATLAB function is called. By default, time is represented in seconds. The interface also supports full 64-bit time resolution. For more information see “Starting the MATLAB Server” on page 6–7.
<code>portinfo</code>	For the first invocation of the function (at the start of the simulation) only, receives an array of information that describes the ports defined for the associated VHDL entity. For each port, the array identifies information such as the port’s type, direction, and size. The information passed to this parameter is useful for validating the entity under test. You might also consider using the port information to create a generic MATLAB function that operates differently depending on the port information supplied at startup.

---

**Note** You can substitute your own names for the preceding parameters. For example, the following function definition is valid:

```
function [a, b] = foo(c, d, e)
```

---

For more information on using `tnext` and `tnow` for simulation scheduling, see “Deciding on Test Bench Scheduling Options” on page 6–13 and “Controlling Callback Timing from a MATLAB Test Bench Function” on page 6–14. For an example of how to use these parameters, see “Sample MATLAB Test Bench

Function” on page 5–22. For more information on port data, see “Gaining Access to and Applying Port Information” on page 5–15.

## Gaining Access to and Applying Port Information

The Link for ModelSim interface passes information about the entity under test as the third parameter, `portinfo`, in the first call to your MATLAB function. The information passed to this parameter is useful for validating the entity under simulation. You might also consider using the port information to create a generic MATLAB function that operates differently depending on the port information supplied at startup. The information is supplied in three fields, as indicated below, and the content of those fields depends on the type of ports defined for the VHDL entity.

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

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

## VHDL 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 an input and output port	All port types
	tscale	Indicates the simulator resolution limit in seconds as specified in ModelSim (see the <i>ModelSim SE User's Manual &amp; Command Reference</i> )	All types
<i>field2</i>	<i>portname</i>	Is the name of the port	All port types
<i>field3</i>	type	Identifies whether the port is of type integer, real, time, or enum	All port types
	right	The VHDL RIGHT attribute	All port types
	left	The VHDL LEFT attribute	All port types
	size	The size of the matrix containing the data	All port types
	label	A character literal or label	Enumerated types, including predefined types BIT, STD_LOGIC, STD_ULOGIC, BIT_VECTOR, and STD_LOGIC_VECTOR

To use `portinfo` in your MATLAB function to verify port data, do the following:

- 1 Check whether `portinfo` data has been passed with a call to the MATLAB function `nargin`. For example:

```
if(nargin == 3),
```

- 2 If data has been passed, you can then verify it. Is the data what was expected? The following code fragment checks whether the resolution limit for time has been set to 1 ns:

```
.
.
.
    tscale = portinfo.tscale;
    if abs(tscale - 1e-9) > eps,
        error('This test requires a resolution limit of 1 ns');
    end
```

## Converting Data for Manipulation

Depending on how your simulation MATLAB function uses the data it receives from ModelSim, 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 <code>double</code>	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>

If the Function Needs To...	Then...
Convert a standard logic or bit vector to an unsigned integer	<p>Use the <code>bin2dec</code> function to convert the data to an unsigned decimal value. For example:</p> <pre>uval = bin2dec(oport.val')</pre> <p>This example assumes the standard logic 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>
Convert a standard logic or bit vector to a signed integer	<p>Use the following application of the <code>bin2dec</code> function to convert the data to a signed decimal value. For example:</p> <pre>suval = bin2dec(oport.val') - 2^length(oport.val);</pre> <p>This example assumes the standard logic 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>
Test port values of type <code>STD_LOGIC</code> and <code>STD_LOGIC_VECTOR</code>	<p>Use the <code>all</code> function as follows:</p> <pre>all(oport.val == '1'   oport.val == '0')</pre> <p>This example returns True if all elements are '1' or '0'.</p>

## Converting Data for Return to ModelSim

If your simulation MATLAB function needs to return data to ModelSim, it may be necessary for you to first convert the data to a type supported by the Link for ModelSim interface. The following table lists circumstances under which such conversions are required.

**Conversions for ModelSim**

<b>To Return Data to an IN Port of Type...</b>	<b>Then...</b>
STD_LOGIC, STD_ULOGIC, or BIT	<p>Declare the data as a character that matches the character literal for the desired logic state. For STD_LOGIC and STD_ULOGIC, the character can be 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', or '-'. For BIT, the character can be '0' or '1'. For example:</p> <pre data-bbox="724 579 1080 638"> iport.s1 = 'X'; %STD_LOGIC iport.bit = '1'; %BIT </pre>
STD_LOGIC_VECTOR, STD_ULOGIC_VECTOR, BIT_VECTOR, SIGNED, or UNSIGNED	<p>Declare the data as a column vector or row vector of characters (as defined above) with one bit per character. For example:</p> <pre data-bbox="724 812 1357 904"> iport.s1v = 'X10ZZ'; %STD_LOGIC_VECTOR iport.bitv = '10100'; %BIT_VECTOR iport.uns = dec2bin(10,8); %UNSIGNED, 8 bits </pre>

To Return Data to an IN Port of Type...	Then...
<p>Array of STD_LOGIC_VECTOR, STD_ULOGIC_VECTOR, BIT_VECTOR, SIGNED, or UNSIGNED</p>	<p>Declare the data as an array of type character with a size that is equivalent to the VHDL port size. Keep in mind that MATLAB uses a column-major numbering scheme to represent data elements internally and begins at 1. That means that MATLAB internally stores data elements from the first column first, then data elements from the second column second, and so on through the last column. For example:VHDL array indexing</p> <pre> PORT (   sta : OUT ARRAY(1 TO 2) OF   BIT_VECTOR(1 TO 8)); . . .   sta(2)(7) &lt;= '1' </pre> <p>MATLAB equivalent array indexing</p> <pre> iport.sta(7,2) = '1'; </pre>
<p>INTEGER or NATURAL</p>	<p>Declare the data as an array of type int32 with a size that is equivalent to the VHDL array size. Alternatively, convert the data to an array of type int32 with the MATLAB int32 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 portinfo structure. For example:</p> <pre> iport.int = int32(1:10)'; </pre>



To Return Data to an IN Port of Type...	Then...
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 6–7), 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>

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 5–15). 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="724 682 1323 770"> iport.char = {"A", "B"}; %Character                                 %literal iport.undef = 'mylabel'; %User-defined label </pre>
Character array for standard logic or bit representation	<p>Use the dec2bin function to convert the integer. For example:</p> <pre data-bbox="724 916 1173 942"> oport.slva =dec2bin([23 99],8)'; </pre> <p>This example converts two integers to a 2-element array of standard logic vectors consisting of eight bits.</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 Link for ModelSim interface. The VHDL entity and MATLAB function code are drawn from the decoder portion of the product’s Manchester Receiver demo. For the complete VHDL and M-code listings see the following:

```

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 program listing highlights key lines of code in the definition of the `manchester_decoder` MATLAB function. The discussion that follows the listing describes the code in detail. The numeric callouts correspond to some of the items in that discussion.

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

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

```

4      %% Compute one row and plot
      isum = isum + 1;
      adj(isum) = bin2dec(oport.adj');
      data(isum) = bin2dec([oport.dvalid oport.odata]);

      if isum == 17
      .
      .
      .

5      iport.isum = dec2bin(isum,5);
      iport.qsum = dec2bin(qsum,5);
      else
      iport.isum = dec2bin(isum,5);
      end

```

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

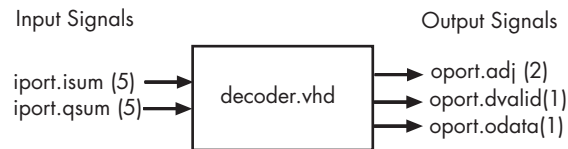
The function definition on the first line represents the communication channel between MATLAB and ModelSim. The function definition

- Names the function. This definition 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` ModelSim command.
- Defines required input and output parameters. A MATLAB test bench function *must* include two input parameters, `iport` and `tnext`, and three output parameters, `oport`, `tnow`, and `portinfo`, and *must* appear in the order shown.

<code>iport</code>	Forces (by deposit) a value onto the signal connected to the entity's input ports, <code>isum</code> and <code>qsum</code> .
<code>tnext</code>	Specifies a time value that indicates when ModelSim is to call back the MATLAB function.

- oport      Receives VHDL signal values from the entity's output ports, adj, dvalid, and odata.
- tnow      Receives the simulation time at which ModelSim calls the MATLAB function.
- portinfo   For the first invocation of the function, receives an array of information that describes the ports defined for the entity.

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



For more information on the required MATLAB function parameters, see “Setting up Expected Parameters” on page 5–13.

## 2 Make note of the data types of ports defined for the entity under simulation.

The Link for ModelSim interface 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 ModelSim and needs to return to ModelSim.

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

**Example Port Definitions**

<b>Port...</b>	<b>Of Direction...</b>	<b>And Type...</b>	<b>Converts To/Needs To Be Converted To...</b>
isum	IN	STD_LOGIC_VECTOR(4 DOWNT0 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 DOWNT0 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 DOWNT0 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.
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 “Data Type Conversions” on page 5–9.

### **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.

### **4 Convert oport data to appropriate MATLAB data types for processing.**

The two calls to `bin2dec` 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.

“Converting Data for Manipulation” on page 5–17 provides a summary of the types of data conversions to consider when coding simulation MATLAB functions.

### **5 Convert data to be returned to ModelSim.**

The three calls to `dec2bin` 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 ModelSim” on page 5–18 provides a summary of the types of data conversions to consider when returning data to ModelSim.

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

The MATLAB function associated with a VHDL entity 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:

```
which MyVhdlFunction
MyVhdlFunction not found
```

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



# Starting and Controlling MATLAB Test Bench Sessions

---

The Link for ModelSim offers flexibility in how you start and control a VHDL model test bench session with MATLAB. A session can consist of a single function invocation, a series of timed invocations, or invocations based on timing data returned by a MATLAB function to ModelSim. This chapter helps you determine what your application's scheduling requirements might be, explains how to start the most basic simulation, and explains how to apply available scheduling mechanisms for finer levels of test bench control:

“Overview” (p. 6–3)

Provides an overview of the steps for starting and controlling a MATLAB test bench session.

“Checking the MATLAB Server's Link Status” (p. 6–5)

Explains how to check the status of the MATLAB server.

“Starting the MATLAB Server” (p. 6–7)

Explains how to start the MATLAB server.

“Starting ModelSim for Use with MATLAB” (p. 6–10)

Explains how to start ModelSim for use with MATLAB.

“Loading a VHDL Entity for Verification” (p. 6–12)

Explains how to load a VHDL entity in ModelSim for simulation and verification with MATLAB.

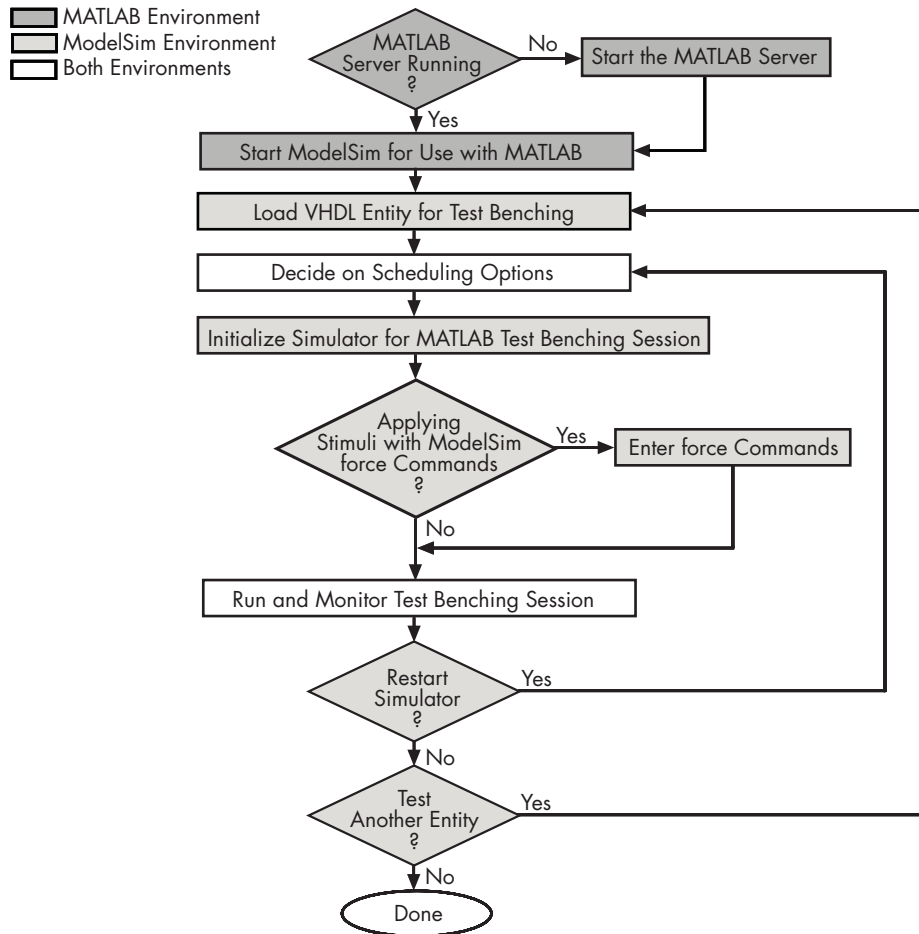
“Deciding on Test Bench Scheduling Options” (p. 6–13)	Describes different ways of scheduling the invocations of a MATLAB test bench function.
“Controlling Callback Timing from a MATLAB Test Bench Function” (p. 6–14)	Explains how to control callback timing from a MATLAB test bench function.
“Initializing the Simulator for a MATLAB Test Bench Session” (p. 6–16)	Explains how to initialize the ModelSim simulator for use with MATLAB as a test bench tool.
“Applying Stimuli with the ModelSim force Command” (p. 6–21)	Explains how to apply test bench stimuli with ModelSim force commands.
“Running and Monitoring a Test Bench Session” (p. 6–22)	Explains how to run and monitor test bench session.
“Restarting a Test Bench Session” (p. 6–25)	Explains how to restart ModelSim during a test bench session.
“Stopping a Test Bench Session ” (p. 6–26)	Explains how to stop a test bench session.

## Overview

To start and control the execution of a simulation in the MATLAB environment,

- 1 Check the MATLAB server's link status.
- 2 Start the MATLAB server.
- 3 Launch ModelSim for use with MATLAB.
- 4 Load a VHDL entity in ModelSim for simulation and verification with MATLAB.
- 5 Decide on how you want to schedule invocations of the MATLAB test bench function.
- 6 Initialize the ModelSim simulator for use with MATLAB as a test bench tool.
- 7 Apply test bench stimuli.
- 8 Run and monitor the test bench session.
- 9 Restart simulator during a test bench session.
- 10 Stop a test bench session.

The following figure shows the steps in a flow diagram.



## Checking the MATLAB Server's Link Status

The first step to starting a ModelSim 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:

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

If the server is not running, the message reads

```
HDLDaemon is NOT running
```

To determine the mode of communication and TCP/IP socket port in use, assign the return value of the function call to a variable. For example:

```
x=hdldaemon('status')
HDLDaemon socket server is running on port 4449 with 0 connections
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 connections. 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]
```

## Starting the MATLAB Server

If the MATLAB server is not running, start it:

- 1 Start MATLAB.
- 2 In the MATLAB Command Window, call the `hdldaemon` function with property name/property value pairs that specify whether the Link for ModelSim interface is to
  - 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...)
```

The following table explains when and how to specify property name/property value pairs.

---

**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 ModelSim simulator for use with a MATLAB with the `matlabtb` or `matlabtbeval` ModelSim command. In addition, if you specify TCP/IP socket mode, the socket port that you specify with this function and the ModelSim command must match. For more information on modes of communication, see “Choosing TCP/IP Socket Ports” on page 1–17. For more information on establishing the ModelSim end of the communication link, see “Initializing the Simulator for a MATLAB Test Bench Session” on page 6–16.

---

**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 1–17.

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'.

The following function call starts the server in TCP/IP socket mode, using port number 4449, with a time resolution of seconds (the default).

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

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.

## Starting ModelSim for Use with MATLAB

Start ModelSim directly from MATLAB by calling the MATLAB function `vsim`. This function starts and configures the ModelSim simulator (`vsim`) for use with the MATLAB feature of the Link for ModelSim. 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 ModelSim starts.

You can customize the DO file that starts ModelSim by specifying the call to `vsim` with the following property name\property value pairs:

---

### Notes

- The `vsim` function overrides any options previously defined by the `setupmodelsim` function.
- To start ModelSim from MATLAB with a default configuration previously defined by `setupmodelsim`, issue the command `!vsim` at the MATLAB command prompt.

---

### To...

Include one or more Tcl commands in the DO file that are to execute after ModelSim launches

### Specify...

'`tclstart`', '`tcl_commands`', where `tcl_commands` is a command string or cell array of command strings, which can include the `matlabtb` and `matlabtbeval` ModelSim commands that initialize the simulator for a test bench session (see “Initializing the Simulator for a MATLAB Test Bench Session” on page 6–16)

**To...**

Start a specific version of the simulator or a version of the simulator that is not on the system path

Create a DO file for future reference or use

**Specify...**

'vsimdir', 'pathname', where pathname identifies the path and file name for the version of the simulator executable you want to launch

'startupfile', 'pathname', where pathname specifies a path and file name for the generated DO file

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 ModelSim simulator for test bench instance `modsimrand`.

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

## Loading a VHDL Entity for Verification

After you start ModelSim from MATLAB with a call to `vsim`, load an instance of a VHDL entity for verification with the ModelSim command `vsimmatlab`. At this point, it is assumed that you have coded and compiled your VHDL model as explained in Chapter 5, “Coding a Link for ModelSim MATLAB Application”. Issue the ModelSim command `vsimmatlab` for each instance of an entity in your model that you want to cosimulate. For example:

```
vsimmatlab work.modsimrand
```

This command opens a simulation workspace for `modsimrand` and displays a series of messages in the ModelSim command window as the simulator loads the entity’s packages and architectures.

## Deciding on Test Bench Scheduling Options

By default, the Link for ModelSim interface 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 Link for ModelSim interface is to invoke the relevant MATLAB function. Depending on your choices, you may need to modify the function or specify specific arguments when you initiate a MATLAB test bench session with the `matlabtb` or `matlabtbeval` command.

You can schedule a MATLAB simulation function to execute

- At a time that the MATLAB function passes to ModelSim with the `tnext` input 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, see “Controlling Callback Timing from a MATLAB Test Bench Function” on page 6–14. For information on setting other scheduling parameters, see “Initializing the Simulator for a MATLAB Test Bench Session” on page 6–16.

## Controlling Callback Timing from a MATLAB Test Bench Function

You can control the callback timing of a MATLAB test bench function by using that function's `tnext` input parameter. This parameter passes a time value to ModelSim, 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 string or value of type `double` or `int64`. The following table explains how the interface converts each type of data for use in the ModelSim environment.

### Time Representations for `tnext` Parameter

If You Specify a...	The Interface...
String that includes a unit specification	<p>Parses the string as a scaled time value with units of fs (femtoseconds), ps (picoseconds), ns (nanoseconds), us (microseconds), ms (milliseconds), or sec (seconds). The value is scaled to the nearest multiple of the current time value resolution. For example, the following string scales to the simulation time nearest to 12.2 nanoseconds as a multiple of the current ModelSim time resolution.</p> <pre>tnext = '12.2 nsec'</pre>
String that does not specify units	<p>Parses the string as the number of ticks based on the ModelSim time resolution limit. For example, the following string parses to 100 ticks of the current time resolution.</p> <pre>tnext = '1e2'</pre>

If You Specify a...	The Interface...
double value	<p>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 ModelSim time resolution.</p> <pre data-bbox="842 491 1013 517">tnext = 1e-9</pre>
int64 value	<p>Converts to an integer multiple of the current ModelSim time resolution limit. For example, the following value converts to 100 ticks of the current time resolution.</p> <pre data-bbox="828 725 1055 751">tnext=int64(100)</pre>

---

**Note** The `tnext` parameter represents time from the start of the simulation. Therefore, `tnext` should always be greater than `tnow`.

---

## Initializing the Simulator for a MATLAB Test Bench Session

Once you decide on the controls you need to apply for a test bench, you are ready to initialize the ModelSim simulator for a specific MATLAB test bench session. You initialize ModelSim for a cosimulation session with the `matlabtb` or `matlabtbeval` command. These commands

- Identify the instance of an entity in the VHDL model being simulated and test benched
- Define the communication link between ModelSim and MATLAB
- Specify a callback to a MATLAB function that executes in the context of MATLAB on behalf of the instance under simulation in ModelSim

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

You must specify at least one instance of an entity in your VHDL model. By default, the command applies a shared memory communication link and attaches the specified instance to a MATLAB function that has the same name as the instance. For example, if the instance is `modsimrand`, the command links the instance with the MATLAB function `modsimrand` in file `modsimrand.m`. Alternatively, you can specify a different function name with the option `-mfunc`.

To apply TCP/IP socket communication, specify the command with the `-socket` option and a TCP/IP specification. If ModelSim 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 host name or Internet address in addition to the socket port. The following table lists different ways of specifying a TCP/IP specification.

<b>Format</b>	<b>Example</b>
Port number	4449 on this computer
Port alias	matlabservice on this computer
Port number and remote host name	4449@compa



<b>Format</b>	<b>Example</b>
Remote host name and port number	compa:4449
Port alias and remote host Internet address	matlabservice@123.34.55.23

For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports” on page 1–17.

---

**Note** The communication mode and, if appropriate, the TCP/IP specification that you specify with the `matlabtb` or `matlabtbeval` command must match what you specify for the communication mode when you call the `hdldaemon` function in MATLAB. For more information on modes of communication, see “Modes of Communication” on page 1–8. For information on choosing socket ports, see “Choosing TCP/IP Socket Ports” on page 1–17. For more information on starting the MATLAB end of the communication link, see “Starting the MATLAB Server” on page 6–7.

---

The `matlabtbeval` command executes the MATLAB function once at the start of the simulation, while `matlabtb` provides several options for scheduling MATLAB function execution. The following table lists the various scheduling options.

---

**Note** For time-based parameters, you can specify any standard time units (ns, sec, and so on). If you do not specify units, the command treats the time value as an integer value of simulation intervals.

---

**Simulation Scheduling Options**

<b>To Specify that the MATLAB function Execute...</b>	<b>Include...</b>	<b>Where...</b>
At explicit times	time[, ...]	<p>time represents one of n time values, past time 0, at which the MATLAB function is to execute.</p> <p>For example:</p> <p style="padding-left: 40px;">10 ns, 10 ms, 10 sec</p> <p>The MATLAB function executes when time equals 0 and then 10 nanoseconds, 10 milliseconds, and 10 seconds from time zero.</p>
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 is to execute 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>

To Specify that the MATLAB function Execute...	Include...	Where...
When a specific signal experiences a rising or falling edge	<pre>-rising signal[, ...] -falling signal[, ...]</pre>	<p>signal represents a pathname of a signal defined as a logic type — STD_LOGIC, BIT, X01, and so on.</p>
A sensitivity list	<pre>-sensitivity signal[, ...]</pre>	<p>signal represents a pathname 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>-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 pathname format. If you do not specify a full pathname, the command applies ModelSim rules to resolve signal specifications.

---

Consider the following `matlabtb` 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 VHDL 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 `matlabtbval` 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

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

## Applying Stimuli with the ModelSim force Command

Once you establish a link between ModelSim 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

- Force the `clk` signal to 0 at 0 nanoseconds after the current simulation time and to 1 at 5 nanoseconds after the current ModelSim 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.



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

## Running and Monitoring a Test Bench Session

Start a test bench session from ModelSim. ModelSim offers a number of options for running a simulation to debug, analyze, or verify a VHDL model. A typical sequence for running a simulation interactively from the main ModelSim window is shown below:

- 1 Start the simulation by entering the ModelSim run command or the **Simulate>Run** option in the Main window.

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 a number of time steps. Alternatively, you can specify the `-all` option, which causes the simulation to run forever, until the simulation hits a breakpoint, or a breakpoint event occurs.

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

```
run 50000
```

- 2 Set breakpoints in the VHDL and MATLAB code to verify and analyze simulation progress and correctness. The following table lists ways you can set breakpoints in each application environment.

### ModelSim Environment

Enter the `bp` command

Click **Simulate>Break** in the **Main** window

Click the **Break** button on the **Main** or **Wave** window toolbar

### 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

Click **Set/Clear Breakpoint** on the **Breakpoints** menu

Click **Set/Clear Breakpoint** on the context menu

Call the `dbstop` function

The following ModelSim command sets a breakpoint at line 50 in the VHDL file `modsimrand.vhd`:

```
bp modsimrand.vhd 50
```

- 3 Step through the simulation and examine values. The following table lists ways you can step through code in each application environment.

**ModelSim Environment**

Click the **Step** or **Step Over** button on the **Main** or **Wave** window toolbar

Click the **Step** or **Step-Over** options on the **Simulate>Run** menu

Enter the step command

**MATLAB Environment**

Click the **Step**, **Step-In**, or **Step-Out** toolbar button

Click the **Step**, **Step-In**, or **Step-Out** option on the **Debug** menu

Click the **Go Until Cursor** menu option

Call the `dbstep` function

- 4 When you block execution of the MATLAB function, ModelSim also blocks and remains blocked until you clear all breakpoints in the function's M-code.
- 5 Resume the simulation, as needed. The following table lists ways you can resume a simulation in each application environment.

**ModelSim Environment**

Click the **Run Continue** button on the **Main** or **Wave** window toolbar

Click the **Continue** option on the **Simulate>Run** menu

Enter the run command with the `-continue` option

**MATLAB Environment**

Click the **Continue** toolbar button

Click the **Continue**, **Run**, or **Save and Run** option on the **Debug** menu

Call the `dbcont` function

The following ModelSim command instructs `vsim` to resume a simulation:

```
run -continue
```

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



## Restarting a Test Bench Session

Because ModelSim issues the service requests during a MATLAB test bench, you must restart a test bench session from ModelSim. To restart a session,

- 1 Make ModelSim your active window, if your input focus was not already set to that application.
- 2 Reload VHDL design elements and reset the simulation time to zero by doing one of the following:
  - Click the **Restart** button on the **Source Window** toolbar
  - Click the **Restart** option on the **Simulate->Run** menu
  - Enter the restart command in the main window
- 3 Reissue the `matlabtb` 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 Test Bench 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 ModelSim your active window, if your input focus was not already set to that application.
- 2 Halt the simulation by clicking the **Simulate>End Simulation** option on the main window.
- 3 Close your project by clicking the **File>Close>Project** option on the main window.
- 4 Exit ModelSim, 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 ModelSim sessions, see the ModelSim online help or documentation.

# Modeling and Verifying a VHDL Design with Simulink

---

Simulink is a software package used widely in academia and industry to model and simulate dynamic systems. Together, ModelSim, Simulink, and Simulink blocksets provide a powerful modeling and cosimulation environment for Electronic Design Automation (EDA). This chapter explains how to set up a cosimulation environment in Simulink that includes VHDL models designed and simulated with ModelSim.

“Overview” (p. 7–3)

Provides an overview of the process for integrating Link for ModelSim blocks into a Simulink design.

“Creating a Hardware Model Design in Simulink” (p. 7–5)

Lists questions to think about as you decide to include Simulink in an EDA solution.

“Handling of Signal Values Across Simulation Domains” (p. 7–8)

Explains how the Link for ModelSim addresses the differences in treatment of simulation time in ModelSim and Simulink.

“Configuring Simulink for VHDL Models” (p. 7–17)

Gives suggestions for configuring Simulink more optimally for use with Link for ModelSim blocks.

“Running and Testing a Hardware Model in Simulink” (p. 7–19)

Suggests fully testing a Simulink model into which you plan to later integrate Link for ModelSim blocks.

“Starting ModelSim for Use with Simulink ” (p. 7–20)	Introduces the tools for coding the VHDL components of a cosimulation model and explains how to establish the communication link between Simulink and ModelSim.
“Loading a VHDL Entity for Cosimulation” (p. 7–23)	Explains how to load an instance of a VHDL entity for cosimulation in ModelSim.
“Adding the VHDL Representation of a Model Component into a Simulink Model” (p. 7–24)	Explains how to integrate the VHDL representation of a model component into a Simulink model with Link for ModelSim blocks.
“Configuring a VHDL Cosimulation Block” (p. 7–26)	Explains how to use a Simulink block parameters dialog to configure Link for ModelSim blocks.
“Running and Testing a Cosimulation Model in Simulink” (p. 7–42)	Explains how to use the To VCD File block to generate VCDs.
“Using a Value Change Dump File for Design Verification” (p. 7–43)	Explains how to start a cosimulation model in Simulink. This section also explains how to reset clocks and restart ModelSim during testing.

## Overview

Link for ModelSim blocks link hardware components that are concurrently simulating in ModelSim to the rest of a Simulink model.

Two potential use scenarios follow:

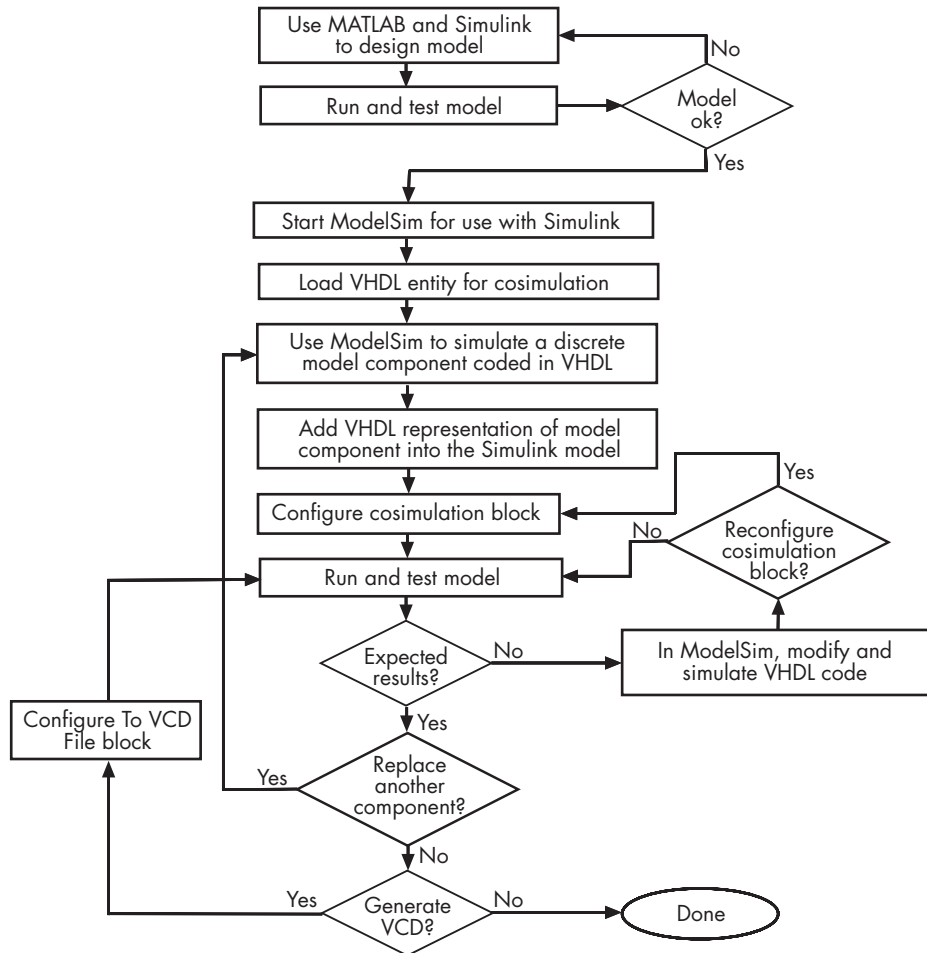
- A single VHDL Cosimulation block fits into the framework of a larger system-oriented Simulink model.
- The Simulink model is a collection of VHDL Cosimulation blocks, each representing a specific hardware component.

The following process shows the typical workflow for integrating VHDL Cosimulation blocks into a Simulink design that includes one or hardware components:

- 1 Design your application model in Simulink. One or more components of the model can represent hardware that you intend to describe with VHDL.
- 2 Run and test the model design in Simulink.
- 3 Verify that the model runs as expected. If it does not, repeat steps 1 and 2 to rework and fine tune the design.
- 4 Use ModelSim to simulate a discrete model component of the design coded in VHDL.
- 5 Integrate the VHDL representation of the model component into the Simulink model as a VHDL Cosimulation block.
- 6 Configure the VHDL Cosimulation block. The block parameters dialog includes tabs for configuring port, communication, clock, and tool command language (Tcl) parameters.
- 7 Run and test the revised model design in Simulink.
- 8 Verify that the revised model runs as expected. If it does not,
  - a Modify the VHDL code and simulate it in ModelSim.
  - b Determine whether you need to reconfigure the VHDL Cosimulation block. If you do, repeat steps 6 to 8. If you do not, repeat steps 7 and 8.

- 9 Determine whether you need to replace another component of the Simulink model with a VHDL Cosimulation block. If you do, go to step 4.
- 10 Consider using a To VCD File block to verify cosimulation results.

The following figure shows the steps in a flow diagram.



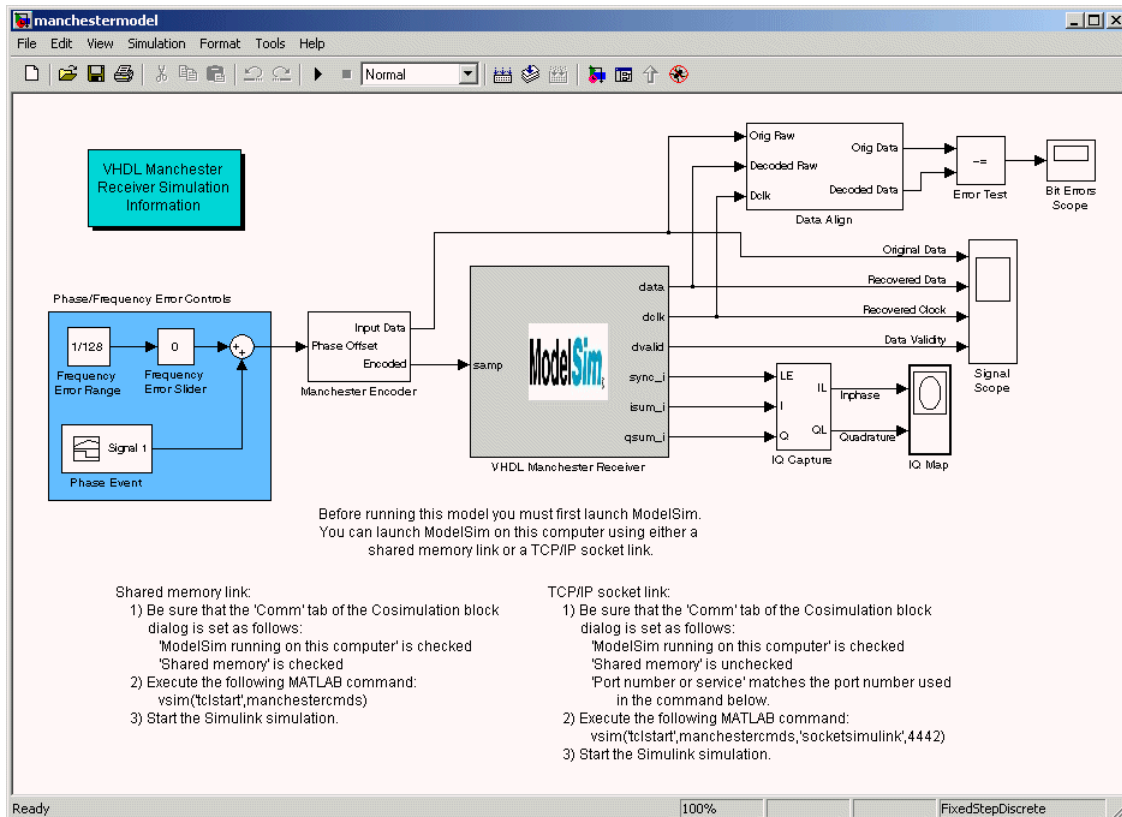
## Creating a Hardware Model Design in Simulink

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

- Will you start by developing a VHDL application, using ModelSim, and possibly MATLAB, 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, once the model runs as expected, replace the black boxes with VHDL Cosimulation blocks?
- What other Simulink blocksets might apply to your application? Blocksets of particular interest for EDA applications include the Communications, DSP, and Fixed-Point Blocksets.
- Will you set up VHDL 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 VCD?

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

The following window display shows a sample Simulink model that includes a Link for ModelSim block.



The VHDL Cosimulation block models a Manchester receiver that is coded in VHDL. 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
- Bit Errors block



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

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

## Handling of Signal Values Across Simulation Domains

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

As you develop a Link for ModelSim cosimulation application, you should be familiar with how signal values are handled across the simulation domains with respect to

- “How Simulink Drives Cosimulation Signals” on page 7–8
- “Representation of Simulation Time” on page 7–9
- “Handling of Multirate Signals” on page 7–10
- “Block Simulation Latency” on page 7–11

### How Simulink Drives Cosimulation Signals

Although you can connect the output ports of a Link for ModelSim cosimulation block to any signal in a VHDL entity’s hierarchy, you must use some caution when connecting signals to input ports. Simulink uses the deposit method of changing signal values to drive input to a cosimulation block. The deposit method is the weakest method of forcing a VHDL signal and can produce unexpected or undesired results when a signal is driven by multiple sources. To avoid such conditions, you should attach the input ports to signals that are not driven, such as the input ports of a top-level VHDL entity.

If you need to use a signal that has multiple drivers and it is resolved (for example, it is of 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.

## Representation of Simulation Time

A significant difference to note between the ModelSim and Simulink applications is the representation of simulation time. ModelSim simulation occurs at multiples of a resolution limit that is user defined in increments of 10 (for example, 1 ns, 10 ns, 100 ns, and so on). By default, events in a ModelSim simulation occur at increments of 1 nanosecond.

Simulink maintains simulation time as a double-precision value scaled to seconds. This more complex representation accommodates continuous models and discrete controllers. The differences in representation may cause timing errors between ModelSim and Simulink simulation times.

To address this, Link for ModelSim blocks interpret Simulink sample time as integer multiples of the ModelSim resolution limit and *not* as seconds. All sample times in a Simulink model that includes Link for ModelSim blocks are scaled as follows:

$$\text{actual sample time} = \frac{\text{specified Simulink sample time}}{\text{ModelSim resolution limit}}$$

In addition, the Link for ModelSim interface limits the cosimulation blocks to operate as fixed-rate devices. Signals driven and read by the blocks have a defined, fixed sample time.

When you configure output ports for cosimulation blocks, you have the option of specifying a sample time for block output ports. The value you specify must be an integer multiple of the resolution limit that is defined in ModelSim. For example, if the resolution limit defined in ModelSim is 10 nanoseconds, you can specify a sample time that is any multiple of 10; a value of 200 causes Simulink to interact with ModelSim every 2 microseconds.

In general, Simulink handles the sample time for a cosimulation block's ports 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 times, it applies the user-specified output sample time to all output ports. If you do not specify an explicit output sample time, Simulink applies the fastest input sample time to all output ports.

As you develop a Simulink model for use with ModelSim, consider the following sample time guidelines:

- Specify the output sample time for a cosimulation block as an integer multiple of the resolution limit defined in ModelSim. Use the ModelSim command `report simulator state` to check the resolution limit of the loaded model. If the ModelSim resolution limit is 10 nanoseconds and you specify a block's output sample time as 200, Simulink interacts with ModelSim every 2 microseconds.
- Specify the Simulink model's start and stop time values (see the Solver tab of the model's Simulation Parameters dialog) as integers. Link for ModelSim calculates the actual simulation start and stop times by multiplying the integer you specify by number of ticks of the ModelSim resolution limit.
- Derive the sample times for discrete blocks that interact with a cosimulation block from the cosimulation block's sample time.
- Use the Simulink Zero-Order Hold block to apply a zero-order hold (ZOH) on continuous signals that are driven into a cosimulation block.
- Scale the sample times of all continuous signals driven or read by continuous blocks in the model, using the equation shown above.

### Handling of Multirate Signals

Link for ModelSim supports the use of multirate signals, signals that are sampled or updated at different rates, in a single VHDL Cosimulation block. A cosimulation block exchanges data for each signal at the Simulink sample rate for that signal. For input signals, a cosimulation block accepts and honors all signal rates.

Although cosimulation blocks can support only one sample time for all output signals, you can produce multirate results. First, determine by how much the desired output rates differ. If the differences are small, for example, two to eight times a common base rate, do the following:

- 1 Identify the fastest rate.
- 2 Configure the cosimulation block to use the fastest output sample time for all output signals.
- 3 Use a Simulink rate converter block to produce the required rate variations, outside the scope of the cosimulation block.

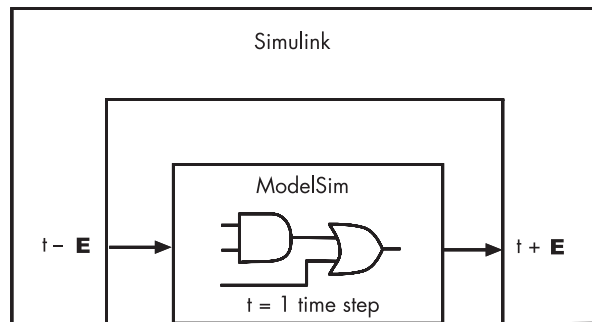
This approach requires some additional communications overhead, but in cases where the difference in rates is small, the overhead is small and may be acceptable.

For all other cases, use separate cosimulation blocks, each specified with a different output sample time.

## Block Simulation Latency

Simulink and the Link for ModelSim cosimulation blocks supplement the hardware simulator environment, rather than operate as part of it. During cosimulation, Simulink does not participate in ModelSim 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 ModelSim and Simulink with regard to use and treatment of simulation time, some degree of latency is introduced when you use Link for ModelSim cosimulation blocks. The latency is a time lag that occurs between when Simulink initiates the deposit of a signal and when the effect of the deposit is visible on cosimulation block output.

Consider the following figure:



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

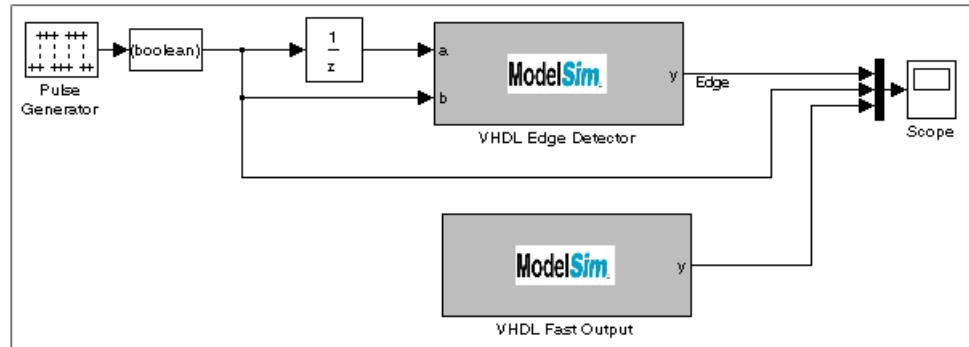
Regardless of whether your VHDL code is specified with latency, the cosimulation block has a minimum latency that is equivalent to the cosimulation block's output sample time. For large sample times, the delay can appear to be quite long, but this is an artifact of the cosimulation block, which exchanges 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 contain pure combinatorial paths, however, it might be necessary to adjust the sample time frequency to achieve simulation performance required for circuit analysis.

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  $z^{-1}$  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.

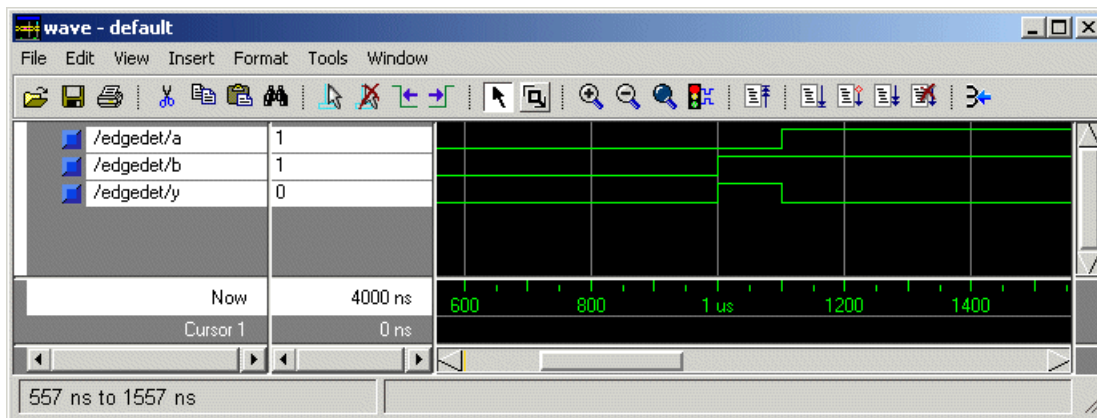
The second cosimulation block, VHDL Fast Output, is a source block that reads the output of the XOR circuit and passes it on to the Scope block for display.

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

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

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

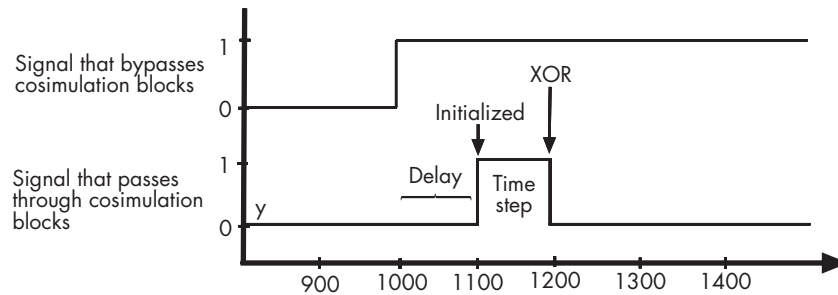


Note the following:

- 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.



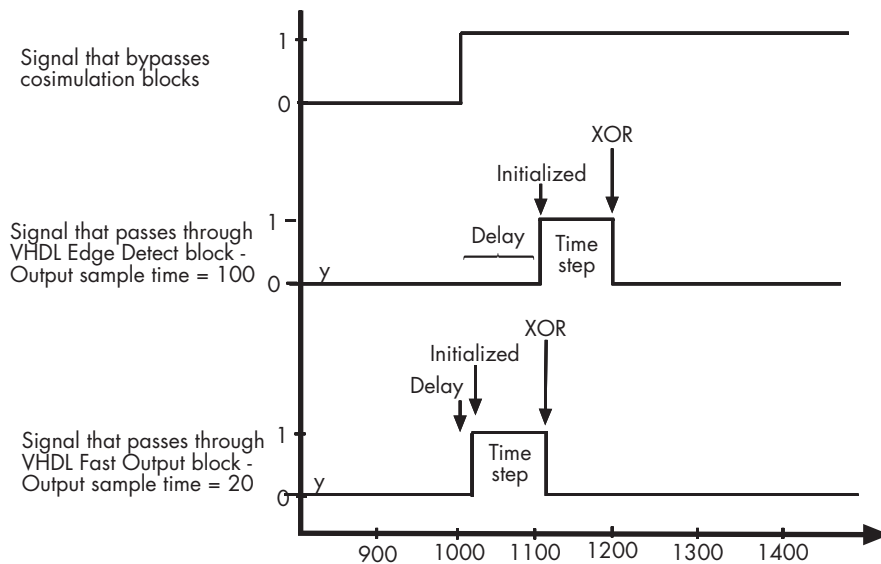
The following figure highlights the individual signal paths that get appear in the Simulink Scope window.



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.

---

## Configuring Simulink for VHDL Models

When you create a Simulink model that includes one or more Link for ModelSim blocks, you might want to adjust certain Simulink parameter settings to best meet the needs of VHDL modeling. For example, you might want to adjust the value of the **Stop time** parameter in the **Simulation 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:

Parameter	Default Setting
'SingleTaskRateTransMsg'	'error'
'Solver'	'fixedstepdiscrete'
'SolverMode'	'singletasking'
'StartTime'	'0.0'
'StopTime'	'inf'
'FixedStep'	'auto'
'SaveTime'	'off'
'SaveOutput'	'off'
'AlgebraicLoopMsg'	'error'
'InvariantConstants'	'on'

The default settings for 'SaveTime', 'SaveOutput', and 'InvariantConstants' 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 DSP 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 Link for ModelSim blocks. Gather and save test bench data that you can use later for comparing the model with a version that includes Link for ModelSim blocks.

## Starting ModelSim for Use with Simulink

The options available for starting ModelSim for use with Simulink vary depending on whether you run ModelSim and Simulink on the same computer system.

If both tools are running on the same system, start ModelSim directly from MATLAB by calling the MATLAB function `vsim`. This function starts and configures the ModelSim simulator (`vsim`) for use with the Link for ModelSim. 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 for each ModelSim start.

You can customize the DO file and communication mode to be used between Simulink and ModelSim by specifying the call to `vsim` with property name/property value pairs.

---

**Note** The following options may have been set previously with a call to `setupmodelsim`. To check on current settings, search for and browse through the contents of the file `\tcl\modelsim\tclFunctionsForMATLAB.tcl` in your ModelSim installation path. Any options that you explicitly specify with the MATLAB `vsim` function override these default settings.

---

### To...

Include one or more Tcl commands in the DO file that are to execute during ModelSim startup

### Specify...

'`tclstart`', '`tcl_commands`', where `tcl_commands` is a command string or cell array of command strings, which can include the `matlabtb` and `matlabtbeval` ModelSim commands that initialize the simulator for a test bench session (see "Initializing the Simulator for a MATLAB Test Bench Session" on page 6–16)

<b>To...</b>	<b>Specify...</b>
Start a specific version of the simulator that is not on the system path	'vsimdir', 'pathname', where pathname identifies the path and file name for the version of the simulator executable you want to start
Create a ModelSim startup file for future use (for example, test scripts)	'startupfile', 'pathname', where pathname specifies a path and filename for the generated DO file
Specify default TCP/IP socket communication for the link between Simulink and ModelSim	'socketsimulink', 'tcp_spec', where tcp_spec specifies a socket port a TCP/IP socket port number or service name. For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports” on page 1–17.
Specify shared memory communication for the link between ModelSim and Simulink on a single computer	No 'socketsimulink' property. Shared memory is the default mode of communication and takes effect if you omit 'socketsimulink' from the function call.

---

### Notes

- The `vsim` function applies the specified communication mode to all invocations of Simulink from ModelSim.
- The `vsim` function overrides any options previously defined by the `setupmodelsim` function.
- To start ModelSim from MATLAB with a default configuration previously defined by `setupmodelsim`, issue the command `!vsim` at the MATLAB command prompt.

---

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')
```

If ModelSim is running on a remote computer system,

- 1 Identify a valid and available socket port on the system that is running ModelSim.
- 2 Execute the MATLAB `vsim` function on the system running MATLAB and Simulink. In the function call, specify
  - `'tclstart'` with a Tcl command string that includes a `vsimulink` command that specifies the socket port identified in step 1.
  - `'startupfile'` with the name of the DO file that is to include the Tcl startup commands.
  - `'socketsimulink'` with the socket port number or service name identified in step 1.

For example:

```
vsim('tclstart', 'vsimulink work.parse', 'startupfile',
'simulinkstart.do', 'socketsimulink', '4449')
```

- 3 Copy the generated DO file to the system that is running ModelSim. For example, based on the preceding `vsim` command, you would copy the file `simulinkstart.do`.
- 4 From an operating system prompt, enter the generated DO file with the `vsim` command and `-do` option. For example

```
vsim -do simulinkstart.do
```



## Loading a VHDL Entity for Cosimulation

After you start ModelSim from MATLAB with a call to `vsim`, load an instance of a VHDL entity for cosimulation with the ModelSim command `vsimulink`. Issue the command for each instance of an entity in your model that you want to cosimulate. For example:

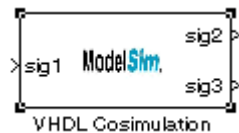
```
vsimulink work.manchester
```

This command opens a simulation workspace for `manchester` and displays a series of messages in the ModelSim command window as the simulator loads the entity's packages and architectures.

## Adding the VHDL Representation of a Model Component into a Simulink Model

After you code one of your model's components in VHDL and simulate it in the ModelSim environment, integrate the VHDL representation into your Simulink model as a VHDL Cosimulation block:

- 1 Open your Simulink model, if it is not already open.
- 2 Delete the model component that the VHDL Cosimulation block is to replace.
- 3 In the Simulink Library Browser, click the Link for ModelSim library. The browser displays three block icons.



VHDL  
Cosimulation

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



VHDL Sink

Block that has no output ports. Inherits data types of input ports from driving blocks. Typically, a sink analyzes signals.



VHDL Source

Block that has no input ports. Typically, generates signals that are imported into a model. Specifies an output sample time parameter.

---

### Notes

- The VHDL Sink and VHDL Source icons in the Link for ModelSim block library are provided for convenience only and map directly to the VHDL Cosimulation block.
  - The Link for ModelSim library also includes a To VCD File block. For information on using this block, see “Using a Value Change Dump File for Design Verification” on page 7–43.
- 
- 4 Copy one of the three icons from the Library Browser to your model. Simulink creates a link to the block at the point where you drop the block icon. For modeling sink and source devices, alternatively you can use the VHDL Cosimulation block directly, and configure the block appropriately for that type of device.
  - 5 Connect any VHDL block ports to appropriate blocks in your Simulink model.

## Configuring a VHDL Cosimulation Block

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

- Ports — Block input and output ports that correspond to signals, including internal signals, of your VHDL design, and an output sample time
- Comm — Type of communication and communication settings to be used for exchanging data between simulators
- Clocks — Rising-edge and falling-edge clocks to apply to your model
- Tcl — Tcl commands that you want to run before and after a simulation

The following sections help you identify what you need to configure, how to open the **Block Parameter** dialog, and how to configure each pane.

### What Are Your VHDL Cosimulation Block Requirements?

Before you start to configure a VHDL Cosimulation block, review the following checklist. The checklist will help you identify the parameters you need to set. If your answer to a question is something other than “no,” go to the topic listed in the second column of the table for information on how to adjust the parameter setting to meet your block requirements.

#### VHDL Cosimulation Block Requirements Checklist

Requirement	For More Information, See...
<b>Ports</b>	
<input type="checkbox"/> Does the VHDL model you are mapping to Simulink receive signals on input ports? If so, what are the input ports?	“Mapping VHDL Signals to Block Ports” on page 7–29
<input type="checkbox"/> Does the VHDL model you are mapping to Simulink transmit signals to output ports? If so, what are the output ports?	“Mapping VHDL Signals to Block Ports” on page 7–29

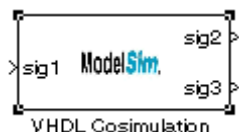
Requirement	For More Information, See...
<input type="checkbox"/> If the block is modeling an input and output device, do you want to specify an explicit output sample time for output ports? By default, if you specify both input and output ports, the block inherits the fastest sample time from its driver.	“Mapping VHDL Signals to Block Ports” on page 7–29
<input type="checkbox"/> If the block is modeling a source device, do you want to specify an output sample time other than two clock ticks? If you do not specify an input port, the block uses a default sample time of two clock ticks.	“Mapping VHDL Signals to Block Ports” on page 7–29
<b>Communication</b>	
<input type="checkbox"/> Is it critical that communication performance be as optimal as possible?	“Configuring the Communication Link” on page 7–33
<input type="checkbox"/> Are you running ModelSim and Simulink on the same computer?	“Configuring the Communication Link” on page 7–33
<input type="checkbox"/> If ModelSim and Simulink are running on the same computer, do you want to use shared memory communication?	“Configuring the Communication Link” on page 7–33
<input type="checkbox"/> Do you want to choose a TCP/IP socket port? If so, what port number or service will you use to establish a link?	“Configuring the Communication Link” on page 7–33
<input type="checkbox"/> If you are running ModelSim and Simulink different computers, what is the host name of the computer running ModelSim?	“Configuring the Communication Link” on page 7–33
<b>Clocks</b>	
<input type="checkbox"/> Do you want to create a rising-edge clock to apply stimuli to your cosimulation model?	“Creating Optional Clocks” on page 7–35
<input type="checkbox"/> Do you want to create a falling-edge clock to apply stimuli to your cosimulation model?	“Creating Optional Clocks” on page 7–35
<b>Tcl</b>	

Requirement	For More Information, See...
<input type="checkbox"/> Are there any Tcl commands that you want ModelSim to execute before running a simulation, but after loading the project in ModelSim?	“Specifying Before and After Simulation Tcl Commands” on page 7–37
<input type="checkbox"/> Are there any Tcl commands that you want ModelSim to execute after running a simulation?	“Specifying Before and After Simulation Tcl Commands” on page 7–37

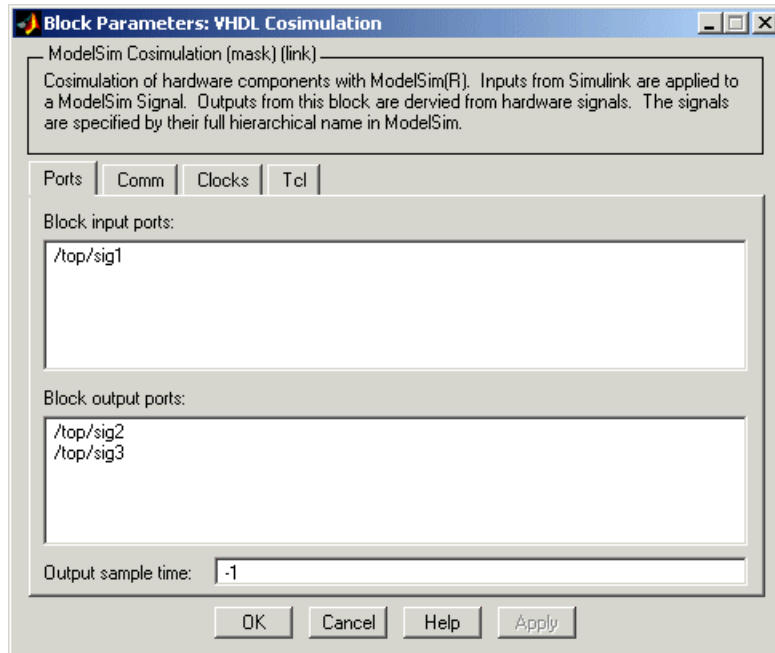
---

### Opening the Block Parameters Dialog

To open a block parameters dialog, double-click the block icon. For example, to open the block parameters dialog for the VHDL Cosimulation block, double-click



Simulink displays the following **Block Parameters: VHDL Cosimulation** dialog.

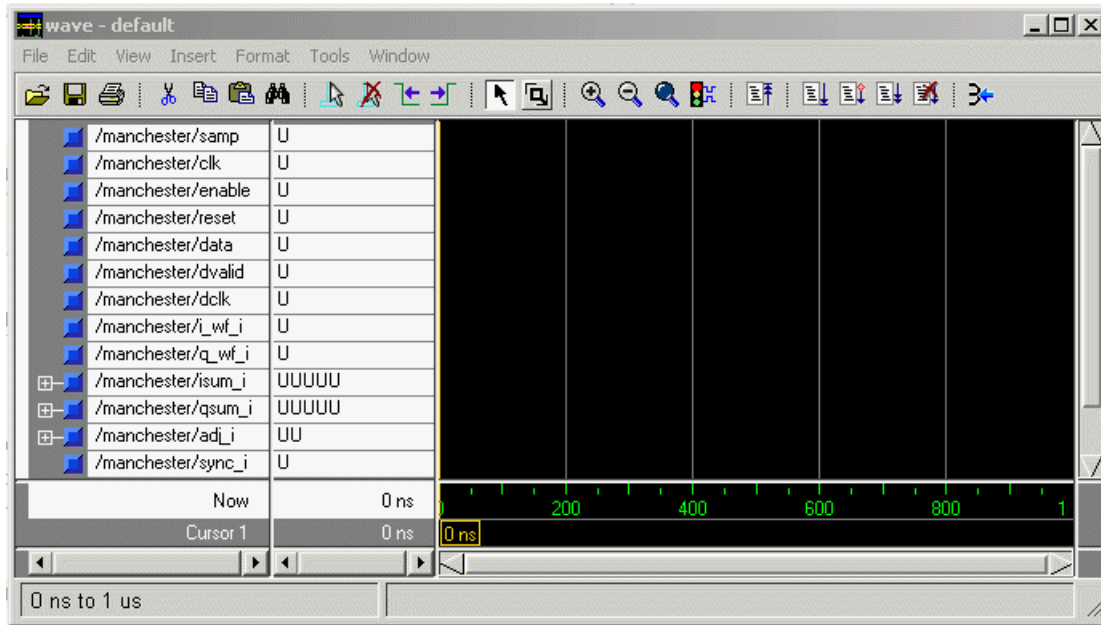


## Mapping VHDL Signals to Block Ports

The first step to configuring your Link for ModelSim block is to map signals and signal instances of your VHDL design to port definitions in your block. In addition to identifying input and output ports, you can specify a sample time for all output ports. VHDL Cosimulation block input ports inherit sample times from source signals.

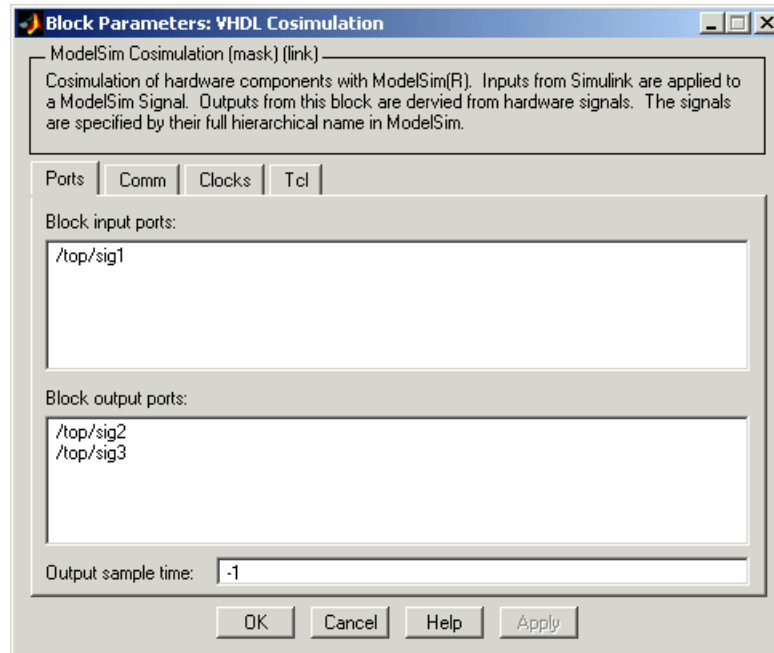
The signals that you map can be at any level of the VHDL design hierarchy. To map the signals,

- 1 In ModelSim, determine the test signal pathnames for the VHDL signals you plan to define in your block. The ModelSim signal pathname feature allows you to visualize and specify the hierarchy of signals in a VHDL design. One way of displaying the pathnames is to view the test signals in the pathname pane of the **wave** window with the full pathname option enabled. For example, the following display shows all signals are subordinate to the top-level entity manchester.



- 2 In Simulink, open the block parameters dialog for your VHDL Cosimulation block, if it is not already open.
- 3 Click the **Ports** tab of the **Block Parameters** dialog. Simulink displays the dialog as shown below.



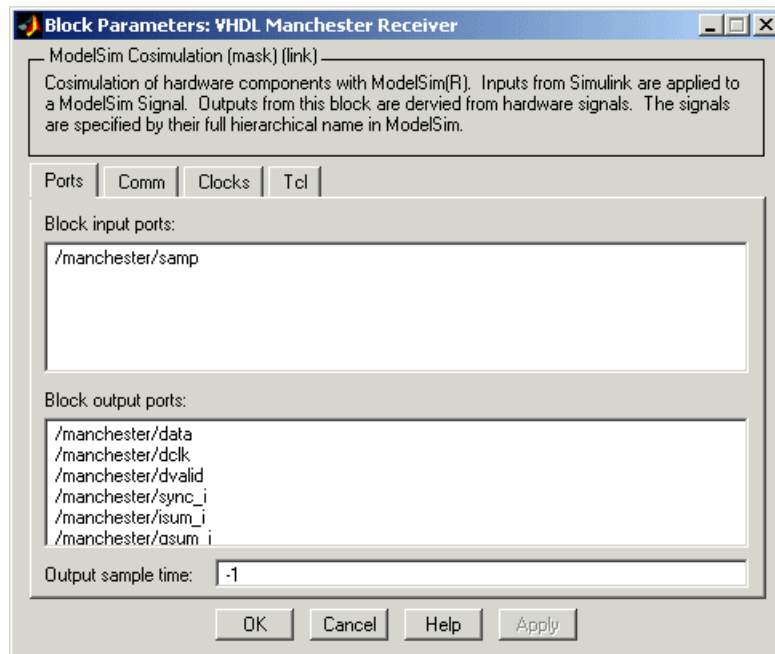


In this tab, you define the VHDL signals of your design that you want to include in your Simulink block and set a sample time for output ports. The parameters that you should specify on the **Ports** tab depend on the type of device the block is modeling.

For...	Specify...
An input and output device	Block input ports, block output ports, and an output sample time; a default sample time of -1 causes the output ports to inherit the sample time of the signal source
A sink device	Block input ports
A source device	Block output ports and an output sample time; a default sample time of 2 causes the output ports to apply a sample time of two ticks of the ModelSim resolution limit

- 4 Enter test signal pathnames of interest in the **Block input ports** and **Block output ports** text fields. Use the ModelSim pathname syntax and enter one pathname per line. You can include the ModelSim simulator prefix `sim:`, but it is not required.

The following dialog display shows port definitions for a VHDL Cosimulation block. Note the signal pathnames match pathnames that appear in the ModelSim **wave** window shown in step 1.



---

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

---

- 5 If you are configuring a VHDL Cosimulation block to model an input and output device and the output rate is different than the input rate, specify a sample time for the output ports. If the output and input rates

are the same, apply the default of -1. When specifying an explicit output rate, specify an integer. Simulink uses the value that you specify and the current ModelSim resolution limit to calculate an actual simulation sample time, which accounts for simulation timing differences between the cosimulation applications. The equation that Simulink uses to derive the actual sample time follows:

$$\text{actual sample time} = \frac{\text{specified Simulink sample time}}{\text{ModelSim resolution limit}}$$

The default output sample time is different for each type of device the block might be modeling.

For...	The Default Is...
Input and output device	-1, which indicates that the block is to inherit the sample time of the input signal.
Source device	Two ticks of the ModelSim resolution limit

For more information on sample times in the Link for ModelSim environment, see “Representation of Simulation Time” on page 7–9.

#### 6 Click **Apply**.

## Configuring the Communication Link

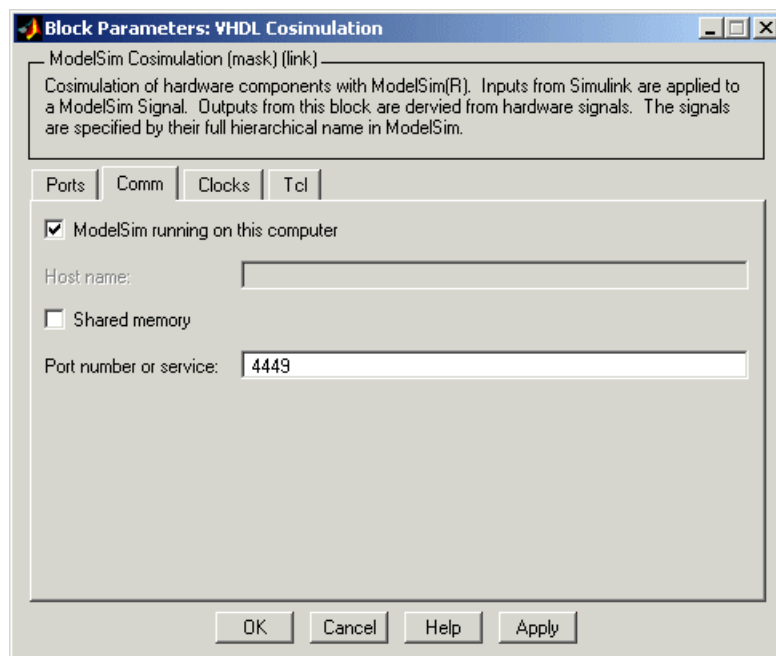
Configure a block’s communication link with the **Comm** tab of the block parameters dialog.

The following steps guide you through the communication configuration. The figure that follows shows the steps in a flow diagram:

- 1 Determine whether Simulink and ModelSim are running on the same computer. If they are, skip to step 4.
- 2 Clear the **ModelSim running on this computer** check box. This check box is selected by default.
- 3 In the **Host name** text field, specify the host name of the computer that is running your VHDL simulation in ModelSim. Skip to step 6.

- 4 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 “Modes of Communication” on page 1–8. If you choose TCP/IP sockets, skip to step 6.
- 5 Select the **Shared memory** check box and skip step 7.
- 6 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 1–17.
- 7 Click **Apply**.

The following dialog display shows communication definitions for a VHDL Cosimulation block.



The preceding sample **Comm** tab display specifies that

- Simulink and ModelSim are running on the same computer.

- TCP/IP socket mode of communication is used.
- TCP/IP port 4449 is used for the TCP/IP connection.

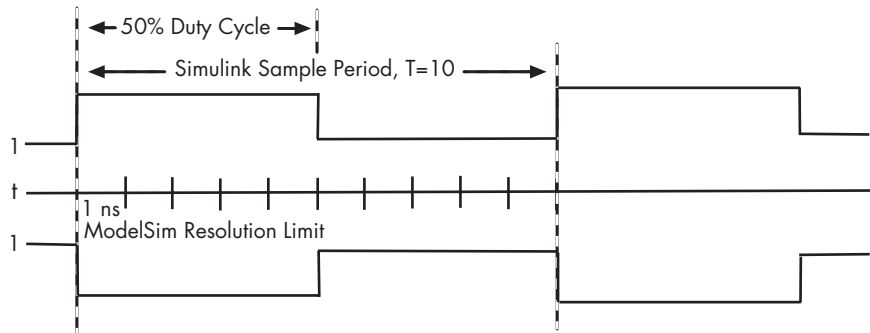
## Creating Optional Clocks

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 VHDL 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 some 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 a ModelSim resolution limit of 1 ns. The figure also shows that given those timing parameters, the clock duty cycle is 50%.

Rising Edge Clock

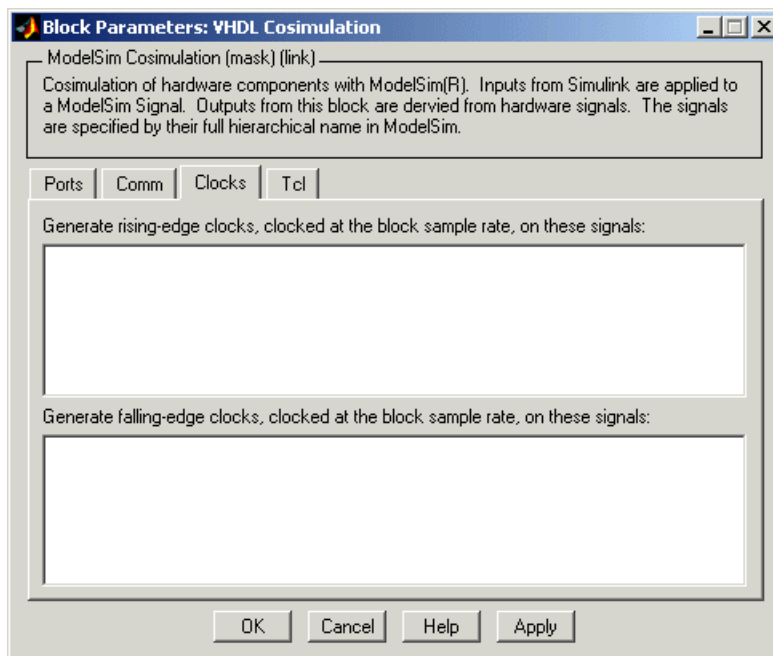


Falling Edge Clock

To create clocks,

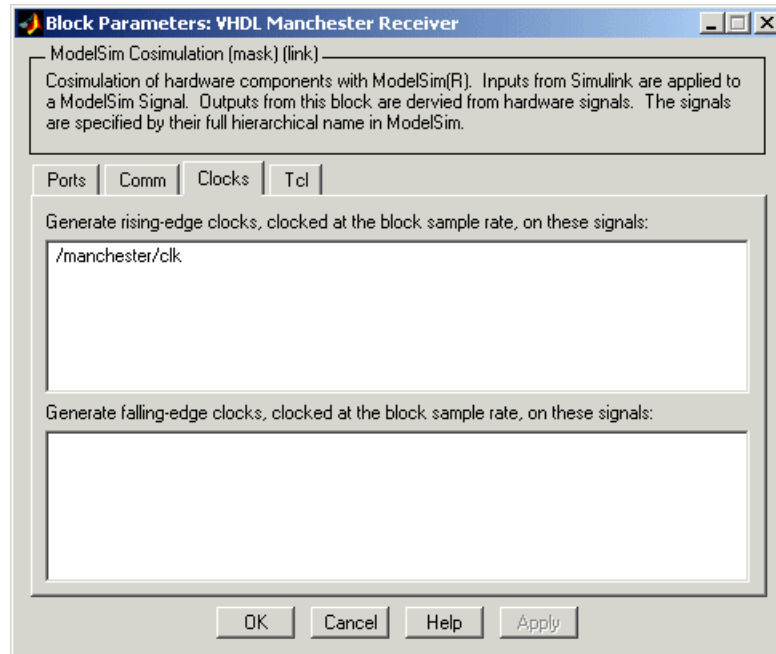
- 1 In ModelSim, determine the clock signal pathnames you plan to define in your block. To do this, you can use the same method explained for determining the signal pathnames for ports in step 1 of “Mapping VHDL Signals to Block Ports” on page 7–29.

- 2 Click the **Clocks** tab of the **Block Parameters** dialog. Simulink displays the dialog as shown below.



- 3 Enter clock signal pathnames of interest in the **Rising-edge clocks** and **Falling-edge clocks** text fields. Use the ModelSim pathname syntax and enter one pathname per line.

The following dialog display defines rising-edge clock `c1k` for the VHDL Cosimulation block.



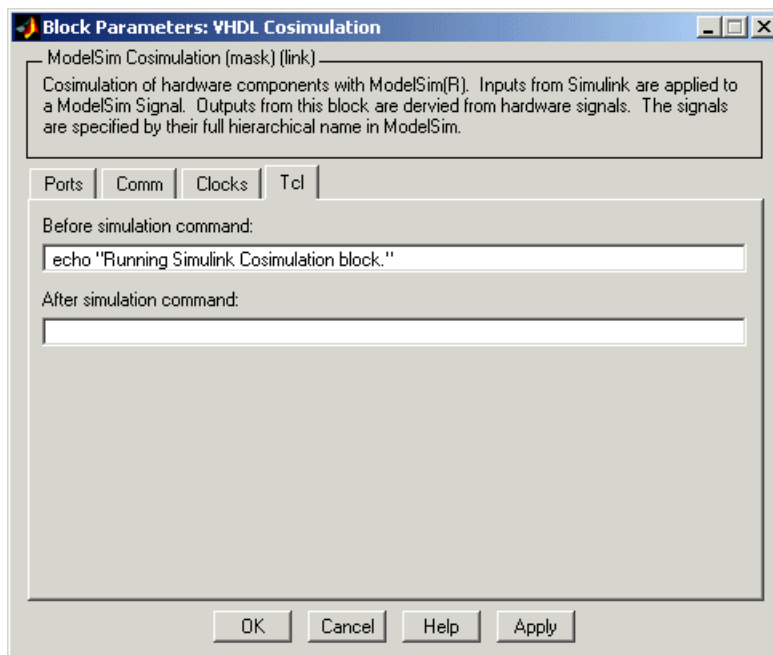
- 4 Click **Apply**.

## Specifying Before and After Simulation Tcl Commands

You have the option of specifying Tcl commands to execute before and after ModelSim simulates the VHDL component of your Simulink model. Tcl is a programmable scripting language supported by the ModelSim simulation environment. Use of Tcl can range from something as simple as a one-line echo command to confirm that a simulation is running or as complete as a complex script that performs an extensive simulation initialization and startup sequence. The **After simulation command** field is particularly useful for restarting ModelSim at the end of a simulation run.

To specify Tcl commands,

- 1 Click the **Tcl** tab of the **Block Parameters** dialog. The dialog display appears as follows.



The **Before simulation command** text box includes an echo command for reference purposes.

- 2 Enter one or more commands in the **Before simulation command** and **After simulation command** text boxes. If you enter multiple commands in a text box, separate them with a blank space. *Do not* separate commands with a carriage return.

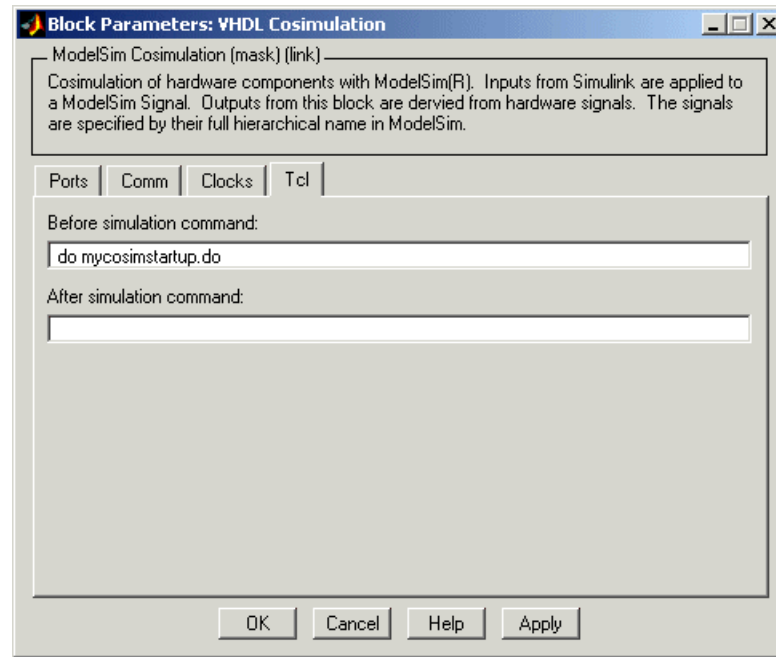


---

## 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 for either **Before simulation command** or **After simulation command** cannot include commands that load a ModelSim project or modify simulator state. For example, they cannot include commands such as `start`, `stop`, or `restart`.
- 

Alternatively, you can create a ModelSim DO file that lists Tcl commands and then specify that file with the ModelSim `do` command as shown below.



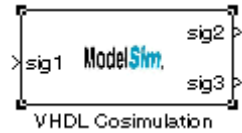
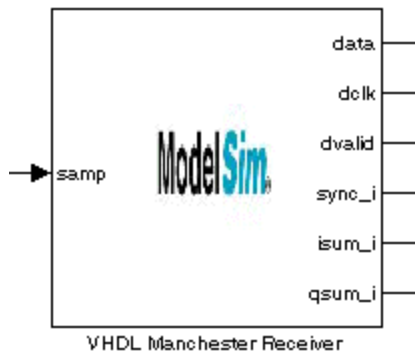
- 3 Click **Apply**.

### **Applying Your Block Parameters Configuration Settings and Closing the Dialog**

After you enter your block parameters settings,


- 1 Review the content of each dialog tab.
- 2 When you are satisfied with the dialog content, click **Apply** to apply any new settings.
- 3 Click **OK** to dismiss the dialog window.

Simulink applies the parameter settings and updates the VHDL Cosimulation block display to include specified input and output ports. For example:

**Before Configuration:****After Configuration:**

To verify the connection with ModelSim and the signal names, click **Edit->Update diagram** or press **Ctrl+D**.

## Running and Testing a Cosimulation Model in Simulink

To run and test a cosimulation model in Simulink, click **Simulation->Start** or the Start simulation tool  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 ModelSim force commands at the ModelSim command prompt or by specifying ModelSim force commands in the **After simulation command** text field on the **Tcl** tab of your Link for ModelSim block's parameters dialog.

## Using a Value Change Dump File for Design Verification

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

- 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 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.

The To VCD File block provided in the Link for ModelSim block library serves as a VCD file generator during a ModelSim and Simulink cosimulation session. 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 filename.

---

**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'.

---

The following sections discuss

- “Generating a VCD File” on page 7–44
- “VCD File Format” on page 7–45
- “A Sample VCD File Application ” on page 7–48

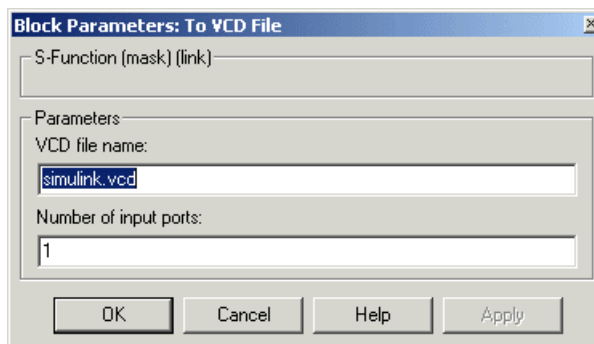
## Generating a VCD File

To generate a VCD file:

- 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 Link for ModelSim library. The browser displays four types of blocks, one of which is the To VCD File block.



- 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.
- 6 Configure the To VCD File block by specifying values for parameters in the Block Parameters dialog.
  - a Open the Block Parameters dialog by double-clicking the block icon. Simulink displays the following dialog.



- b Specify a filename for the generated VCD file in the **VCD file name** text box. If you specify a filename only, Simulink places the file in your current MATLAB directory. Specify a complete pathname 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 uniquely identify each instance.

---

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

---

- 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) bits, 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 is necessary when the input port receives a vector of real numbers or a fixed-point real number. For example, a signal of type `sfix16_En15` requires 16 symbols.

- d Click **OK**.

- 7 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 7–45. For a sample application of a VCD file, see “A Sample VCD File Application” on page 7–48.

## VCD File Format

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

File Content	Description
<pre>\$date 23-Sep-2003 14:38:11 \$end</pre>	Data and time the file was generated.

<b>File Content</b>	<b>Description</b>
<code>\$version Link for ModelSim version 1.0 \$ end</code>	Version of the VCD block that generated the file.
<code>\$timescale 1 ns \$ end</code>	The time scale that was used during the simulation.
<code>\$scope module manchestermodel \$end</code>	The scope of the module being dumped.
<code>\$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</code>	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.
<code>\$upscope \$end</code>	Marks a change to the next higher level in the HDL design hierarchy.
<code>\$enddefinitions \$end</code>	Marks the end of the header and definitions section.
<code>#0</code>	Simulation start time.



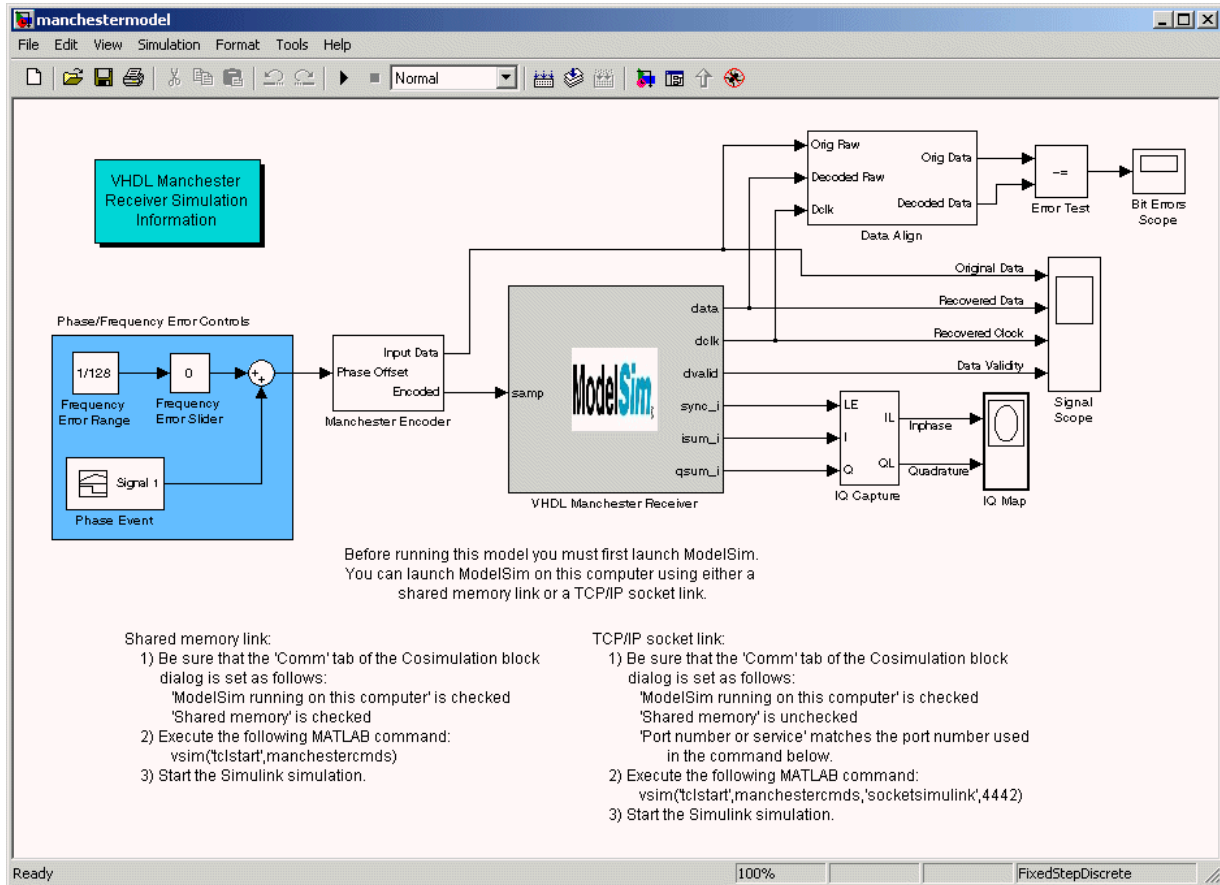
File Content	Description
\$dumpvars 0! 0" 0# 0\$ \$end	Lists the values of all defined variables at time equals 0.
#630 1!	The starting point of logged value changes. Variable values are checked at each simulation time increment and are logged if a change occurs. This entry indicates that at 63 nanoseconds, the value of signal Original Data changed from 0 to 1.
. . . #1160 1# 1\$	At 116 nanoseconds the values of signals Recovered Data and Data Validity changed from 0 to 1.
\$dumpoff x! x" x# x\$ \$end	Marks the end of the file by dumping the values of all variables as the value x.

VCD files can grow very large for larger designs or smaller designs with longer simulation runs. 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).

## A Sample VCD File Application

VCD files include data that can be graphically displayed or analyzed with postprocessing tools. An example of such a tool is the ModelSim `vcd2w1f` 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 `vcd2w1f` tool.

- 1 Place a copy of the Manchester Receiver Simulink demo `manchestermodel.mdl` in a writable directory.
- 2 Open your writable copy of the Manchester Receiver model. For example, click **File**→**Open**, select the file `manchestermodel.mdl` and click **Open**. The Simulink model should appear as follows.



**3** Open the Library Browser.

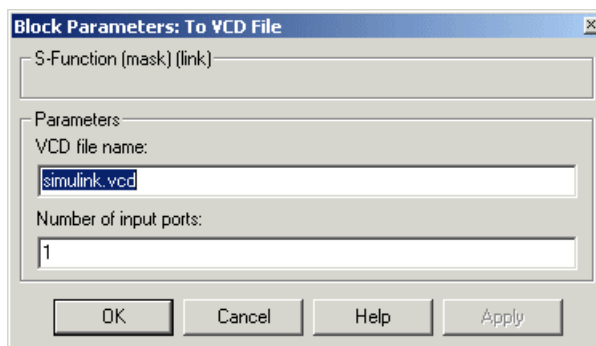
**4** Replace the Signal Scope block with a To VCD File block.

**a** Delete the Signal Scope block. The lines representing the signal connections to that block change to red dashed lines, indicating the disconnection.

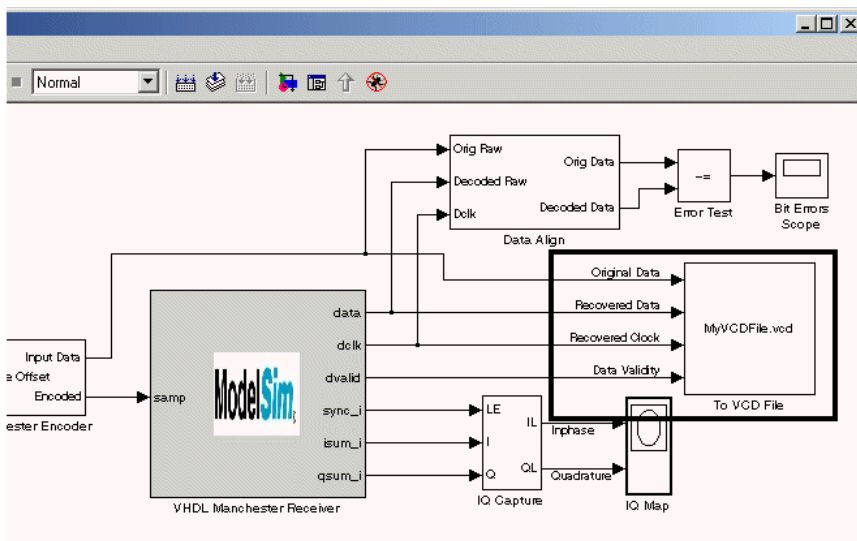
**b** Find and open the Link for ModelSim block library.

**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 VHDL Cosimulation block. To check the port setting for that block, double click the block icon and then click the **Comm** 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 dataset `MyVCDwlf` by entering the following command:

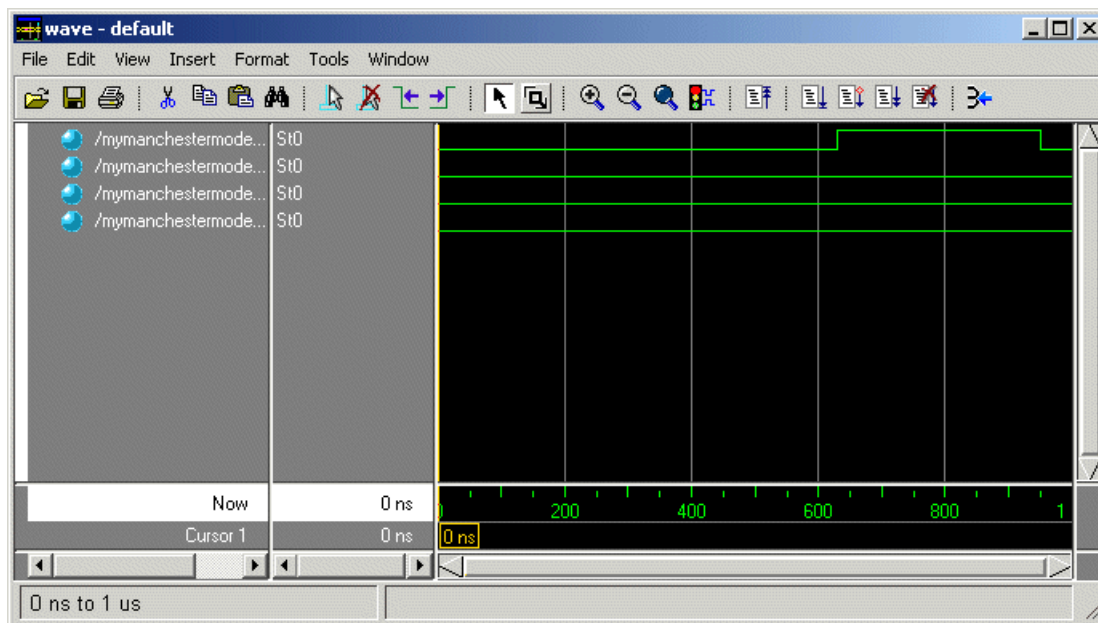
```
vsim -view MyVCDfile.wlf
```

- 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 icon  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 `vcd2wlf` utility and working with data sets, see the ModelSim documentation.

# MATLAB Functions — Alphabetical List

---

# hdldaemon

---

**Purpose** Start the MATLAB server component of the Link for ModelSim interface

**Syntax**

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

**Description** **Server Activation**

hdldaemon starts the MATLAB server component of the Link for ModelSim with the following default settings:

- Shared memory communication enabled
- Time resolution for the MATLAB simulation function output ports set to scaled (type double)

Use shared memory communication when your application configuration consists of a single system.

---

**Note** 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 `matlabtb` or `matlabtbeval` command in ModelSim. 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 ModelSim command must match the socket being used by the server. For more information on modes of communication, see “Modes of Communication” on page 1–8. For more information on establishing the ModelSim end of the communication link, see “Initializing the Simulator for a MATLAB Test Bench Session” on page 6–16.

---



`hdldaemon('PropertyName', 'PropertyValue'...)` starts the MATLAB server component of the Link for ModelSim with property-value pair settings that specify the mode of the communication for the link between MATLAB and ModelSim and the time resolution for the MATLAB simulation function output ports. See Property Name/Property Value Pairs for details.

### Link Status

`hdldaemon('status')` returns the following message indicating that a link (connection) exists between MATLAB and ModelSim:

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

You can also use this function to check on the mode of communication being used, the number of existing connections, and that an 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 ModelSim 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.

### Server Shutdown

`hdldaemon('kill')` Shuts down the MATLAB server without shutting down MATLAB.

# hdldaemon

---

## Property Name/Property Value Pairs

'socket', tcp\_spec

Specifies the TCP/IP socket mode of communication for the link between MATLAB and ModelSim. 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. Some valid tcp\_spec examples follow:

Option	Examples
Port number	'4449' or 4449
Alias or service name	'MATLAB Service'
Operating system assigned	'0' or 0

For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports” on page 1–17.

---

**Note** If you specify the operating system option ('0' or 0), use hlddaemon('status') to acquire the assigned socket port number. You must specify this port number when you issue a link request with the matlabbt or matlabbtbeval command in ModelSim.

---

'time', 'sec' | 'time', 'int64'

Specifies the time resolution for MATLAB function output ports and simulation times (tnow).

Specify...	For...
'time' 'sec' (default)	A double value that is scaled to seconds based on the current ModelSim simulation resolution
'time' 'int64'	64-bit integer representing the number of simulation steps

If you omit this argument, the server uses scaled resolution time.

## 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 output ports.:

```
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')
```

# setupmodelsim

---

**Purpose** Configure or deconfigure ModelSim for use with MATLAB and Simulink

**Syntax**  
`setupmodelsim`  
`setupmodelsim('PropertyName', 'PropertyValue'...)`

**Description** `setupmodelsim` starts an interactive installation script that configures ModelSim for use with the MATLAB and Simulink features of the Link for ModelSim.

`setupmodelsim('PropertyName', 'PropertyValue'...)` starts an interactive or programmatic script that configures or deconfigures ModelSim for use with the MATLAB and Simulink. If you specify only the 'action' property, the script runs in interactive mode.

The function modifies an installed version of ModelSim such that it subsequently starts ready to use the Link for ModelSim based on settings of property name/value pairs that specify

- Whether the function configures or deconfigures ModelSim
- Tcl commands to start ModelSim
- The `vsim` executable

After you call this function, you can use Link for ModelSim commands from the ModelSim environment to

- Load instances of VHDL entities for simulations that use MATLAB or Simulink for verification or cosimulation
- Initiate MATLAB test bench sessions for loaded instances
- Terminate MATLAB test bench sessions
- Apply a VHDL wrapper around Verilog modules to be compiled and used with Link for ModelSim (see the `wrapverilog` command)

<b>Property Name/Property Value Pairs</b>	<p>'action', 'install'          Configures ModelSim for use with the MATLAB and Simulink by modifying an installed version of ModelSim such that it subsequently starts ready to use the Link for ModelSim.</p> <p>'action', 'uninstall'          Deconfigures ModelSim for use with the MATLAB and Simulink.</p> <p>'tclstart', 'tcl_commands'          Specifies one or more Tcl commands to execute during ModelSim startup. Specify a command string or a cell array of command strings that is to be appended to the ModelSim startup file.</p> <p>'vsimdir', 'pathname'          Specifies the pathname 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.</p>
---	---

**Examples**

The following function call starts the interactive installation script that configures ModelSim for use with the MATLAB and Simulink:

```
setupmodelsim
```

The following function call configures ModelSim such that it subsequently starts ready for use with MATLAB and Simulink. Based on the specified property data, ModelSim starts vsim from its default executable and creates a temporary DO file in a temporary directory for the Link for ModelSim commands. The Link for ModelSim commands are specified with the 'tclstart' property and include

- A vsimmatlab command that loads an instance of the VHDL entity parse in library work for MATLAB verification.
- A matlabtb command that initiates 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 nanoseconds.

## setupmodelsim

---

```
setupmodelsim('action','install','tclstart','vsimmatlab  
work.parse; matlabbtb parse 10 ns -socket 4449')
```

**Purpose** Start and configure ModelSim for use with the Link for ModelSim

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

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

After you call this function, you can use ModelSim commands to

- Load instances of VHDL entities for simulations that use MATLAB for verification
- Load instances of VHDL entities for simulations that use Simulink for cosimulation
- Apply a VHDL wrapper around Verilog modules compiled and used with Link for ModelSim (see the `wrapverilog` command)

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.

**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 pathname 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
```

'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. The following table lists different ways of specifying `tcp_spec`.

<b>Format</b>	<b>Example</b>
<port-num>	4449
<port-alias>	matlabservice
<port-num>@<host>	4449@compa
<host>:<port-num>	compa:4449
<port-alias>@<host-ia>	matlabservice@123.34.55.23

For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports” on page 1–17

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.

---

## 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 library `work` for MATLAB verification.
- The `matlabtb` command initiates 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 nanoseconds.

```
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 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:

## **vsim**

---

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

# ModelSim Commands — Alphabetical List

---

# matlabtb

---

<b>Purpose</b>	Initiate a MATLAB test bench session for an instance of a VHDL entity based on specified test bench stimuli
<b>Syntax</b>	<pre>matlabtb &lt;instance&gt; [&lt;time-specs&gt;] [-socket &lt;tcp-spec&gt;]     [-rising &lt;port&gt;[,&lt;port&gt;...]] [-falling &lt;port&gt;[,&lt;port&gt;,...]]     [-sensitivity &lt;port&gt;[,&lt;port&gt;,...]] [-mfunc &lt;name&gt;]</pre>
<b>Arguments</b>	<p><b>&lt;instance&gt;</b> Specifies the instance of a VHDL entity that attaches to a MATLAB function. By default, matlabtb attaches the instance to 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.</p> <p><b>&lt;time-specs&gt;</b> Specifies a combination of time specifications consisting of any or all of the following:</p>

- `<timen>,...` 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.
- `-repeat <time>` Specifies that the MATLAB function be called repeatedly based on the specified `<timen>,...` pattern. The time values are relative to the value of `tnow` at the time the MATLAB function is initially called.
- `-cancel <time>` Specifies a time at which the specified MATLAB function stops executing. The time value is relative to the value of `tnow` at the time the MATLAB function is initially called. If you do not specify a cancel time, the command calls the MATLAB function.
- `-socket <tcp_spec>`  
 Specifies TCP/IP socket communication for the link between ModelSim and MATLAB. 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 (alias). If you are setting up communication between computing systems, you must also specify the name or Internet address of the remote host that is running the MATLAB server (`hdldaemon`). 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

Format	Example
<port-num>@<host>	4449@compa
<host>:<port-num>	compa:4449
<port-alias>@<host-ia>	matlabservice@123.34.55.23

For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports” on page 1–17.

If ModelSim and MATLAB 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 `-ocket <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 “Modes of Communication” on page 1–8. For more information on establishing the MATLAB end of the communication link, see “Starting the MATLAB Server” on page 6–7.

---

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

Indicates that the specified MATLAB function is called when any of the specified signals experiences a rising edge — changes from '0' to '1'. Specify `-rising` with the pathnames of one or more signals defined as a logic type — `std_logic`, `bit`, `x01`, and so on.

`-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 pathnames of one or more signals defined as a logic type — `std_logic`, `bit`, `x01`, and so on.

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

Indicates that the specified MATLAB function is called when any of the specified signals changes state. Specify sensitivity with the pathnames of one or more signals. Signals in the sensitivity list can be any type and can be at any level of the VHDL hierarchy.

-mfunc <name>

The name of the MATLAB function that is attached to the entity you specify for instance. If you omit this argument, matlabtb attaches the entity to a MATLAB function that has the same name as the entity. For example, if the entity is myfirfilter, matlabtb associates the entity 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,

- 1 Starts the ModelSim client component of the Link for ModelSim.
- 2 Associates a specified instance of a VHDL entity created in ModelSim with a MATLAB MATLAB function.
- 3 Creates a process that schedules invocations of the specified MATLAB function.

---

**Note** For ModelSim 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.

This command 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.

---

## Examples

The following command starts the ModelSim client component of the Link for ModelSim, associates an instance of the entity `myfirfilter` with the MATLAB function `myfirfilter`, and initiates 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 (when `t` equals 0, 5, 15, 25, and so on).

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

The following command starts the ModelSim client component of the Link for ModelSim, and initiates 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
```



**Purpose** Initiate an immediate call to a MATLAB function on behalf of an instance of a VHDL entity

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

**Arguments** `<instance>`  
 Specifies the instance of a VHDL entity that attaches to a MATLAB function. By default, `matlabtbeval` attaches the instance to a MATLAB function that has the same name as the instance. For example, if the instance is `myfirfilter`, `matlabtbeval` associates the instance with the MATLAB function `myfirfilter`. Alternatively, you can specify a different MATLAB function with the `-mfunc` property.

`-socket <tcp_spec>`  
 Specifies TCP/IP socket communication for the link between ModelSim and MATLAB. 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 (alias). If you are setting up communication between computers, 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 on this computer
<code>&lt;port-alias&gt;</code>	matlabservice on this computer
<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

For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports” on page 1–17.

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 `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 modes of communication, see “Modes of Communication” on page 1–8. For more information on establishing the MATLAB end of the communication link, see “Starting the MATLAB Server” on page 6–7.

---

`-mfunc <name>`

The name of the MATLAB function that is attached to the entity you specify for instance. If you omit this argument, `matlabtbeval` attaches the entity to a MATLAB function that has the same name as the entity. For example, if the entity is `myfirfilter`, `matlabtbeval` associates the entity 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,

- 1 Starts the ModelSim client component of the Link for ModelSim.
- 2 Associates a specified instance of a VHDL entity created in ModelSim with a MATLAB function.
- 3 Executes the specified MATLAB function once and immediately on behalf of the specified entity instance.

---

**Note** For ModelSim 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

The following command starts the ModelSim client component of the Link for ModelSim, associates an instance of the entity `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`.

```
vsim> matlabtbeval myfirfilter -socket 4449
```

The following command starts the ModelSim client component of the Link for ModelSim, associates an instance of the entity `filter` with the function `myfirfilter.m`, and uses a remote TCP/IP socket-based communication link to host `compb` and TCP/IP port 4449 to execute the function `myfirfilter.m`.

```
vsim> matlabtbeval myfirfilter -socket 4449@compa
```

# nomatlabtb

---

<b>Purpose</b>	Terminate a MATLAB test bench session that was initiated with <code>matlabtb</code>
<b>Syntax</b>	<code>nomatlabtb</code>
<b>Description</b>	The <code>unmatlabtb</code> command terminates all active test bench sessions.
<b>Examples</b>	The following command terminates all test bench sessions: <pre>vsim&gt; nomatlabtb</pre>

---

<b>Purpose</b>	Load an instance of a VHDL entity for verification with MATLAB
<b>Syntax</b>	<code>vsimmatlab &lt;instance&gt; [&lt;vsim_args&gt;]</code>
<b>Arguments</b>	<p><code>&lt;instance&gt;</code> Specifies the instance of a VHDL entity to load for verification.</p> <p><code>&lt;vsim_args&gt;</code> Specifies one or more <code>vsim</code> command arguments. For details, see the description of <code>vsim</code> in the ModelSim documentation.</p>
<b>Description</b>	The <code>vsimmatlab</code> command loads the specified instance of an entity 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 entity's packages and architectures.
<b>Examples</b>	<p>The following command loads the entity instance <code>parse</code> from library <code>work</code> for verification and sets up ModelSim so it can establish a communication link with MATLAB.</p> <pre>ModelSim&gt; vsimmatlab work.parse</pre>

# vsimulink

---

**Purpose** Load an instance of a VHDL entity for cosimulation with Simulink

**Syntax** vsimulink <instance> [-socket <tcp\_spec>]

**Argument** <instance>  
Specifies the instance of a VHDL entity 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 `matlab-service`.

For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports” on page 1–17.

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 Link for ModelSim blocks in your Simulink model. For more information on modes of communication, see “Modes of Communication” on page 1–8. For more information on establishing the Simulink end of the communication link, see “Configuring the Communication Link” on page 7–33.

---

**Description** The `vsimulink` command loads the specified instance of an entity 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 entity's packages and architectures.

**Examples**

The following command loads the entity 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

---

<b>Purpose</b>	Apply a VHDL wrapper to a Verilog module
<b>Syntax</b>	<code>wrapverilog &lt;verilog_module&gt; [-nocompile]</code>
<b>Argument</b>	<code>&lt;verilog_module&gt;</code> 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.  <code>-nocompile</code> Suppresses automatic compilation of the resulting VHDL file, <code>verilog_module_wrap.vhd</code> .
<b>Description</b>	The <code>wrapverilog</code> 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 the Link for ModelSim.
<b>Examples</b>	The following command applies a VHDL wrapper to Verilog module <code>myverilogmod.v</code> and writes the output to <code>myverilogmod_wrap.vhd</code> . The <code>-nocompile</code> option suppresses automatic compilation.  <code>ModelSim&gt; wrapverilog myverilogmod.v -nocompile</code>



# Simulink Blocks — Alphabetical List

---

# To VCD File

---

**Purpose** Generate a value change dump (VCD) file

**Library** Link for ModelSim

**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 filename. VCD files can be very useful during design verification. Some examples of how you might apply VCD files include

- 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.

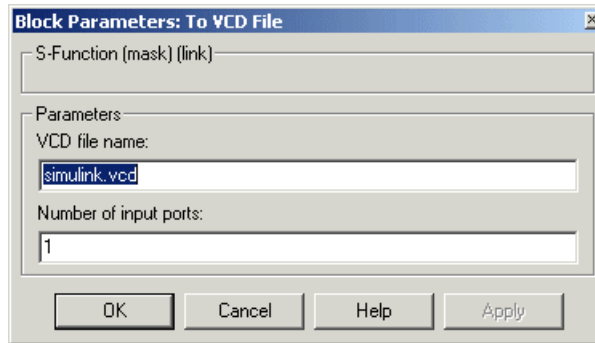
Using the Block Parameters dialog, you can specify the following:

- The filename to be used for the generated file
- The number of block input ports that are to receive signal data

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). Each bit maps to one symbol.

For a description of the VCD file format see “VCD File Format” on page 7–45.

## Dialog Box



### VCD file name

The filename to be used for the generated VCD file. If you specify a filename only, Simulink places the file in your current MATLAB directory. Specify a complete pathname 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 uniquely identify each instance.

---

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

---

### 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 one of the following:

- Vector of real numbers
- Fixed-point real number

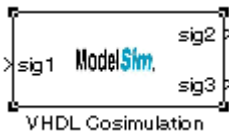
# VHDL Cosimulation

---

**Purpose** Cosimulate a hardware component by applying input signals to and reading output signals from a VHDL model under simulation in ModelSim

**Library** Link for ModelSim

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



---

**Note** The VHDL Sink and VHDL Source icons in the Link for ModelSim block library are provided for convenience only and map directly to the VHDL Cosimulation block.

---

Using tabbed panels on the block's dialog box, you can configure the following:

- Block input and output ports that correspond to signals, including internal signals, of a VHDL model and an output sample time
- Type of communication and communication settings to be used for exchanging data between simulators
- Rising-edge or falling-edge clocks to apply to your model
- Tcl commands that you want to run before and after the simulation

The **Ports** tab provides fields for mapping signals of your VHDL design to input and output ports in your block. The signals that you map can be at any level of the VHDL design hierarchy. Simulink deposits an input port signal on a ModelSim signal at the signal's sample rate. Conversely, Simulink reads an output port signal from a specified ModelSim signal at the Simulink 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 a user-specified output sample time to all output ports. If you do not specify an output sample time, Simulink applies the fastest input sample period to all output ports.

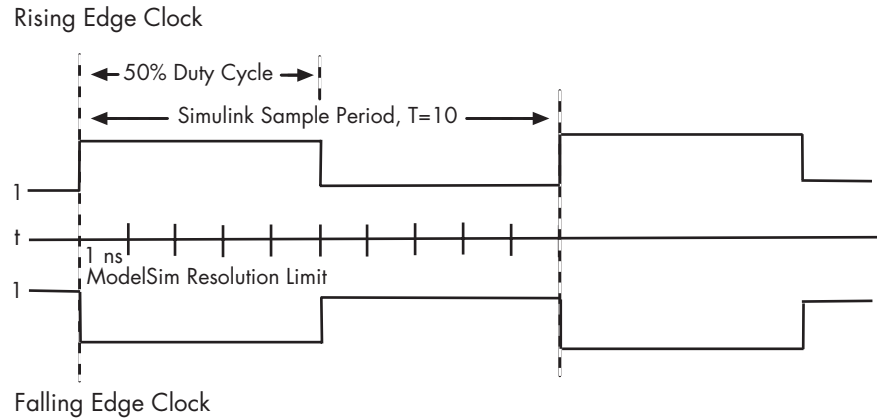
The **Comm** tab specifies the mode of communication to be used between Simulink and ModelSim. If you use TCP socket communication, this tab also provides fields for specifying a socket port and, for remote communication, the host name of the computer running ModelSim.

You can create optional rising-edge and falling-edge clocks that apply stimuli to your cosimulation model with the **Clocks** tab. 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, as you approach the resolution limit of the ModelSim simulation, with a worst case duty cycle of 66% for a sample period of  $T=3$ .

The following figure shows a timing diagram that includes rising-edge and falling-edge clocks with a Simulink sample period of  $T=10$  and a ModelSim resolution limit of 1 ns. The figure also shows that given those timing parameters, the clock duty cycle is 50%.

# VHDL Cosimulation

---



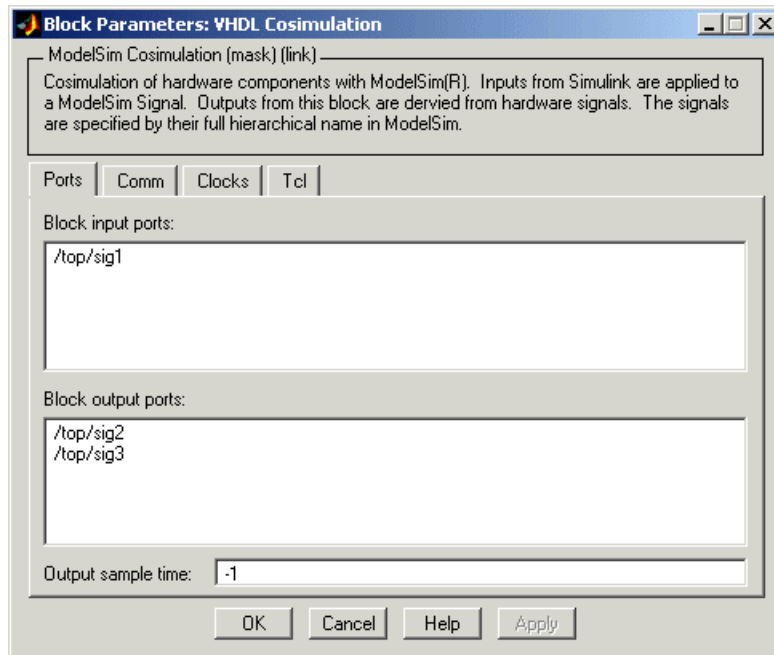
The **Tcl** tab provides a way of specifying tools command language (Tcl) commands to be executed before and after ModelSim simulates the VHDL component of your Simulink model. The before command field on this tab is particularly useful for simulation initialization and startup operations, but cannot be used to change simulation state.

## Dialog Box

The **Block Parameters** dialog box consists of four tabbed panes of configuration options:

- Ports Tab
- Comm Tab
- Clocks Tab
- Tcl Tab

## Ports Tab



### Block input ports

Signals of your VHDL model that are to be driven by Simulink. Simulink deposits values on the specified ModelSim signal at the signal's sample rate.

---

**Note** Leave this field blank if you are using the block to model a source device.

---

Specify each port of interest as a test signal pathname, using ModelSim pathname syntax, and enter one pathname per line. A sample pathname for an input port might be /manchester/samp.

# VHDL Cosimulation

---

The ports that you map can be at any level of the VHDL design hierarchy.

---

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

---

## **Block output ports**

Signals of your VHDL model that are to be read by Simulink. Simulink reads an output port signal from the specified ModelSim signal at the Simulink signal's sample rate. You can specify a sample rate or -1 in the **Output sample time** text field. The value -1 instructs the block to inherit the output sample rate from the input rate.

---

**Note** Leave this field blank to configure a cosimulation sink block.

---

Specify each port of interest as a test signal pathname, using ModelSim pathname syntax, and enter one pathname per line. A sample pathname for an output port might be `/manchester/data`.

The ports that you map can be at any level of the VHDL design hierarchy.

## **Output sample time**

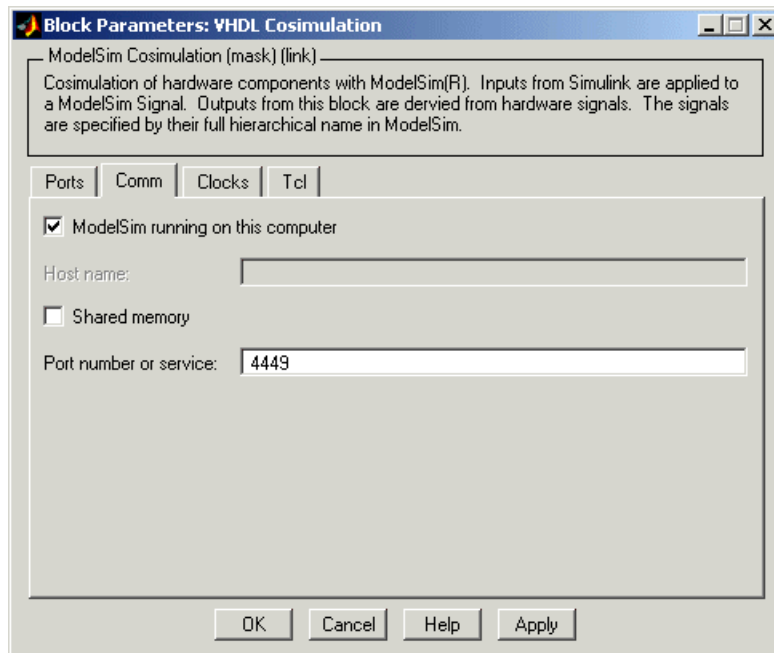
The time interval (ticks) between consecutive samples applied to all output ports. The resolution of the time interval is equal to the simulator resolution that is set for ModelSim. To determine the resolution, at the ModelSim prompt, enter `echo $resolution` or



report simulator state. The default for this field depends on how you are using the block.

If the Block Has...	The Default Is...
Input and output ports	-1, inherit the sample time of the signal source
Input ports only (sink)	0, has no impact
Output ports only (source)	2

## Comm Tab



**ModelSim running on this computer**

If selected, the block configuration assumes Simulink and ModelSim are running on the same computer. When both applications run on the same computer, you have the option of using shared memory or TCP sockets for the communication channel between the two applications.

**Host name**

If Simulink and ModelSim are running on different computers, this text field specifies the host name of the computer that is running your VHDL simulation in ModelSim.

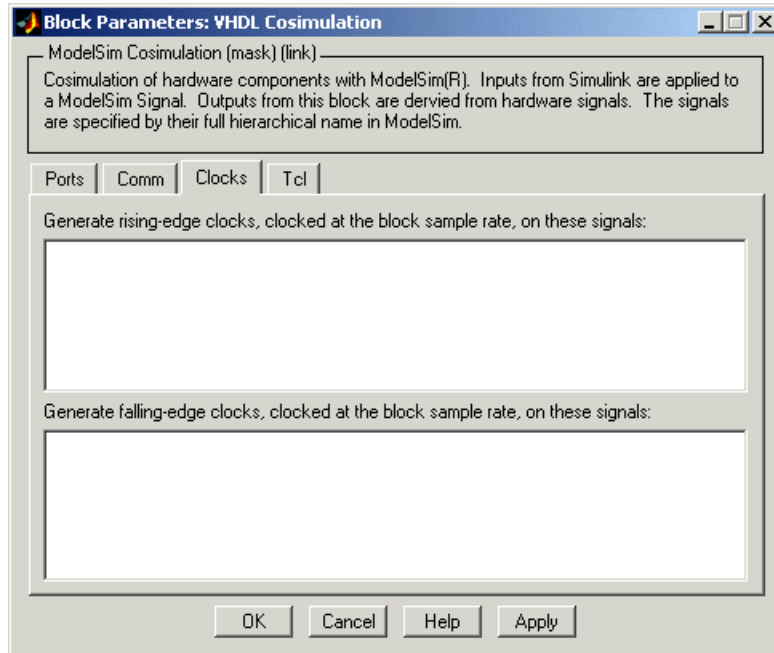
**Shared memory**

If selected, Simulink and ModelSim use the shared memory for communication. To select this option, you must also select **ModelSim running on this computer**. For more information on modes of communication, see “Modes of Communication” on page 1–8.

**Port number or service**

A valid port number or service for your computer system. For information on choosing TCP socket ports, see “Choosing TCP/IP Socket Ports” on page 1–17.

## Clocks Tab



### Rising-edge clocks

One or more rising-edge clocks that drive values to the VHDL signals that you are modeling, using the deposit method. Specify each clock as a signal pathname, using ModelSim pathname syntax, and enter one pathname per line. A sample pathname for a clock might be `/manchester/clk`. You can include the ModelSim simulator prefix `sim:`, but it is not required.

### Falling-edge clocks

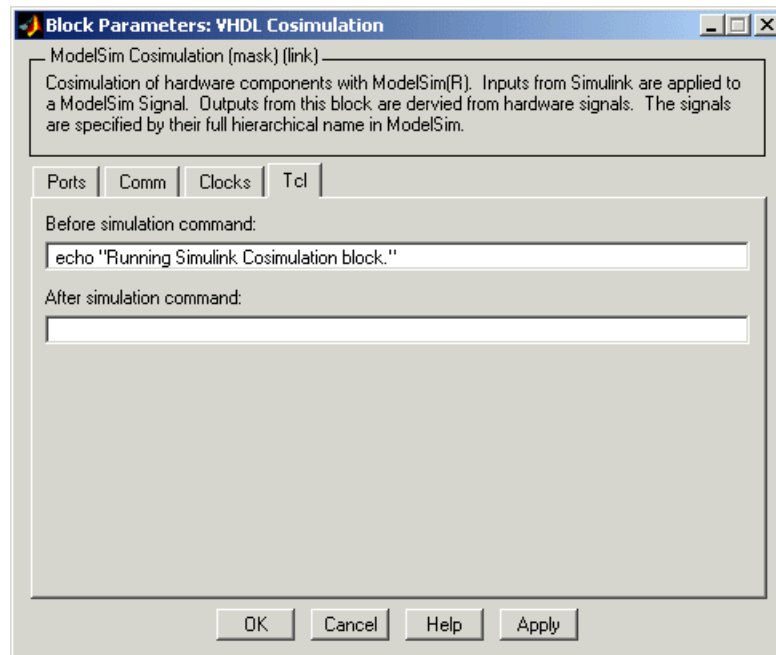
One or more falling-edge clocks that drive deposit values to the VHDL signals that you are modeling. Specify each clock as a signal pathname, using ModelSim pathname syntax, and enter one pathname per line. A sample pathname for a clock might

# VHDL Cosimulation

---

be `/manchester/c1k`. You can include the ModelSim simulator prefix `sim:`, but it is not required.

## Tcl Tab



### Before simulation command

A Tcl command line to be executed before ModelSim simulates the VHDL component of your Simulink model. You can specify multiple commands 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.

---

### **After simulation command**

A Tcl command line to be executed before ModelSim simulates the VHDL component of your Simulink model. You can specify multiple commands 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`.
-

## A

- action property
  - description of 8–6
  - setting up ModelSim during installation with 1–21
- addresses, Internet 1–17
- After simulation command parameter
  - description of 10–4
  - specifying block Tcl commands with 7–37
- application software 1–18
- application specific integrated circuits (ASICs) 1–2
- applications 1–3
  - coding Link for ModelSim 5–1
    - overview of 5–2
  - programming Link for ModelSim 5–1
    - overview of 5–2
- arguments
  - for matlabbtb command 9–2
  - for matlabbtbeval command 9–7
  - for vsimmatlab command 9–11
  - for vsimulink command 9–12
  - for wrapverilog command 9–14
- arrays
  - converting to 5–18
  - indexing elements of 5–9
  - of VHDL data types 5–5
- ASICs (application specific integrated circuits) 1–2

## B

- Before simulation command parameter
  - description of 10–4
  - specifying block simulation Tcl commands with 7–37
- behavioral model 1–3
- BIT data type 5–5
  - conversion of 5–9
  - converting to 5–18

- bit vectors
  - converting for MATLAB 5–17
  - converting to 5–18
- BIT\_VECTOR data type 5–5
  - conversion of 5–9
  - converting for MATLAB 5–17
  - converting to 5–18
- Block input ports parameter
  - description of 10–4
  - mapping signals with 7–29
- block latency 7–11
- block library
  - description of 7–24
  - Link for ModelSim 1–5
- Block output ports parameter
  - description of 10–4
  - mapping signals with 7–29
- Block Parameters dialog
  - for To VCD File block 7–44
  - for VHDL Cosimulation block 7–28
- block ports
  - mapping signals to 7–29
  - requirements for VHDL Cosimulation blocks 7–26
- blocks
  - To VCD File
    - configuring 7–44
    - description of 10–2
    - generating VCD files with 7–44
  - VHDL Cosimulation
    - applying configuration settings for 7–40
    - configuring 7–26
    - description of 10–4
- blocksets
  - for creating hardware models 7–5
  - for EDA applications 7–5
  - installing 1–19
- bp ModelSim command 6–22
- Break button, ModelSim 6–22

Break option, ModelSim 6–22  
breakpoints 6–22

## C

callback specification 5–13

callback timing 6–14

-cancel option 9–2

CHARACTER data type 5–5

conversion of 5–9

checklists

environment requirements 1–12

VHDL Cosimulation block  
requirements 7–26

client

for MATLAB and ModelSim links 1–5

for Simulink and ModelSim links 1–7

client/server environment 1–5

clocks

requirements for VHDL Cosimulation  
blocks 7–26

specifying for VHDL Cosimulation  
blocks 7–35

Clocks tab

configuring block clocks with 7–35

description of 10–4

column-major numbering 5–9

comm status field

checking with `hdldaemon` function 6–5

description of 8–2

Comm tab

configuring block communication  
with 7–33

description of 10–4

commands, ModelSim 9–1

*See also* ModelSim commands

communication

configuring for blocks 7–33

features 1–5

initializing for ModelSim and MATLAB  
session 6–16

modes of 1–8

requirements for VHDL Cosimulation  
blocks 7–26

socket ports for 1–17

communication channel

checking identifier for 6–5

communication modes

checking 6–5

specifying for Simulink links 7–20

specifying for VHDL Cosimulation  
block 7–26

specifying with `hdldaemon` function 6–7

Communications Blockset

as optional software 1–18

using for EDA applications 7–5

compilation, VHDL code 5–7

compiler, VHDL 5–7

components 1–5

composite data types

conversions of 5–9

VHDL 5–5

configuration information 1–23

configurations

deciding on 1–15

multiple-link 1–15

single-system 1–15

valid for MATLAB and ModelSim 1–15

valid for Simulink and ModelSim 1–16

connections status field

checking with `hdldaemon` function 6–5

description of 8–2

connections, link

checking number of 6–5

TCP/IP socket 1–17

Continue button, MATLAB 6–22

Continue option 6–22

continuous signals 7–9

convolution 4–6



- convolver, I/Q
  - function for 4-19
  - VHDL code for 4-11
- cosimulation 1-5
  - configuring a VHDL Cosimulation block for 7-26
  - controlling MATLAB 6-1
    - overview of 6-3
  - loading VHDL entities for 7-23
  - logging changes to signal values during 7-43
  - requirements for 7-26
  - running Simulink and ModelSim tutorial 3-14
  - shutting down Simulink and ModelSim tutorial 3-17
  - starting MATLAB 6-1
    - overview of 6-3
  - starting with Simulink 7-42
- cosimulation block 7-26
  - See also* VHDL Cosimulation block
- cosimulation environment 1-5

## D

- data types
  - conversions of 5-9
  - converting for MATLAB 5-17
  - converting for ModelSim 5-18
  - unsupported VHDL 5-5
  - VHDL port 5-5
    - verifying 5-15
- dbstop function 6-22
- decoder
  - function for 4-23
  - script code for 4-31
  - VHDL code for 4-14
- delta time 7-11
- demos 1-25
  - Manchester receiver 1-32

- MATLAB and ModelSim 1-27
  - random number generator 1-27
  - Simulink and ModelSim 1-32
- deposit
  - changing signals with 7-8
  - for `iport` parameter 5-13
  - with `force` commands 6-21
- design process, hardware 1-3
- dialogs
  - for To VCD File block 10-2
  - for VHDL Cosimulation block 10-4
- discrete blocks 7-9
- do command 7-37
- DO files
  - saving startup commands to 1-22
  - specifying for VHDL Cosimulation blocks 7-37
- documentation overview 1-24
- double values
  - as representation of time 6-14
  - converting for MATLAB 5-17
  - converting for ModelSim 5-18
- DSP Blockset
  - as optional software 1-18
  - using for EDA applications 7-5
- `dspstartup` M-file 7-17
- duty cycle 7-35

## E

- EDA (Electronic Design Automation) 1-2
- Electronic Design Automation (EDA) 1-2
- encoding, Manchester 4-3
- End Simulation option, ModelSim 6-26
- entities
  - coding for MATLAB verification 5-3
  - compiling 4-17
  - for decoder 4-14
  - for I/Q convolver 4-11
  - for state counter 4-15

- getting port information of 5–13
- loading for cosimulation 3–13
- loading for cosimulation with Simulink 7–23
- loading for verification 6–12
- naming 5–3
- programming for MATLAB verification 5–3
- sample definition of 5–5
- specifying ports for 5–4
- using port information for 5–15
- validating 5–15
- verifying port direction modes for 5–15
- enumerated data types 5–5
  - conversion of 5–9
  - converting to 5–18
- environment requirements 1–12
- environment, cosimulation 1–5
- examples 7–5
  - See also* Manchester receiver Simulink model
  - hdldaemon function 8–2
  - Manchester receiver 4–1
  - MATLAB and ModelSim 2–1 4–1
  - matlabtb command 9–2
  - matlabtbbeval command 9–7
  - nomatlabtb command 9–10
  - setupmodelsim function 8–6
  - Simulink and ModelSim 3–1
  - test bench function 5–22
  - VCD file generation 7–48
  - vsim function 8–9
  - vsimmatlab command 9–11
  - vsimulink command 9–12
  - wrapverilog command 9–14

## F

- falling option 9–2
  - specifying scheduling options with 6–16

- falling-edge clocks
  - creating for VHDL Cosimulation blocks 7–35
  - specifying as scheduling options 6–13
  - specifying for VHDL Cosimulation block 7–26
- Falling-edge clocks parameter
  - description of 10–4
  - specifying block clocks with 7–35
- features, product 1–5
- field programmable gate arrays (FPGAs) 1–2
- files
  - generating VCD 7–44
  - VCD 7–45
- Fixed-Point Blockset
  - as optional software 1–18
  - using for EDA applications 7–5
- force command
  - applying simulation stimuli with 6–21
  - resetting clocks during cosimulation with 7–42
- FPGAs (field programmable gate arrays) 1–2
- functions 8–1
  - See also* MATLAB functions
  - resolution 7–8

## G

- Go Until Cursor option, MATLAB 6–22
- gold reference designs 1–4

## H

- hardware definition language (HDL) 1–2
- hardware design process 1–3
- hardware model design
  - creating in Simulink 7–5
  - running and testing in Simulink 7–19
- HDL (hardware definition language) 1–2
- HDL models 1–3 5–7

*See also* Verilog models; VHDL models;  
 VHDL models  
 cosimulation of 1–3  
 verification of 1–3  
 hdldaemon function  
   checking link status of 6–5  
   configuration restrictions for 1–15  
   description of 8–2  
   starting 6–7  
 help 1–24  
 Host name parameter  
   description of 10–4  
   specifying block communication with 7–33  
 host names  
   identifying MATLAB server 6–16  
   identifying ModelSim server 7–33  
   identifying server with 1–17  
  
**I**  
 I/Q convolver  
   function for 4–19  
   script code for 4–34  
   VHDL code for 4–11  
 IN direction mode 5–4  
   verifying 5–15  
 INOUT direction mode 5–4  
   verifying 5–15  
 inphase convolution 4–6  
 input 5–4  
   *See also* input ports  
 input ports  
   attaching to signals 7–8  
   for test bench function 5–13  
   for VHDL model 5–4  
   mapping signals to 7–29  
   simulation time for 7–9  
   specifying block 7–26  
 installation 1–11  
   of Link for ModelSim 1–19

  of related software 1–19  
 int64 values 6–14  
 INTEGER data type 5–5  
   conversion of 5–9  
   converting to 5–18  
 Internet address 1–17  
   identifying server with 1–17  
   specifying 6–16  
 interprocess communication identifier 6–5  
 ipc\_id status field  
   checking with hdldaemon function 6–5  
   description of 8–2  
 iport parameter 5–13

## K

kill option  
   description of 8–2  
   shutting down MATLAB server with 2–21

## L

latency associated with 7–11  
 latency, block 7–11  
 Link for ModelSim  
   block library 1–5  
     using to add VHDL to Simulink  
     with 7–24  
   blocks 1–15  
     *See also* VHDL Cosimulation block  
   definition of 1–2  
   installing 1–19  
   setting up ModelSim for 1–19  
 link status  
   checking MATLAB server 6–5  
   function for acquiring 8–2  
 links  
   MATLAB and ModelSim 1–5  
   Simulink and ModelSim 1–7

**M**

Manchester encoding 4–3

Manchester receiver

background information on 4–5

compiling VHDL code for 4–17

running simulation for 4–40

test bench functions for 4–19

test bench script for 4–30

VHDL code for 4–9

MATLAB

as required software 1–18

in Link for ModelSim environment 1–5

installing 1–19

quitting 6–26

working with ModelSim links to 1–8

MATLAB data types

conversion of 5–9

MATLAB functions 8–1

coding for VHDL verification 5–8

dbstop 6–22

defining 5–13

for decoder 4–23

for I/Q convolver 4–19

for MATLAB and ModelSim tutorial 2–15

for state counter 4–26

hdldaemon 6–7

description of 8–2

modsimrand 1–27

naming 5–13

programming for VHDL verification 5–8

sample of 5–22

scheduling invocation of 6–13

setupmodelsim 7–20

description of 8–6

specifying required parameters for 5–13

test bench 1–5

vsim 7–20

description of 8–9

which 5–28

MATLAB functions,

adding to MATLAB search path 5–28

MATLAB search path 5–28

MATLAB server

checking link status with 6–5

configuration restrictions for 1–15

configurations for 1–15

function for invoking 1–5

identifying in a network

configuration 1–17

starting 6–7

starting for MATLAB and ModelSim

tutorial 2–4

starting from a script 4–31

matlabtb command

description of 9–2

initializing ModelSim for MATLAB

session 6–16

specifying scheduling options with 6–13

matlabtbeval command

description of 9–7

initializing ModelSim for MATLAB

session 6–16

specifying scheduling options with 6–13

-mfunc option

specifying test bench function with 6–16

with matlabtb command 9–2

with matlabtbeval command 9–7

models

compiling VHDL 5–7

debugging VHDL 5–7

for Simulink and ModelSim tutorial 3–6

ModelSim

as required software 1–18

deconfiguring 1–23

handling of signal values for 7–8

in Link for ModelSim environment 1–5

initializing for MATLAB session 6–16

installing 1–19

quitting 6–26

setting up during installation 1–19

- setting up for MATLAB and ModelSim tutorial 2-6
  - setting up for Simulink and ModelSim tutorial 3-12
  - simulation time for 7-9
  - specifying version of 6-10
  - starting for use with Simulink 7-20
  - starting from MATLAB 6-10
  - version of 1-21 to 1-22
  - working with MATLAB links to 1-8
  - working with Simulink links to 1-9
  - ModelSim commands**
    - bp 6-22
    - force
      - applying simulation stimuli with 6-21
      - resetting clocks during cosimulation with 7-42
    - matlabtb
      - description of 9-2
      - initializing ModelSim with 6-16
    - matlabtbeval
      - description of 9-7
      - initializing ModelSim with 6-16
    - nomatlabtb 9-10
    - report simulator state 7-9
    - restart 6-25
    - run 6-22
    - specifying scheduling options with 6-13
    - vcd2wlf 7-43
    - vsimmatlab
      - description of 9-11
      - loading VHDL entities for verification with 6-12
    - vsimulink
      - description of 9-12
      - loading VHDL entities for cosimulation with 7-23
    - wrapverilog 9-14
  - ModelSim Editor 2-8**
  - ModelSim running on this computer parameter**
    - description of 10-4
    - specifying block communication with 7-33
  - ModelSim setup program**
    - running in command line mode 1-22
    - running in interactive mode 1-21
  - modes**
    - communication 1-17
      - specifying with hdlldaemon function 6-7
    - of communication 1-8
    - port direction 5-4 5-15
  - modsimrand function 1-27
  - multirate signals 7-10
- N**
- names**
    - for test bench functions 5-13
    - for VHDL entities 5-3
    - shared memory communication
      - channel 6-5
      - verifying port 5-15
  - NATURAL data type 5-5
    - conversion of 5-9
    - converting to 5-18
  - network configuration 1-17
  - network environment 1-5
  - nocompile option 9-14
  - nomatlabtb command 9-10
  - Number of input ports parameter 10-2
    - configuring To VCD File block with 7-44
  - Number of output ports parameter
    - configuring To VCD File block with 7-44
    - description of 10-2
  - numeric data
    - converting for MATLAB 5-17
    - converting for ModelSim 5-18

**O**

- online help 1–24
- oport parameter 5–13
- options
  - for matlabbtb command 9–2
  - for matlabbtbeval command 9–7
  - for vsimulink command 9–12
  - for wrapverilog command 9–14
- kill 8–2
- property
  - with hdldaemon function 8–2
  - with setupmodelsim function 8–6
  - with vsim function 8–9
- status 8–2
- OUT direction mode 5–4
  - verifying 5–15
- output ports
  - for test bench function 5–13
  - for VHDL model 5–4
  - mapping signals to 7–29
  - simulation time for 7–9
  - specifying block 7–26
- Output sample time parameter
  - description of 10–4
  - specifying sample time with 7–29

**P**

- parameters
  - for To VCD File block 10–2
  - for VHDL Cosimulation block 10–4
  - required for test bench functions 5–13
- phase, clock 7–35
- platform support 1–5
  - required 1–18
- port names
  - verifying 5–15
- Port number or service parameter
  - description of 10–4
  - specifying block communication with 7–33

- port numbers 1–17
  - checking 6–5
  - specifying for MATLAB server 6–7
  - specifying for ModelSim 6–13
  - specifying with setupmodelsim function 1–22
- portinfo parameter 5–13
- portinfo structure 5–15
- ports
  - getting information about 5–13
  - specifying direction modes for 5–4
  - specifying for VHDL entities 5–4
  - specifying VHDL data types for 5–5
  - using information about 5–15
  - verifying data type of 5–15
  - verifying direction modes for 5–15
- Ports tab
  - configuring block ports with 7–29
  - description of 10–4
- ports, block
  - mapping signals to 7–29
  - requirements for 7–26
- postprocessing tools 7–43
- prerequisite knowledge 1–4
- properties
  - action
    - description of 8–6
    - setting up ModelSim with 1–21
  - for hdldaemon function 8–2
  - for setupmodelsim function 8–6
  - for starting MATLAB server 6–7
  - for starting ModelSim for use with Simulink 7–20
  - for vsim function 8–9
  - socket 8–2
  - socketsimulink 8–9
  - startupfile 8–9
  - tlclstart
    - with setupmodelsim function 8–6
    - with vsim function 8–9

- time
  - description of 8-2
- vsimdir
  - with setupmodelsim function 8-6
  - with vsim function 8-9
- property option
  - for hdldaemon function 8-2
  - for setupmodelsim function 8-6
  - for vsim function 8-9

## Q

- quadrature convolution 4-6

## R

- rate converter 7-10
- real data
  - converting for MATLAB 5-17
  - converting for ModelSim 5-18
- REAL data type 5-5
  - conversion of 5-9
  - converting to 5-18
- real values, as time 6-14
  - repeat option 9-2
    - specifying scheduling options with 6-16
- report simulator state ModelSim
  - command 7-9
- requirements
  - application software 1-18
  - checking product 1-18
  - environment 1-12
  - for VHDL Cosimulation block 7-26
  - platform 1-18
- resolution functions 7-8
- resolution limit 5-15
- Restart button, Modelsim 6-25
- restart ModelSim command 6-25
- Restart option, ModelSim 6-25
  - rising option 9-2

- specifying scheduling options with 6-16
- rising-edge clocks
  - creating for VHDL Cosimulation
    - blocks 7-35
  - specifying as scheduling options 6-13
  - specifying for VHDL Cosimulation
    - block 7-26
- Rising-edge clocks parameter
  - description of 10-4
  - specifying block clocks with 7-35
- run command 6-22
- Run Continue button, ModelSim 6-22
- Run option, MATLAB 6-22

## S

- sample periods 7-5
  - See also* sample times
- sample times 7-11
  - design decisions for 7-5
  - handling across simulation domains 7-8
  - specifying for block output ports 7-29
- Save and Run option, MATLAB 6-22
- scalar data types
  - conversions of 5-9
  - VHDL 5-5
- scheduling options 6-13
- script
  - ModelSim setup 1-19
  - test bench 4-30
- search path 5-28
- sensitivity lists 6-13
  - sensitivity option 9-2
    - specifying scheduling options with 6-16
- server activation 8-2
- server shutdown 8-2
- server, MATLAB
  - checking link status of MATLAB 6-5
  - for MATLAB and ModelSim links 1-5
  - for Simulink and ModelSim links 1-7

- identifying in a network
  - configuration 1-17
- starting for MATLAB and ModelSim
  - tutorial 2-4
- starting from a script 4-31
- starting MATLAB 6-7
- set/clear breakpoint button, MATLAB 6-22
- Set/Clear Breakpoint option, MATLAB 6-22
- setup 1-11
- setupmodelsim function
  - deconfiguring ModelSim with 1-23
  - description of 8-6
  - running in command line mode 1-22
  - running in interactive mode 1-21
  - starting ModelSim with 7-20
  - using during installation 1-19
  - using install option with 1-22
- shared memory communication 1-5 1-8
  - as a configuration option 1-15
  - for Simulink applications 7-20
  - specifying for VHDL Cosimulation blocks 7-33
  - specifying with hdldaemon function 6-7
- Shared memory parameter
  - description of 10-4
  - specifying block communication with 7-33
- signal pathnames
  - displaying 7-29
  - specifying for block clocks 7-35
  - specifying for block ports 7-29
- signals
  - continuous 7-9
  - defining ports for 5-4
  - driven by multiple sources 7-8
  - exchanging between simulation domains 7-8
  - handling across simulation domains 7-8
  - how Simulink drives 7-8
  - logging changes to 7-43
  - logging changes to values of 7-43
  - mapping to block ports 7-29
  - multirate 7-10
- signed data 5-17
- SIGNED data type 5-18
- simulation analysis 7-43
- simulation time 5-13
  - guidelines for 7-9
  - representation of 7-9
  - scaling of 7-9
- simulations
  - comparing results of 7-43
  - ending 6-26
  - loading for MATLAB and ModelSim
    - tutorial 2-12
  - logging changes to signal values
    - during 7-43
  - Manchester receiver 4-40
  - quitting 6-26
  - running for MATLAB and ModelSim
    - tutorial 2-18
  - running Simulink and ModelSim
    - tutorial 3-14
  - shutting down for MATLAB and ModelSim
    - tutorial 2-21
  - shutting down Simulink and ModelSim
    - tutorial 3-17
- simulator resolution limit 5-15
- simulators
  - handling of signal values between 7-8
  - ModelSim
    - initializing for MATLAB session 6-16
    - starting from MATLAB 6-10
- Simulink
  - as optional software 1-18
  - configuration restrictions for 1-15
  - configuring for VHDL models 7-17
  - creating hardware model designs with 7-5
  - driving cosimulation signals with 7-8
  - in Link for ModelSim environment 1-5
  - installing 1-19



- running and testing hardware model
  - in 7-19
- setting up ModelSim for use with 3-12
- simulation time for 7-9
- starting ModelSim for use with 7-20
- using with ModelSim 7-1
- working with ModelSim links to 1-9
- Simulink models
  - adding VHDL models to 7-24
  - for Simulink and ModelSim tutorial 3-6
- sink device
  - adding to a Simulink model 7-24
  - specifying block ports for 7-29
  - specifying clocks for 7-35
  - specifying communication for 7-33
  - specifying Tcl commands for 7-37
- socket numbers 6-5
  - See also* port numbers
- socket option
  - specifying TCP/IP socket with 6-16
  - with `matlabtb` command 9-2
  - with `matlabtbeval` command 9-7
  - with `vsimulink` command 9-12
- socket port numbers 1-17
  - as a networking requirement 1-17
  - checking 6-5
  - specifying for TCP/IP link 7-20
  - specifying for VHDL Cosimulation blocks 7-33
  - specifying with `-socket` option 6-16
  - specifying with `setupmodelsim` function 1-22
- socket property
  - description of 8-2
  - specifying with `hdldaemon` function 6-7
- sockets 1-8
  - See also* TCP/IP socket communication
- socketsimulink property
  - description of 8-9
  - specifying TCP/IP socket for ModelSim with 7-20
- software
  - installing 1-19
  - optional 1-18
  - required 1-18
- Solaris 1-5
  - as a required platform 1-18
- source device
  - adding to a Simulink model 7-24
  - specifying block ports for 7-29
  - specifying clocks for 7-35
  - specifying communication for 7-33
  - specifying Tcl commands for 7-37
- standard logic data 5-17
- standard logic vectors
  - converting for MATLAB 5-17
  - converting for ModelSim 5-18
- start time 7-9
- startup commands, ModelSim 6-10
- startupfile property
  - description of 8-9
  - specifying DO file for ModelSim startup with 7-20
  - specifying with `vsim` function 6-10
- state counter
  - function for 4-26
  - script code for 4-36
  - VHDL code for 4-15
- status option
  - checking value of 6-5
  - description of 8-2
- status, link 6-5
- STD\_LOGIC data type 5-5
  - conversion of 5-9
  - converting to 5-18
- STD\_LOGIC\_VECTOR data type 5-5
  - conversion of 5-9
  - converting for MATLAB 5-17
  - converting to 5-18

- STD\_ULONGIC data type 5–5
    - conversion of 5–9
    - converting to 5–18
  - STD\_ULONGIC\_VECTOR data type 5–5
    - conversion of 5–9
    - converting for MATLAB 5–17
    - converting to 5–18
  - Step button
    - in MATLAB 6–22
    - in ModelSim 6–22
  - Step option, ModelSim 6–22
  - Step Over button, ModelSim 6–22
  - Step-In button, MATLAB 6–22
  - Step-Out button, MATLAB 6–22
  - Step-Over option, ModelSim 6–22
  - stimuli, block internal 7–35
  - stop time 7–9
  - strings, time value 6–14
  - subtypes, VHDL 5–5
- T**
- Tcl commands
    - configuring for block simulation 7–37
    - configuring ModelSim to start with 7–20
    - requirements for VHDL Cosimulation
      - blocks 7–26
    - specifying for VHDL Cosimulation
      - block 7–26
    - specifying with `setupmodelsim` function 1–22
    - specifying with `vsim` function 6–10
  - Tcl tab
    - description of 10–4
  - tclstart property
    - specifying with `setupmodelsim` function 7–20
    - specifying with `vsim` function 6–10
    - with `setupmodelsim` function 8–6
    - with `vsim` function 8–9
  - TCP/IP networking protocol 1–8
    - See also* TCP/IP socket communication
    - as a networking requirement 1–17
  - TCP/IP socket communication
    - as a communication option 1–15
    - feature 1–5
    - for Simulink applications 7–20
    - mode 1–8
    - specifying with `hdldaemon` function 6–7
  - TCP/IP socket ports 1–17
    - specifying for VHDL Cosimulation
      - blocks 7–33
    - specifying with `-socket` option 6–16
  - test bench functions
    - adding to MATLAB search path 5–28
    - coding for VHDL verification 5–8
    - defining 5–13
    - for MATLAB and ModelSim tutorial 2–15
    - naming 5–13
    - programming for VHDL verification
      - overview of 5–8
      - sample of 5–22
    - scheduling invocation of 6–13
    - specifying required parameters for 5–13
  - test bench script
    - for decoder 4–31
    - for I/Q convolver 4–34
    - for state counter 4–36
    - Manchester receiver 4–30
    - starting MATLAB server from 4–31
  - test bench sessions
    - controlling 6–1 6–3
    - logging changes to signal values
      - during 7–43
    - monitoring 6–22
    - restarting 6–25
    - running 6–22
    - starting 6–1 6–3
    - stopping 6–26
  - test benches 1–5

*See also* test bench functions  
 time 7-9  
     *See also* time values  
     callback 5-13  
     delta 7-11  
     simulation 5-13  
         guidelines for 7-9  
         representation of 7-9  
 TIME data type 5-5  
     conversion of 5-9  
     converting to 5-18  
 time property  
     description of 8-2  
     setting return time type with 6-7  
 time scale, VCD file 7-45  
 time units 6-16  
 time values 6-16  
     specifying as scheduling options 6-13  
     specifying with `hdl_daemon` function 6-7  
 timing errors 7-9  
`tnext` parameter 5-13  
     controlling callback timing with 6-14  
     specifying as scheduling options 6-13  
     time representations for 6-14  
`tnow` parameter 5-13  
 To VCD File block 1-5  
     configuring 7-44  
     description of 10-2  
     generating VCD files with 7-44  
     uses of 1-9  
 tools, postprocessing 7-43  
`tyscale` parameter 5-15  
 tutorial files 2-3  
 tutorials 1-25  
     Manchester receiver 4-1  
     MATLAB and ModelSim 2-1  
     Simulink and ModelSim 3-1  
 types  
     conversions of 5-9  
     converting for MATLAB 5-17

    converting for ModelSim 5-18  
     VHDL port 5-5

## U

unsigned data 5-17  
 UNSIGNED data type 5-18  
 unsupported data types 5-5  
 users, Link for ModelSim 1-4

## V

value change dump (VCD) files 7-43  
     *See also* VCD files  
 VCD file name parameter  
     configuring To VCD File block with 7-44  
     description of 10-2  
 VCD files 1-5  
     example of generating 7-48  
     format of 7-45  
     generating 7-44  
     using 7-43  
`vcd2wlf` command 7-43  
 vectors  
     converting for MATLAB 5-17  
     converting to 5-18  
 verification  
     coding functions for 5-8  
         overview of 5-8  
     hardware model 1-5  
 verification sessions  
     logging changes to signal values  
         during 7-43  
     monitoring 6-22  
     restarting 6-25  
     running 6-22  
     stopping 6-26  
 VHDL code  
     compiling for MATLAB and ModelSim  
         tutorial 2-11

- compiling for Simulink and ModelSim
    - tutorial 3–4
  - for decoder 4–14
  - for I/Q convolver 4–11
  - for Manchester receiver 4–9
    - compiling 4–17
  - for MATLAB and ModelSim tutorial 2–8
  - for Simulink and ModelSim tutorial 3–2
  - for state counter 4–15
- VHDL Cosimulation block
- adding to a Simulink model 7–24
  - applying configuration settings for 7–40
  - black boxes representing 7–5
  - configuration requirements for 1–15
  - configuring 7–26
  - configuring clocks for 7–35
  - configuring communication for 7–33
  - configuring ports for 7–29
  - configuring Tcl commands for 7–37
  - description of 10–4
  - design decisions for 7–5
  - handling of signal values for 7–8
  - in Link for ModelSim environment 1–5
  - opening Block Parameters dialog for 7–28
  - scaling simulation time for 7–9
  - valid configurations for 1–16
- VHDL data types 1–5
- See also* data types
  - conversion of 5–9
- VHDL design 7–3
- VHDL entities
- coding for MATLAB verification 5–3
  - for Simulink and ModelSim tutorial
    - loading for cosimulation 3–13
  - getting port information of 5–13
  - loading for cosimulation with Simulink 7–23
  - loading for verification 6–12
  - naming 5–3
  - programming for MATLAB
    - verification 5–3
  - sample definition of 5–5
  - specifying ports for 5–4
  - using port information for 5–15
  - validating 5–15
  - verifying port direction modes for 5–15
- VHDL models 1–3
- See also* HDL models
  - adding to Simulink models 7–24
  - compiling 5–7
  - configuring Simulink for 7–17
  - cosimulation 1–3
  - debugging 5–7
  - porting 7–43
  - running in Simulink 7–42
  - testing in Simulink 7–42
  - verifying 1–3
- VHDL Sink block 7–24
- See also* VHDL Cosimulation block
- VHDL Source block 7–24
- See also* VHDL Cosimulation block
- visualization
- coding functions for 5–8
  - overview of 5–8
- vsim function
- description of 8–9
  - starting ModelSim with 6–10 7–20
- vsimdir property
- specifying with setupmodelsim function 7–20
  - specifying with vsim function 6–10
  - with setupmodelsim function 8–6
  - with vsim function 8–9
- vsimmatlab command
- description of 9–11
  - loading VHDL entities for verification with 6–12
- vsimulink command
- description of 9–12

loading VHDL entities for cosimulation  
with 7-23

## **W**

Wave window, ModelSim 7-29  
waveform files 7-43  
which function 5-28  
Windows 2000 1-5

as a required platform 1-18  
Windows XP 1-5  
as a required platform 1-18  
WLF files 7-43  
wrapverilog command 9-14

## **Z**

zero-order hold 7-9